# Hyperdual Numbers and Forward Differentiation

Filip Smola, Jacques Fleuriot

January 5, 2022

**Abstract**

Hyperdual numbers are ones with a real component and a number of infinitesimal components, usually written as $a_0 + a_1 \cdot \epsilon_1 + a_2 \cdot \epsilon_2 + a_3 \cdot \epsilon_1\epsilon_2$. They have been proposed by Fike and Alonso [1] in an approach to automatic differentiation.

In this entry we formalise hyperdual numbers and their application to forward differentiation. We show them to be an instance of multiple algebraic structures and then, along with facts about twice-differentiability, we define what we call the hyperdual extensions of functions on real-normed fields. This extension formally represents the proposed way that the first and second derivatives of a function can be automatically calculated. We demonstrate it on the standard logistic function $f(x) = \frac{1}{1+e^{-x}}$ and also reproduce the example analytic function $f(x) = \frac{e^x}{\sqrt{sin(x)^3+cos(x)^3}}$ used for demonstration by Fike and Alonso.

# Contents

**theory** *Hyperdual*
**imports** *HOL−Analysis.Analysis*
**begin**

# 1 Hyperdual Numbers

Let $\tau$ be some type. Second-order hyperdual numbers over $\tau$ take the form $a_1 + a_2\varepsilon_1 + a_3\varepsilon_2 + a_4\varepsilon_1\varepsilon_2$ where all $a_i :: \tau$, and $\varepsilon_1$ and $\varepsilon_2$ are non-zero but nilpotent infinitesimals: ${\varepsilon_1}^2 = {\varepsilon_2}^2 = (\varepsilon_1\varepsilon_2)^2 = 0$.

We define second-order hyperdual numbers as a coinductive data type with four components: the base component, two first-order hyperdual components and one second-order hyperdual component.

**codatatype** *'a hyperdual* = *Hyperdual* (*Base*: *'a*) (*Eps1*: *'a*) (*Eps2*: *'a*) (*Eps12*: *'a*)

Two hyperduals are equal iff all their components are equal.

**lemma** *hyperdual-eq-iff* [*iff*]:
  $x = y \longleftrightarrow ((Base\ x = Base\ y) \wedge (Eps1\ x = Eps1\ y) \wedge (Eps2\ x = Eps2\ y) \wedge (Eps12\ x = Eps12\ y))$
  **using** *hyperdual.expand* **by** *auto*

**lemma** *hyperdual-eqI*:
  **assumes** *Base x* = *Base y*
    **and** *Eps1 x* = *Eps1 y*
    **and** *Eps2 x* = *Eps2 y*
    **and** *Eps12 x* = *Eps12 y*
   **shows** *x* = *y*
  **by** (*simp add*: *assms*)

The embedding from the component type to hyperduals requires the component type to have a zero element.

**definition** *of-comp* :: (*'a* :: *zero*) ⇒ *'a hyperdual*
  **where** *of-comp a* = *Hyperdual a 0 0 0*

**lemma** *of-comp-simps* [*simp*]:
  *Base* (*of-comp a*) = *a*
  *Eps1* (*of-comp a*) = *0*
  *Eps2* (*of-comp a*) = *0*
  *Eps12* (*of-comp a*) = *0*
  **by** (*simp-all add*: *of-comp-def*)

## 1.1 Addition and Subtraction

We define hyperdual addition, subtraction and unary minus pointwise, and zero by embedding.

**instantiation** *hyperdual* :: (*plus*) *plus*
**begin**

**primcorec** *plus-hyperdual*
  **where**
    *Base* (*x* + *y*) = *Base x* + *Base y*
  | *Eps1* (*x* + *y*) = *Eps1 x* + *Eps1 y*

| *Eps2* (*x* + *y*) = *Eps2 x* + *Eps2 y*
| *Eps12* (*x* + *y*) = *Eps12 x* + *Eps12 y*

**instance by** *standard*
**end**

**instantiation** *hyperdual* :: (*zero*) *zero*
**begin**

**definition** *zero-hyperdual*
**where** *0* = *of-comp 0*

**instance by** *standard*
**end**

**lemma** *zero-hyperdual-simps* [*simp*]:
*Base 0* = *0*
*Eps1 0* = *0*
*Eps2 0* = *0*
*Eps12 0* = *0*
*Hyperdual 0 0 0 0* = *0*
**by** (*simp-all add*: *zero-hyperdual-def*)

**instantiation** *hyperdual* :: (*uminus*) *uminus*
**begin**

**primcorec** *uminus-hyperdual*
**where**
*Base* (−*x*) = − *Base x*
| *Eps1* (−*x*) = − *Eps1 x*
| *Eps2* (−*x*) = − *Eps2 x*
| *Eps12* (−*x*) = − *Eps12 x*

**instance by** *standard*
**end**

**instantiation** *hyperdual* :: (*minus*) *minus*
**begin**

**primcorec** *minus-hyperdual*
**where**
*Base* (*x* − *y*) = *Base x* − *Base y*
| *Eps1* (*x* − *y*) = *Eps1 x* − *Eps1 y*
| *Eps2* (*x* − *y*) = *Eps2 x* − *Eps2 y*
| *Eps12* (*x* − *y*) = *Eps12 x* − *Eps12 y*

**instance by** *standard*
**end**

If the components form a commutative group under addition then so do the hyperduals.

**instance** *hyperdual* :: (*semigroup-add*) *semigroup-add*
  **by** *standard* (*simp add*: *add.assoc*)

**instance** *hyperdual* :: (*monoid-add*) *monoid-add*
  **by** *standard simp-all*

**instance** *hyperdual* :: (*ab-semigroup-add*) *ab-semigroup-add*
  **by** *standard* (*simp-all add*: *add.commute*)

**instance** *hyperdual* :: (*comm-monoid-add*) *comm-monoid-add*
  **by** *standard simp*

**instance** *hyperdual* :: (*group-add*) *group-add*
  **by** *standard simp-all*

**instance** *hyperdual* :: (*ab-group-add*) *ab-group-add*
  **by** *standard simp-all*

**lemma** *of-comp-add*:
  **fixes** *a b* :: *'a* :: *monoid-add*
  **shows** *of-comp* (*a* + *b*) = *of-comp a* + *of-comp b*
  **by** *simp*

**lemma**
  **fixes** *a b* :: *'a* :: *group-add*
  **shows** *of-comp-minus*: *of-comp* (− *a*) = − *of-comp a*
    **and** *of-comp-diff*: *of-comp* (*a* − *b*) = *of-comp a* − *of-comp b*
  **by** *simp-all*

## 1.2 Multiplication and Scaling

Multiplication of hyperduals is defined by distributing the expressions and using the nilpotence of $\varepsilon_1$ and $\varepsilon_2$, resulting in the definition used here. The hyperdual one is again defined by embedding.

**instantiation** *hyperdual* :: ({*one*, *zero*}) *one*
**begin**

**definition** *one-hyperdual*
  **where** *1* = *of-comp 1*

**instance by** *standard*
**end**

**lemma** *one-hyperdual-simps* [*simp*]:
*Base 1* = *1*
*Eps1 1* = *0*
*Eps2 1* = *0*
*Eps12 1* = *0*
*Hyperdual 1 0 0 0* = *1*
**by** (*simp-all add*: *one-hyperdual-def*)

**instantiation** *hyperdual* :: ({*times*, *plus*}) *times*
**begin**

**primcorec** *times-hyperdual*
**where**
*Base* (*x* ∗ *y*) = *Base x* ∗ *Base y*
| *Eps1* (*x* ∗ *y*) = (*Base x* ∗ *Eps1 y*) + (*Eps1 x* ∗ *Base y*)
| *Eps2* (*x* ∗ *y*) = (*Base x* ∗ *Eps2 y*) + (*Eps2 x* ∗ *Base y*)
| *Eps12* (*x* ∗ *y*) = (*Base x* ∗ *Eps12 y*) + (*Eps1 x* ∗ *Eps2 y*) + (*Eps2 x* ∗ *Eps1 y*) + (*Eps12 x* ∗ *Base y*)

**instance by** *standard*
**end**

If the components form a ring then so do the hyperduals.

**instance** *hyperdual* :: (*semiring*) *semiring*
**by** *standard* (*simp-all add*: *mult.assoc distrib-left distrib-right add.assoc add.left-commute*)

**instance** *hyperdual* :: ({*monoid-add*, *mult-zero*}) *mult-zero*
**by** *standard simp-all*

**instance** *hyperdual* :: (*ring*) *ring*
**by** *standard*

**instance** *hyperdual* :: (*comm-ring*) *comm-ring*
**by** *standard* (*simp-all add*: *mult.commute distrib-left*)

**instance** *hyperdual* :: (*ring-1*) *ring-1*
**by** *standard simp-all*

**instance** *hyperdual* :: (*comm-ring-1*) *comm-ring-1*
**by** *standard simp*

**lemma** *of-comp-times*:
**fixes** *a b* :: *'a* :: *semiring-0*
**shows** *of-comp* (*a* ∗ *b*) = *of-comp a* ∗ *of-comp b*
**by** (*simp add*: *of-comp-def times-hyperdual.code*)

Hyperdual scaling is multiplying each component by a factor from the component type.

**primcorec** *scaleH* :: (*'a* :: *times*) ⇒ *'a hyperdual* ⇒ *'a hyperdual* (**infixr** $*_H$ *75*)
**where**
*Base* ($f *_H x$) = $f$ ∗ *Base* $x$
| *Eps1* ($f *_H x$) = $f$ ∗ *Eps1* $x$
| *Eps2* ($f *_H x$) = $f$ ∗ *Eps2* $x$
| *Eps12* ($f *_H x$) = $f$ ∗ *Eps12* $x$

**lemma** *scaleH-times*:
**fixes** $f$ :: *'a* :: {*monoid-add*, *mult-zero*}
**shows** $f *_H x$ = *of-comp* $f * x$
**by** *simp*

**lemma** *scaleH-add*:
**fixes** $a$ :: *'a* :: *semiring*
**shows** $(a + a') *_H b = a *_H b + a' *_H b$
**and** $a *_H (b + b') = a *_H b + a *_H b'$
**by** (*simp-all add*: *distrib-left distrib-right*)

**lemma** *scaleH-diff*:
**fixes** $a$ :: *'a* :: *ring*
**shows** $(a - a') *_H b = a *_H b - a' *_H b$
**and** $a *_H (b - b') = a *_H b - a *_H b'$
**by** (*auto simp add*: *left-diff-distrib right-diff-distrib scaleH-times of-comp-diff*)

**lemma** *scaleH-mult*:
**fixes** $a$ :: *'a* :: *semigroup-mult*
**shows** $(a * a') *_H b = a *_H a' *_H b$
**by** (*simp add*: *mult.assoc*)

**lemma** *scaleH-one* [*simp*]:
**fixes** $b$ :: (*'a* :: *monoid-mult*) *hyperdual*
**shows** $1 *_H b = b$
**by** *simp*

**lemma** *scaleH-zero* [*simp*]:
**fixes** $b$ :: (*'a* :: {*mult-zero*, *times*}) *hyperdual*
**shows** $0 *_H b = 0$
**by** *simp*

**lemma**
**fixes** $b$ :: (*'a* :: *ring-1*) *hyperdual*
**shows** *scaleH-minus* [*simp*]:$- 1 *_H b = - b$
**and** *scaleH-minus-left*: $- (a *_H b) = - a *_H b$
**and** *scaleH-minus-right*: $- (a *_H b) = a *_H - b$
**by** *simp-all*

Induction rule for natural numbers that takes 0 and 1 as base cases.

**lemma** *nat-induct01Suc*[*case-names 0 1 Suc*]:
**assumes** *P 0*

**and** *P 1*
**and** $\bigwedge n.\ n > 0 \Longrightarrow P\ n \Longrightarrow P\ (Suc\ n)$
**shows** *P n*
**by** (*metis One-nat-def assms nat-induct neq0-conv*)

**lemma** *hyperdual-power*:
**fixes** *x* :: (*'a* :: *comm-ring-1*) *hyperdual*
**shows** *x ^ n = Hyperdual* ((*Base x*) ^ *n*)
(*Eps1 x* ∗ *of-nat n* ∗ (*Base x*) ^ (*n* − *1*))
(*Eps2 x* ∗ *of-nat n* ∗ (*Base x*) ^ (*n* − *1*))
(*Eps12 x* ∗ *of-nat n* ∗ (*Base x*) ^ (*n* − *1*) + *Eps1 x* ∗ *Eps2 x* ∗ *of-nat n* ∗ *of-nat* (*n* − *1*) ∗ (*Base x*) ^ (*n* − *2*))
**proof** (*induction n rule*: *nat-induct01Suc*)
**case** *0*
**show** *?case*
**by** *simp*
**next**
**case** *1*
**show** *?case*
**by** *simp*
**next**
**case** *hyp*: (*Suc n*)
**show** *?case*
**proof** (*simp add*: *hyp*, *intro conjI*)
**show** *Base x* ∗ (*Eps1 x* ∗ *of-nat n* ∗ *Base x* ^ (*n* − *Suc 0*)) + *Eps1 x* ∗ *Base x* ^ *n* = *Eps1 x* ∗ (*1* + *of-nat n*) ∗ *Base x* ^ *n*
**and** *Base x* ∗ (*Eps2 x* ∗ *of-nat n* ∗ *Base x* ^ (*n* − *Suc 0*)) + *Eps2 x* ∗ *Base x* ^ *n* = *Eps2 x* ∗ (*1* + *of-nat n*) ∗ *Base x* ^ *n*
**by** (*simp-all add*: *distrib-left distrib-right power-eq-if*)
**show**
*2* ∗ (*Eps1 x* ∗ (*Eps2 x* ∗ (*of-nat n* ∗ *Base x* ^ (*n* − *Suc 0*)))) +
*Base x* ∗ (*Eps12 x* ∗ *of-nat n* ∗ *Base x* ^ (*n* − *Suc 0*) + *Eps1 x* ∗ *Eps2 x* ∗ *of-nat n* ∗ *of-nat* (*n* − *Suc 0*) ∗ *Base x* ^ (*n* − *2*)) +
*Eps12 x* ∗ *Base x* ^ *n* =
*Eps12 x* ∗ (*1* + *of-nat n*) ∗ *Base x* ^ *n* + *Eps1 x* ∗ *Eps2 x* ∗ (*1* + *of-nat n*) ∗ *of-nat n* ∗ *Base x* ^ (*n* − *Suc 0*)
**proof** −
**have**
*2* ∗ (*Eps1 x* ∗ (*Eps2 x* ∗ (*of-nat n* ∗ *Base x* ^ (*n* − *Suc 0*)))) +
*Base x* ∗ (*Eps12 x* ∗ *of-nat n* ∗ *Base x* ^ (*n* − *Suc 0*) + *Eps1 x* ∗ *Eps2 x* ∗ *of-nat n* ∗ *of-nat* (*n* − *Suc 0*) ∗ *Base x* ^ (*n* − *2*)) +
*Eps12 x* ∗ *Base x* ^ *n* =
*2* ∗ *Eps1 x* ∗ *Eps2 x* ∗ *of-nat n* ∗ *Base x* ^ (*n* − *Suc 0*) +
*Eps12 x* ∗ *of-nat* (*n* + *1*) ∗ *Base x* ^ *n* + *Eps1 x* ∗ *Eps2 x* ∗ *of-nat n* ∗ *of-nat* (*n* − *Suc 0*) ∗ *Base x* ^ (*n* − *Suc 0*)
**by** (*simp add*: *field-simps power-eq-if*)
**also have** ... = *Eps12 x* ∗ *of-nat* (*n* + *1*) ∗ *Base x* ^ *n* + *of-nat* (*n* − *1* + *2*) ∗ *Eps1 x* ∗ *Eps2 x* ∗ *of-nat n* ∗ *Base x* ^ (*n* − *Suc 0*)
**by** (*simp add*: *distrib-left mult.commute*)

```
      finally show ?thesis
        by (simp add: hyp.hyps)
    qed
  qed
qed

lemma hyperdual-power-simps [simp]:
  shows Base ((x :: 'a :: comm-ring-1 hyperdual) ^ n) = Base x ^ n
    and Eps1 ((x :: 'a :: comm-ring-1 hyperdual) ^ n) = Eps1 x * of-nat n * (Base
x) ^ (n − 1)
    and Eps2 ((x :: 'a :: comm-ring-1 hyperdual) ^ n) = Eps2 x * of-nat n * (Base
x) ^ (n − 1)
    and Eps12 ((x :: 'a :: comm-ring-1 hyperdual) ^ n) =
  (Eps12 x * of-nat n * (Base x) ^ (n − 1) + Eps1 x * Eps2 x * of-nat n * of-nat
(n − 1) * (Base x) ^ (n − 2))
  by (simp-all add: hyperdual-power)
```

Squaring the hyperdual one behaves as expected from the reals.

```
lemma hyperdual-square-eq-1-iff [iff]:
  fixes x :: ('a :: {real-div-algebra, comm-ring}) hyperdual
  shows x * x = 1 ⟷ x = 1 ∨ x = − 1
proof
  assume a: x * x = 1

  have base: Base x * Base x = 1
    using a by simp
  moreover have e1: Eps1 x = 0
  proof −
    have Base x * Eps1 x = − (Base x * Eps1 x)
    using mult.commute[of Base x] add-eq-0-iff[of Base x * Eps1 x] times-hyperdual.simps(2)[of
x x]
      by (simp add: a)
    then have Base x * Base x * Eps1 x = − Base x * Base x * Eps1 x
      using mult-left-cancel[of Base x] base by fastforce
    then show ?thesis
      using base mult-right-cancel[of Eps1 x Base x * Base x − Base x * Base x]
one-neq-neg-one
      by auto
  qed
  moreover have e2: Eps2 x = 0
  proof −
    have Base x * Eps2 x = − (Base x * Eps2 x)
      using a mult.commute[of Base x Eps2 x] add-eq-0-iff[of Base x * Eps2 x]
times-hyperdual.simps(3)[of x x]
      by simp
    then have Base x * Base x * Eps2 x = − Base x * Base x * Eps2 x
      using mult-left-cancel[of Base x] base by fastforce
    then show ?thesis
      using base mult-right-cancel[of Eps2 x Base x * Base x − Base x * Base x]
```

*one-neq-neg-one*
    **by** *auto*
  **qed**
  **moreover have** *Eps12 x = 0*
  **proof** –
    **have** *Base x ∗ Eps12 x = − (Base x ∗ Eps12 x)*
      **using** *a e1 e2 mult.commute[of Base x Eps12 x] add-eq-0-iff [of Base x ∗ Eps12 x] times-hyperdual.simps(4)[of x x]*
      **by** *simp*
    **then have** *Base x ∗ Base x ∗ Eps12 x = − Base x ∗ Base x ∗ Eps12 x*
      **using** *mult-left-cancel[of Base x] base* **by** *fastforce*
    **then show** *?thesis*
      **using** *base mult-right-cancel[of Eps12 x Base x ∗ Base x − Base x ∗ Base x] one-neq-neg-one*
      **by** *auto*
  **qed**
  **ultimately show** *x = 1 ∨ x = − 1*
    **using** *square-eq-1-iff [of Base x]* **by** *simp*
**next**
  **assume** *x = 1 ∨ x = − 1*
  **then show** *x ∗ x = 1*
    **by** *(simp, safe, simp-all)*
**qed**

### 1.2.1 Properties of Zero Divisors

Unlike the reals, hyperdual numbers may have non-trivial divisors of zero as we show below.

First, if the components have no non-trivial zero divisors then that behaviour is preserved on the base component.

**lemma** *divisors-base-zero*:
  **fixes** *a b :: ('a :: ring-no-zero-divisors) hyperdual*
  **assumes** *Base (a ∗ b) = 0*
  **shows** *Base a = 0 ∨ Base b = 0*
  **using** *assms* **by** *auto*
**lemma** *hyp-base-mult-eq-0-iff [iff]*:
  **fixes** *a b :: ('a :: ring-no-zero-divisors) hyperdual*
  **shows** *Base (a ∗ b) = 0 ⟷ Base a = 0 ∨ Base b = 0*
  **by** *simp*

However, the conditions are relaxed on the full hyperdual numbers. This is due to some terms vanishing in the multiplication and thus not constraining the result.

**lemma** *divisors-hyperdual-zero [iff]*:
  **fixes** *a b :: ('a :: ring-no-zero-divisors) hyperdual*
  **shows** *a ∗ b = 0 ⟷ (a = 0 ∨ b = 0 ∨ (Base a = 0 ∧ Base b = 0 ∧ Eps1 a ∗ Eps2 b = − Eps2 a ∗ Eps1 b))*

**proof**
  **assume** *mult*: $a * b = 0$
  **then have** *split*: $Base\ a = 0 \lor Base\ b = 0$
    **by** *simp*
  **show** $a = 0 \lor b = 0 \lor Base\ a = 0 \land Base\ b = 0 \land Eps1\ a * Eps2\ b = -\ Eps2$
$a * Eps1\ b$
  **proof** (*cases* $Base\ a = 0$)
    **case** *aT*: *True*
    **then show** *?thesis*
    **proof** (*cases* $Base\ b = 0$)
      — $Base\ a = (0::'a) \land Base\ b = (0::'a)$
      **case** *bT*: *True*
      **then have** $Eps12\ (a * b) = Eps1\ a * Eps2\ b + Eps2\ a * Eps1\ b$
        **by** (*simp add*: *aT*)
      **then show** *?thesis*
        **by** (*simp add*: *aT bT mult eq-neg-iff-add-eq-0*)
    **next**
      — $Base\ a = (0::'a) \land Base\ b \neq (0::'a)$
      **case** *bF*: *False*
      **then have** *e1*: $Eps1\ a = 0$
      **proof** −
        **have** $Eps1\ (a * b) = Eps1\ a * Base\ b$
          **by** (*simp add*: *aT*)
        **then show** *?thesis*
          **by** (*simp add*: *bF mult*)
      **qed**
      **moreover have** *e2*: $Eps2\ a = 0$
      **proof** −
        **have** $Eps2\ (a * b) = Eps2\ a * Base\ b$
          **by** (*simp add*: *aT*)
        **then show** *?thesis*
          **by** (*simp add*: *bF mult*)
      **qed**
      **moreover have** $Eps12\ a = 0$
      **proof** −
        **have** $Eps12\ (a * b) = Eps1\ a * Eps2\ b + Eps2\ a * Eps1\ b$
          **by** (*simp add*: *e1 e2 mult*)
        **then show** *?thesis*
          **by** (*simp add*: *aT bF*)
      **qed**
      **ultimately show** *?thesis*
        **by** (*simp add*: *aT*)
    **qed**
  **next**
    **case** *aF*: *False*
    **then show** *?thesis*
    **proof** (*cases* $Base\ b = 0$)
      — $Base\ a \neq (0::'a) \land Base\ b = (0::'a)$
      **case** *bT*: *True*

**then have** *e1*: *Eps1 b = 0*
**proof** –
**have** *Eps1* (*a* ∗ *b*) = *Base a* ∗ *Eps1 b*
**by** (*simp add*: *bT*)
**then show** *?thesis*
**by** (*simp add*: *aF mult*)
**qed**
**moreover have** *e2*: *Eps2 b = 0*
**proof** –
**have** *Eps2* (*a* ∗ *b*) = *Base a* ∗ *Eps2 b*
**by** (*simp add*: *bT*)
**then show** *?thesis*
**by** (*simp add*: *aF mult*)
**qed**
**moreover have** *Eps12 b = 0*
**proof** –
**have** *Eps12* (*a* ∗ *b*) = *Eps1 a* ∗ *Eps2 b* + *Eps2 a* ∗ *Eps1 b*
**by** (*simp add*: *e1 e2 mult*)
**then show** *?thesis*
**by** (*simp add*: *bT aF*)
**qed**
**ultimately show** *?thesis*
**by** (*simp add*: *bT*)
**next**
— *Base a* ≠ (*0*::′*a*) ∧ *Base b* ≠ (*0*::′*a*)
**case** *bF*: *False*
**then have** *False*
**using** *split aF* **by** *blast*
**then show** *?thesis*
**by** *simp*
**qed**
**qed**
**next**
**show** *a* = *0* ∨ *b* = *0* ∨ *Base a* = *0* ∧ *Base b* = *0* ∧ *Eps1 a* ∗ *Eps2 b* = − *Eps2 a* ∗ *Eps1 b* ⟹ *a* ∗ *b* = *0*
**by** (*simp*, *auto*)
**qed**

### 1.2.2 Multiplication Cancellation

Similarly to zero divisors, multiplication cancellation rules for hyperduals are not exactly the same as those for reals.

First, cancelling a common factor has a relaxed condition compared to reals. It only requires the common factor to have base component zero, instead of requiring the whole number to be zero.

**lemma** *hyp-mult-left-cancel* [*iff*]:
**fixes** *a b c* :: (′*a* :: *ring-no-zero-divisors*) *hyperdual*
**assumes** *baseC*: *Base c* ≠ *0*

**shows** $c * a = c * b \longleftrightarrow a = b$
**proof**
**assume** *mult*: $c * a = c * b$
**show** $a = b$
**proof** (*simp*, *safe*)
**show** *base*: *Base a* = *Base b*
**using** *mult mult-cancel-left baseC* **by** *simp-all*
**show** *Eps1 a* = *Eps1 b*
**and** *Eps2 a* = *Eps2 b*
**using** *mult base mult-cancel-left baseC* **by** *simp-all*
**then show** *Eps12 a* = *Eps12 b*
**using** *mult base mult-cancel-left baseC* **by** *simp-all*
**qed**
**next**
**show** $a = b \Longrightarrow c * a = c * b$
**by** *simp*
**qed**

**lemma** *hyp-mult-right-cancel* [*iff*]:
**fixes** *a b c* :: (*'a* :: *ring-no-zero-divisors*) *hyperdual*
**assumes** *baseC*: *Base c* ≠ *0*
**shows** $a * c = b * c \longleftrightarrow a = b$
**proof**
**assume** *mult*: $a * c = b * c$
**show** $a = b$
**proof** (*simp*, *safe*)
**show** *base*: *Base a* = *Base b*
**using** *mult mult-cancel-left baseC* **by** *simp-all*
**show** *Eps1 a* = *Eps1 b*
**and** *Eps2 a* = *Eps2 b*
**using** *mult base mult-cancel-left baseC* **by** *simp-all*
**then show** *Eps12 a* = *Eps12 b*
**using** *mult base mult-cancel-left baseC* **by** *simp-all*
**qed**
**next**
**show** $a = b \Longrightarrow a * c = b * c$
**by** *simp*
**qed**

Next, when a factor absorbs another there are again relaxed conditions compared to reals. For reals, either the absorbing factor is zero or the absorbed is the unit. However, with hyperduals there are more possibilities again due to terms vanishing during the multiplication.

**lemma** *hyp-mult-cancel-right1* [*iff*]:
**fixes** *a b* :: (*'a* :: *ring-1-no-zero-divisors*) *hyperdual*
**shows** $a = b * a \longleftrightarrow a = 0 \lor b = 1 \lor$ (*Base a* = *0* ∧ *Base b* = *1* ∧ *Eps1 b* ∗ *Eps2 a* = − *Eps2 b* ∗ *Eps1 a*)
**proof**
**assume** *mult*: $a = b * a$

**show** *a* = *0* ∨ *b* = *1* ∨ (*Base a* = *0* ∧ *Base b* = *1* ∧ *Eps1 b* ∗ *Eps2 a* = − *Eps2 b* ∗ *Eps1 a*)
**proof** (*cases Base a* = *0*)
**case** *aT*: *True*
**then show** *?thesis*
**proof** (*cases Base b* = *1*)
— *Base a* = (*0*::′*a*) ∧ *Base b* = (*1*::′*a*)
**case** *bT*: *True*
**then show** *?thesis*
**using** *aT mult add-cancel-right-right add-eq-0-iff*[*of Eps1 b* ∗ *Eps2 a*] *times-hyperdual.simps*(*4*)[*of b a*]
**by** *simp*
**next**
— *Base a* = (*0*::′*a*) ∧ *Base b* ≠ (*1*::′*a*)
**case** *bF*: *False*
**then show** *?thesis*
**using** *aT mult* **by** (*simp*, *auto*)
**qed**
**next**
**case** *aF*: *False*
**then show** *?thesis*
**proof** (*cases Base b* = *1*)
— *Base a* ≠ (*0*::′*a*) ∧ *Base b* = (*1*::′*a*)
**case** *bT*: *True*
**then show** *?thesis*
**using** *aF mult* **by** (*simp*, *auto*)
**next**
— *Base a* ≠ (*0*::′*a*) ∧ *Base b* ≠ (*1*::′*a*)
**case** *bF*: *False*
**then show** *?thesis*
**using** *aF mult* **by** *simp*
**qed**
**qed**
**next**
**have** *a* = *0* ⟹ *a* = *b* ∗ *a*
**and** *b* = *1* ⟹ *a* = *b* ∗ *a*
**by** *simp-all*
**moreover have** *Base a* = *0* ∧ *Base b* = *1* ∧ *Eps1 b* ∗ *Eps2 a* = − *Eps2 b* ∗ *Eps1 a* ⟹ *a* = *b* ∗ *a*
**by** *simp*
**ultimately show** *a* = *0* ∨ *b* = *1* ∨ (*Base a* = *0* ∧ *Base b* = *1* ∧ *Eps1 b* ∗ *Eps2 a* = − *Eps2 b* ∗ *Eps1 a*) ⟹ *a* = *b* ∗ *a*
**by** *blast*
**qed**
**lemma** *hyp-mult-cancel-right2* [*iff*]:
**fixes** *a b* :: (′*a* :: *ring-1-no-zero-divisors*) *hyperdual*
**shows** *b* ∗ *a* = *a* ⟷ *a* = *0* ∨ *b* = *1* ∨ (*Base a* = *0* ∧ *Base b* = *1* ∧ *Eps1 b* ∗ *Eps2 a* = − *Eps2 b* ∗ *Eps1 a*)
**using** *hyp-mult-cancel-right1* **by** *smt*

**lemma** *hyp-mult-cancel-left1* [*iff*]:
**fixes** *a b* :: (*'a* :: *ring-1-no-zero-divisors*) *hyperdual*
**shows** $a = a * b \longleftrightarrow a = 0 \vee b = 1 \vee (Base\ a = 0 \wedge Base\ b = 1 \wedge Eps1\ a * Eps2\ b = -\ Eps2\ a * Eps1\ b)$
**proof**
**assume** *mult*: $a = a * b$
**show** $a = 0 \vee b = 1 \vee (Base\ a = 0 \wedge Base\ b = 1 \wedge Eps1\ a * Eps2\ b = -\ Eps2\ a * Eps1\ b)$
**proof** (*cases* $Base\ a = 0$)
**case** *aT*: *True*
**then show** *?thesis*
**proof** (*cases* $Base\ b = 1$)
— $Base\ a = (0::'a) \wedge Base\ b = (1::'a)$
**case** *bT*: *True*
**then show** *?thesis*
**using** *aT mult add-cancel-right-right add-eq-0-iff*[*of Eps1 a* $*$ *Eps2 b*] *times-hyperdual.simps*(*4*)[*of a b*]
**by** *simp*
**next**
— $Base\ a = (0::'a) \wedge Base\ b \neq (1::'a)$
**case** *bF*: *False*
**then show** *?thesis*
**using** *aT mult* **by** (*simp*, *auto*)
**qed**
**next**
**case** *aF*: *False*
**then show** *?thesis*
**proof** (*cases* $Base\ b = 1$)
— $Base\ a \neq (0::'a) \wedge Base\ b = (1::'a)$
**case** *bT*: *True*
**then show** *?thesis*
**using** *aF mult* **by** (*simp*, *auto*)
**next**
— $Base\ a \neq (0::'a) \wedge Base\ b \neq (1::'a)$
**case** *bF*: *False*
**then show** *?thesis*
**using** *aF mult* **by** *simp*
**qed**
**qed**
**next**
**have** $a = 0 \Longrightarrow a = a * b$
**and** $b = 1 \Longrightarrow a = a * b$
**by** *simp-all*
**moreover have** $Base\ a = 0 \wedge Base\ b = 1 \wedge Eps1\ a * Eps2\ b = -\ Eps2\ a * Eps1\ b \Longrightarrow a = a * b$
**by** *simp*
**ultimately show** $a = 0 \vee b = 1 \vee (Base\ a = 0 \wedge Base\ b = 1 \wedge Eps1\ a * Eps2\ b = -\ Eps2\ a * Eps1\ b) \Longrightarrow a = a * b$

**by** *blast*
**qed**
**lemma** *hyp-mult-cancel-left2* [*iff*]:
**fixes** *a b* :: (*'a* :: *ring-1-no-zero-divisors*) *hyperdual*
**shows** *a* ∗ *b* = *a* ⟷ *a* = *0* ∨ *b* = *1* ∨ (*Base a* = *0* ∧ *Base b* = *1* ∧ *Eps1 a* ∗ *Eps2 b* = − *Eps2 a* ∗ *Eps1 b*)
**using** *hyp-mult-cancel-left1* **by** *smt*

## 1.3 Multiplicative Inverse and Division

If the components form a ring with a multiplicative inverse then so do the hyperduals. The hyperdual inverse of *a* is defined as the solution to *a* ∗ *x* = (*1*::*'a*). Hyperdual division is then multiplication by divisor's inverse.

Each component of the inverse has as denominator a power of the base component. Therefore this inverse is only well defined for hyperdual numbers with non-zero base components.

**instantiation** *hyperdual* :: ({*inverse*, *ring-1*}) *inverse*
**begin**

**primcorec** *inverse-hyperdual*
**where**
*Base* (*inverse a*) = *1* / *Base a*
| *Eps1* (*inverse a*) = − *Eps1 a* / (*Base a*)^*2*
| *Eps2* (*inverse a*) = − *Eps2 a* / (*Base a*)^*2*
| *Eps12* (*inverse a*) = *2* ∗ (*Eps1 a* ∗ *Eps2 a* / (*Base a*)^*3*) − *Eps12 a* / (*Base a*)^*2*

**primcorec** *divide-hyperdual*
**where**
*Base* (*divide a b*) = *Base a* / *Base b*
| *Eps1* (*divide a b*) = (*Eps1 a* ∗ *Base b* − *Base a* ∗ *Eps1 b*) / ((*Base b*)^*2*)
| *Eps2* (*divide a b*) = (*Eps2 a* ∗ *Base b* − *Base a* ∗ *Eps2 b*) / ((*Base b*)^*2*)
| *Eps12* (*divide a b*) = (*2* ∗ *Base a* ∗ *Eps1 b* ∗ *Eps2 b* −
*Base a* ∗ *Base b* ∗ *Eps12 b* −
*Eps1 a* ∗ *Base b* ∗ *Eps2 b* −
*Eps2 a* ∗ *Base b* ∗ *Eps1 b* +
*Eps12 a* ∗ ((*Base b*)^*2*)) / ((*Base b*)^*3*)
**instance**
**by** *standard*
**end**

Because hyperduals have non-trivial zero divisors, they do not form a division ring and so we can't use the *division-ring* type class to establish properties of hyperdual division. However, if the components form a division ring as well as a commutative ring, we can prove some similar facts about hyperdual division inspired by *division-ring*.

Inverse is multiplicative inverse from both sides.

**lemma**
 **fixes** *a* :: (*'a* :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
 **assumes** *Base a* ≠ *0*
 **shows** *hyp-left-inverse* [*simp*]: *inverse a* ∗ *a* = *1*
  **and** *hyp-right-inverse* [*simp*]: *a* ∗ *inverse a* = *1*
 **by** (*simp-all add*: *assms power2-eq-square power3-eq-cube field-simps*)

Division is multiplication by inverse.

**lemma** *hyp-divide-inverse*:
 **fixes** *a b* :: (*'a* :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
 **shows** *a* / *b* = *a* ∗ *inverse b*
 **by** (*cases Base b* = *0* ; *simp add*: *field-simps power2-eq-square power3-eq-cube*)

Hyperdual inverse is zero when not well defined.

**lemma** *zero-base-zero-inverse*:
 **fixes** *a* :: (*'a* :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
 **assumes** *Base a* = *0*
 **shows** *inverse a* = *0*
 **by** (*simp add*: *assms*)

**lemma** *zero-inverse-zero-base*:
 **fixes** *a* :: (*'a* :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
 **assumes** *inverse a* = *0*
 **shows** *Base a* = *0*
 **using** *assms right-inverse hyp-left-inverse* **by** *force*

**lemma** *hyp-inverse-zero*:
 **fixes** *a* :: (*'a* :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
 **shows** (*inverse a* = *0*) = (*Base a* = *0*)
 **using** *zero-base-zero-inverse*[*of a*] *zero-inverse-zero-base*[*of a*] **by** *blast*

Inverse preserves invertibility.

**lemma** *hyp-invertible-inverse*:
 **fixes** *a* :: (*'a* :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
 **shows** (*Base a* = *0*) = (*Base* (*inverse a*) = *0*)
 **by** (*safe*, *simp-all add*: *divide-inverse*)

Inverse is the only number that satisfies the defining equation.

**lemma** *hyp-inverse-unique*:
 **fixes** *a b* :: (*'a* :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
 **assumes** *a* ∗ *b* = *1*
 **shows** *b* = *inverse a*
**proof** −
 **have** *Base a* ≠ *0*
  **using** *assms one-hyperdual-def of-comp-simps zero-neq-one hyp-base-mult-eq-0-iff*
**by** *smt*
 **then show** *?thesis*
  **by** (*metis assms hyp-right-inverse mult.left-commute mult.right-neutral*)

**qed**

Multiplicative inverse commutes with additive inverse.

**lemma** *hyp-minus-inverse-comm*:
  **fixes** *a* :: (*'a* :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
  **shows** *inverse* (− *a*) = − *inverse a*
**proof** (*cases Base a* = *0*)
  **case** *True*
  **then show** *?thesis*
    **by** (*simp add*: *zero-base-zero-inverse*)
**next**
  **case** *False*
  **then show** *?thesis*
    **by** (*simp*, *simp add*: *nonzero-minus-divide-right*)
**qed**

Inverse is an involution (only) where well defined. Counter-example for non-invertible is *Hyperdual* (*0*::*'a*) (*0*::*'a*) (*0*::*'a*) (*0*::*'a*) with inverse *Hyperdual* (*0*::*'a*) (*0*::*'a*) (*0*::*'a*) (*0*::*'a*) which then inverts to *Hyperdual* (*0*::*'a*) (*0*::*'a*) (*0*::*'a*) (*0*::*'a*).

**lemma** *hyp-inverse-involution*:
  **fixes** *a* :: (*'a* :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
  **assumes** *Base a* ≠ *0*
  **shows** *inverse* (*inverse a*) = *a*
  **by** (*metis assms hyp-inverse-unique hyp-right-inverse mult.commute*)

**lemma** *inverse-inverse-neq-Ex*:
  ∃ *a* :: (*'a* :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual* . *inverse* (*inverse a*) ≠ *a*
**proof** −
  **have** ∃ *a* :: *'a hyperdual* . *Base a* = *0* ∧ *a* ≠ *0*
    **by** (*metis* (*full-types*) *hyperdual.sel*(*1*) *hyperdual.sel*(*4*) *zero-neq-one*)
  **moreover have** ⋀*a* :: *'a hyperdual* . (*Base a* = *0* ∧ *a* ≠ *0*) ⟹ (*inverse* (*inverse a*) ≠ *a*)
    **using** *hyp-inverse-zero hyp-invertible-inverse* **by** *smt*
  **ultimately show** *?thesis*
    **by** *blast*
**qed**

Inverses of equal invertible numbers are equal. This includes the other direction by inverse preserving invertibility and being an involution.

From a different point of view, inverse is injective on invertible numbers. The other direction for is again by inverse preserving invertibility and being an involution.

**lemma** *hyp-inverse-injection*:
  **fixes** *a b* :: (*'a* :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
  **assumes** *Base a* ≠ *0*
      **and** *Base b* ≠ *0*

```
    shows (inverse a = inverse b) = (a = b)
  by (metis assms hyp-inverse-involution)
```

One is its own inverse.

```
lemma hyp-inverse-1 [simp]:
  inverse (1 :: ('a :: {inverse, comm-ring-1, division-ring}) hyperdual) = 1
  using hyp-inverse-unique mult.left-neutral by metis
```

Inverse distributes over multiplication (even when not well defined).

```
lemma hyp-inverse-mult-distrib:
  fixes a b :: ('a :: {inverse, comm-ring-1, division-ring}) hyperdual
  shows inverse (a ∗ b) = inverse b ∗ inverse a
proof (cases Base a = 0 ∨ Base b = 0)
  case True
  then show ?thesis
   by (metis hyp-base-mult-eq-0-iff mult-zero-left mult-zero-right zero-base-zero-inverse)
next
  case False
  then have a ∗ (b ∗ inverse b) ∗ inverse a = 1
    by simp
  then have a ∗ b ∗ (inverse b ∗ inverse a) = 1
    by (simp only: mult.assoc)
  thus ?thesis
    using hyp-inverse-unique[of a ∗ b (inverse b ∗ inverse a)] by simp
qed
```

We derive expressions for addition and subtraction of inverses.

```
lemma hyp-inverse-add:
  fixes a b :: ('a :: {inverse, comm-ring-1, division-ring}) hyperdual
  assumes Base a ≠ 0
      and Base b ≠ 0
    shows inverse a + inverse b = inverse a ∗ (a + b) ∗ inverse b
  by (simp add: assms distrib-left mult.commute semiring-normalization-rules(18)
add.commute)

lemma hyp-inverse-diff:
  fixes a b :: ('a :: {inverse, comm-ring-1, division-ring}) hyperdual
  assumes a: Base a ≠ 0
      and b: Base b ≠ 0
    shows inverse a − inverse b = inverse a ∗ (b − a) ∗ inverse b
proof −
  have x: inverse a − inverse b = inverse b ∗ (inverse a ∗ b − 1)
    by (simp add: b mult.left-commute right-diff-distrib')
  show ?thesis
    by (simp add: x a mult.commute right-diff-distrib')
qed
```

Division is one only when dividing by self.

```
lemma hyp-divide-self:
```

**fixes** *a b* :: (*'a* :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
**assumes** *Base b* ≠ *0*
**shows** *a* / *b* = *1* ⟷ *a* = *b*
**by** (*metis assms hyp-divide-inverse hyp-inverse-unique hyp-right-inverse mult.commute*)

Taking inverse is the same as division of one, even when not invertible.

**lemma** *hyp-inverse-divide-1* [*divide-simps*]:
**fixes** *a* :: (*'a* :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
**shows** *inverse a* = *1* / *a*
**by** (*simp add*: *hyp-divide-inverse*)

Division distributes over addition and subtraction.

**lemma** *hyp-add-divide-distrib*:
**fixes** *a b c* :: (*'a* :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
**shows** (*a* + *b*) / *c* = *a*/*c* + *b*/*c*
**by** (*simp add*: *distrib-right hyp-divide-inverse*)

**lemma** *hyp-diff-divide-distrib*:
**fixes** *a b c* :: (*'a* :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
**shows** (*a* − *b*) / *c* = *a* / *c* − *b* / *c*
**by** (*simp add*: *left-diff-distrib hyp-divide-inverse*)

Multiplication associates with division.

**lemma** *hyp-times-divide-assoc* [*simp*]:
**fixes** *a b c* :: (*'a* :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
**shows** *a* ∗ (*b* / *c*) = (*a* ∗ *b*) / *c*
**by** (*simp add*: *hyp-divide-inverse mult.assoc*)

Additive inverse commutes with division, because it is multiplication by inverse.

**lemma** *hyp-divide-minus-left* [*simp*]:
**fixes** *a b* :: (*'a* :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
**shows** (−*a*) / *b* = − (*a* / *b*)
**by** (*simp add*: *hyp-divide-inverse*)

**lemma** *hyp-divide-minus-right* [*simp*]:
**fixes** *a b* :: (*'a* :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
**shows** *a* / (−*b*) = − (*a* / *b*)
**by** (*simp add*: *hyp-divide-inverse hyp-minus-inverse-comm*)

Additive inverses on both sides of division cancel out.

**lemma** *hyp-minus-divide-minus*:
**fixes** *a b* :: (*'a* :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
**shows** (−*a*) / (−*b*) = *a* / *b*
**by** *simp*

We can multiply both sides of equations by an invertible denominator.

**lemma** *hyp-denominator-eliminate* [*divide-simps*]:

**fixes** *a b c* :: ('a :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
**assumes** *Base* $c \neq 0$
**shows** $a = b / c \longleftrightarrow a * c = b$
**by** (*metis assms hyp-divide-self hyp-times-divide-assoc mult.commute mult.right-neutral*)

We can move addition and subtraction to a common denominator in the following ways:

**lemma** *hyp-add-divide-eq-iff*:
**fixes** *x y z* :: ('a :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
**assumes** *Base* $z \neq 0$
**shows** $x + y / z = (x * z + y) / z$
**by** (*metis assms hyp-add-divide-distrib hyp-denominator-eliminate*)

Result of division by non-invertible number is not invertible.

**lemma** *hyp-divide-base-zero* [*simp*]:
**fixes** *a b* :: ('a :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
**assumes** *Base* $b = 0$
**shows** *Base* $(a / b) = 0$
**by** (*simp add*: *assms*)

Division of self is 1 when invertible, 0 otherwise.

**lemma** *hyp-divide-self-if* [*simp*]:
**fixes** *a* :: ('a :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
**shows** $a / a =$ (*if Base* $a = 0$ *then* 0 *else* 1)
**by** (*metis hyp-divide-inverse zero-base-zero-inverse hyp-divide-self mult-zero-right*)

Repeated division is division by product of the denominators.

**lemma** *hyp-denominators-merge*:
**fixes** *a b c* :: ('a :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
**shows** $(a / b) / c = a / (c * b)$
**using** *hyp-inverse-mult-distrib*[*of c b*]
**by** (*simp add*: *hyp-divide-inverse mult.assoc*)

Finally, we derive general simplifications for division with addition and subtraction.

**lemma** *hyp-add-divide-eq-if-simps* [*divide-simps*]:
**fixes** *a b z* :: ('a :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
**shows** $a + b / z =$ (*if Base* $z = 0$ *then* $a$ *else* $(a * z + b) / z$)
**and** $a / z + b =$ (*if Base* $z = 0$ *then* $b$ *else* $(a + b * z) / z$)
**and** $-(a / z) + b =$ (*if Base* $z = 0$ *then* $b$ *else* $(-a + b * z) / z$)
**and** $a - b / z =$ (*if Base* $z = 0$ *then* $a$ *else* $(a * z - b) / z$)
**and** $a / z - b =$ (*if Base* $z = 0$ *then* $-b$ *else* $(a - b * z) / z$)
**and** $-(a / z) - b =$ (*if Base* $z = 0$ *then* $-b$ *else* $(-a - b * z) / z$)
**by** (*simp-all add*: *algebra-simps hyp-add-divide-eq-iff hyp-divide-inverse zero-base-zero-inverse*)

**lemma** *hyp-divide-eq-eq* [*divide-simps*]:
**fixes** *a b c* :: ('a :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
**shows** $b / c = a \longleftrightarrow$ (*if Base* $c \neq 0$ *then* $b = a * c$ *else* $a = 0$)

**by** (*metis hyp-divide-inverse hyp-denominator-eliminate mult-not-zero zero-base-zero-inverse*)

**lemma** *hyp-eq-divide-eq* [*divide-simps*]:
  **fixes** $a\ b\ c$ :: (*'a* :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
  **shows** $a = b\ /\ c \longleftrightarrow$ (*if Base* $c \neq 0$ *then* $a * c = b$ *else* $a = 0$)
  **by** (*metis hyp-divide-eq-eq*)

**lemma** *hyp-minus-divide-eq-eq* [*divide-simps*]:
  **fixes** $a\ b\ c$ :: (*'a* :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
  **shows** $-\ (b\ /\ c) = a \longleftrightarrow$ (*if Base* $c \neq 0$ *then* $-\ b = a * c$ *else* $a = 0$)
  **by** (*metis hyp-divide-minus-left hyp-eq-divide-eq*)

**lemma** *hyp-eq-minus-divide-eq* [*divide-simps*]:
  **fixes** $a\ b\ c$ :: (*'a* :: {*inverse*, *comm-ring-1*, *division-ring*}) *hyperdual*
  **shows** $a = -\ (b\ /\ c) \longleftrightarrow$ (*if Base* $c \neq 0$ *then* $a * c = -\ b$ *else* $a = 0$)
  **by** (*metis hyp-minus-divide-eq-eq*)

## 1.4 Real Scaling, Real Vector and Real Algebra

If the components can be scaled by real numbers then so can the hyperduals. We define the scaling pointwise.

**instantiation** *hyperdual* :: (*scaleR*) *scaleR*
**begin**

**primcorec** *scaleR-hyperdual*
  **where**
    *Base* $(f *_R x) = f *_R$ *Base* $x$
  | *Eps1* $(f *_R x) = f *_R$ *Eps1* $x$
  | *Eps2* $(f *_R x) = f *_R$ *Eps2* $x$
  | *Eps12* $(f *_R x) = f *_R$ *Eps12* $x$

**instance**
  **by** *standard*
**end**

If the components form a real vector space then so do the hyperduals.

**instance** *hyperdual* :: (*real-vector*) *real-vector*
  **by** *standard* (*simp-all add*: *algebra-simps*)

If the components form a real algebra then so do the hyperduals

**instance** *hyperdual* :: (*real-algebra-1*) *real-algebra-1*
  **by** *standard* (*simp-all add*: *algebra-simps*)

If the components are reals then *of-real* matches our embedding *of-comp*, and $(*_R)$ matches our scalar product $(*_H)$.

**lemma** *of-real* = *of-comp*
  **by** (*standard*, *simp add*: *of-real-def*)

**lemma** *scaleR-eq-scale*:
  $(*_R) = (*_H)$
  **by** (*standard*, *standard*, *simp*)

Hyperdual scalar product $(*_H)$ is compatible with $(*_R)$.

**lemma** *scaleH-scaleR*:
  **fixes** $a$ :: *'a* :: *real-algebra-1*
    **and** $b$ :: *'a hyperdual*
  **shows** $(f *_R a) *_H b = f *_R a *_H b$
    **and** $a *_H f *_R b = f *_R a *_H b$
  **by** *simp-all*

## 1.5 Real Inner Product and Real-Normed Vector Space

We now take a closer look at hyperduals as a real vector space.

If the components form a real inner product space then we can define one on the hyperduals as the sum of componentwise inner products. The norm is then defined as the square root of that inner product. We define signum, distance, uniformity and openness similarly as they are defined for complex numbers.

**instantiation** *hyperdual* :: (*real-inner*) *real-inner*
**begin**

**definition** *inner-hyperdual* :: *'a hyperdual* ⇒ *'a hyperdual* ⇒ *real*
  **where** $x \cdot y = Base\ x \cdot Base\ y + Eps1\ x \cdot Eps1\ y + Eps2\ x \cdot Eps2\ y + Eps12\ x \cdot Eps12\ y$

**definition** *norm-hyperdual* :: *'a hyperdual* ⇒ *real*
  **where** *norm-hyperdual* $x = sqrt\ (x \cdot x)$

**definition** *sgn-hyperdual* :: *'a hyperdual* ⇒ *'a hyperdual*
  **where** *sgn-hyperdual* $x = x\ /_R\ norm\ x$

**definition** *dist-hyperdual* :: *'a hyperdual* ⇒ *'a hyperdual* ⇒ *real*
  **where** *dist-hyperdual* $a\ b = norm(a - b)$

**definition** *uniformity-hyperdual* :: (*'a hyperdual* × *'a hyperdual*) *filter*
  **where** *uniformity-hyperdual* = (*INF* $e \in \{0 <..\}$. *principal* $\{(x, y).\ dist\ x\ y < e\}$)

**definition** *open-hyperdual* :: (*'a hyperdual*) *set* ⇒ *bool*
  **where** *open-hyperdual* $U \longleftrightarrow (\forall x \in U.\ eventually\ (\lambda(x', y).\ x' = x \longrightarrow y \in U)\ uniformity)$

**instance**
**proof**
  **fix** $x\ y\ z$ :: *'a hyperdual*
  **fix** $U$ :: (*'a hyperdual*) *set*
  **fix** $r$ :: *real*

**show** *dist x y = norm (x − y)*
**by** (*simp add*: *dist-hyperdual-def*)
**show** *sgn x = x /$_R$ norm x*
**by** (*simp add*: *sgn-hyperdual-def*)
**show** *uniformity = (INF e∈{0<..}. principal {(x :: 'a hyperdual, y). dist x y < e})*
**using** *uniformity-hyperdual-def* **by** *blast*
**show** *open U = (∀ x∈U. ∀$_F$ (x', y) in uniformity. x' = x ⟶ y ∈ U)*
**using** *open-hyperdual-def* **by** *blast*
**show** *x · y = y · x*
**by** (*simp add*: *inner-hyperdual-def inner-commute*)
**show** *(x + y) · z = x · z + y · z*
**and** *r *$_R$ x · y = r * (x · y)*
**and** *0 ≤ x · x*
**and** *norm x = sqrt (x · x)*
**by** (*simp-all add*: *inner-hyperdual-def norm-hyperdual-def algebra-simps*)
**show** *(x · x = 0) = (x = 0)*
**proof**
**assume** *x · x = 0*
**then have** *Base x · Base x + Eps1 x · Eps1 x + Eps2 x · Eps2 x + Eps12 x · Eps12 x = 0*
**by** (*simp add*: *inner-hyperdual-def*)
**then have** *Base x = 0 ∧ Eps1 x = 0 ∧ Eps2 x = 0 ∧ Eps12 x = 0*
**using** *inner-gt-zero-iff inner-ge-zero add-nonneg-nonneg*
**by** *smt*
**then show** *x = 0*
**by** *simp*
**next**
**assume** *x = 0*
**then show** *x · x = 0*
**by** (*simp add*: *inner-hyperdual-def*)
**qed**
**qed**
**end**

We then show that with this norm hyperduals with components that form a real normed algebra do not themselves form a normed algebra, by counter-example to the assumption that class adds.

**lemma** *not-normed-algebra*:
**shows** ¬(∀ *x y* :: (*'a* :: {*real-normed-algebra-1*, *real-inner*}) *hyperdual* . *norm (x * y) ≤ norm x * norm y*)
**proof** −
**have** *norm (Hyperdual (1 :: 'a) 1 1 1) = 2*
**by** (*simp add*: *norm-hyperdual-def inner-hyperdual-def dot-square-norm*)
**moreover have** *(Hyperdual (1 :: 'a) 1 1 1) * (Hyperdual 1 1 1 1) = Hyperdual 1 2 2 4*
**by** (*simp add*: *hyperdual.expand*)
**moreover have** *norm (Hyperdual (1 :: 'a) 2 2 4) > 4*

**by** (*simp add*: *norm-hyperdual-def inner-hyperdual-def dot-square-norm*)
**ultimately have** $\exists x\ y :: 'a\ hyperdual\ .\ norm\ (x * y) > norm\ x * norm\ y$
**by** (*smt power2-eq-square real-sqrt-four real-sqrt-pow2*)
**then show** *?thesis*
**by** (*simp add*: *not-le*)
**qed**

### 1.6 Euclidean Space

Next we define a basis for the space, consisting of four elements one for each component with *1* in the relevant component and *0* elsewhere.

**definition** *ba* :: (*'a* :: *zero-neq-one*) *hyperdual*
**where** *ba* = *Hyperdual 1 0 0 0*
**definition** *e1* :: (*'a* :: *zero-neq-one*) *hyperdual*
**where** *e1* = *Hyperdual 0 1 0 0*
**definition** *e2* :: (*'a* :: *zero-neq-one*) *hyperdual*
**where** *e2* = *Hyperdual 0 0 1 0*
**definition** *e12* :: (*'a* :: *zero-neq-one*) *hyperdual*
**where** *e12* = *Hyperdual 0 0 0 1*

**lemmas** *hyperdual-bases* = *ba-def e1-def e2-def e12-def*

Using the constructor *Hyperdual* is equivalent to using the linear combination with coefficients the relevant arguments.

**lemma** *Hyperdual-eq*:
**fixes** *a b c d* :: *'a* :: *ring-1*
**shows** *Hyperdual a b c d* = $a *_H ba + b *_H e1 + c *_H e2 + d *_H e12$
**by** (*simp add*: *hyperdual-bases*)

Projecting from the combination returns the relevant coefficient:

**lemma** *hyperdual-comb-sel* [*simp*]:
**fixes** *a b c d* :: *'a* :: *ring-1*
**shows** *Base* $(a *_H ba + b *_H e1 + c *_H e2 + d *_H e12) = a$
**and** *Eps1* $(a *_H ba + b *_H e1 + c *_H e2 + d *_H e12) = b$
**and** *Eps2* $(a *_H ba + b *_H e1 + c *_H e2 + d *_H e12) = c$
**and** *Eps12* $(a *_H ba + b *_H e1 + c *_H e2 + d *_H e12) = d$
**using** *Hyperdual-eq hyperdual.sel* **by** *metis+*

Any hyperdual number is a linear combination of these four basis elements.

**lemma** *hyperdual-linear-comb*:
**fixes** *x* :: (*'a* :: *ring-1*) *hyperdual*
**obtains** *a b c d* :: *'a* **where** $x = a *_H ba + b *_H e1 + c *_H e2 + d *_H e12$
**using** *hyperdual.exhaust Hyperdual-eq* **by** *metis*

The linear combination expressing any hyperdual number has as coefficients the projections of that number onto the relevant basis element.

**lemma** *hyperdual-eq*:

**fixes** $x$ :: ($'a$ :: *ring-1*) *hyperdual*
**shows** $x = Base\ x *_H\ ba + Eps1\ x *_H\ e1 + Eps2\ x *_H\ e2 + Eps12\ x *_H\ e12$
**using** *Hyperdual-eq hyperdual.collapse* **by** *smt*

Equality of hyperduals as linear combinations is equality of corresponding components.

**lemma** *hyperdual-eq-parts-cancel* [*simp*]:
**fixes** *a b c d* :: $'a$ :: *ring-1*
**shows** $(a *_H\ ba + b *_H\ e1 + c *_H\ e2 + d *_H\ e12 = a' *_H\ ba + b' *_H\ e1 + c' *_H\ e2 + d' *_H\ e12) \equiv$
$(a = a' \wedge b = b' \wedge c = c' \wedge d = d')$
**by** (*smt Hyperdual-eq hyperdual.inject*)

**lemma** *scaleH-cancel* [*simp*]:
**fixes** *a b* :: $'a$ :: *ring-1*
**shows** $(a *_H\ ba = b *_H\ ba) \equiv (a = b)$
**and** $(a *_H\ e1 = b *_H\ e1) \equiv (a = b)$
**and** $(a *_H\ e2 = b *_H\ e2) \equiv (a = b)$
**and** $(a *_H\ e12 = b *_H\ e12) \equiv (a = b)$
**by** (*auto simp add*: *hyperdual-bases*)+

We can now also show that the multiplication we use indeed has the hyperdual units nilpotent.

**lemma** *epsilon-squares* [*simp*]:
$(e1$ :: ($'a$ :: *ring-1*) *hyperdual*) $* e1 = 0$
$(e2$ :: ($'a$ :: *ring-1*) *hyperdual*) $* e2 = 0$
$(e12$ :: ($'a$ :: *ring-1*) *hyperdual*) $* e12 = 0$
**by** (*simp-all add*: *hyperdual-bases*)

However none of the hyperdual units is zero.

**lemma** *hyperdual-bases-nonzero* [*simp*]:
$ba \neq 0$
$e1 \neq 0$
$e2 \neq 0$
$e12 \neq 0$
**by** (*simp-all add*: *hyperdual-bases*)

Hyperdual units are orthogonal.

**lemma** *hyperdual-bases-ortho* [*simp*]:
($ba$ :: ($'a$ :: {*real-inner*,*zero-neq-one*}) *hyperdual*) $\cdot$ $e1 = 0$
($ba$ :: ($'a$ :: {*real-inner*,*zero-neq-one*}) *hyperdual*) $\cdot$ $e2 = 0$
($ba$ :: ($'a$ :: {*real-inner*,*zero-neq-one*}) *hyperdual*) $\cdot$ $e12 = 0$
($e1$ :: ($'a$ :: {*real-inner*,*zero-neq-one*}) *hyperdual*) $\cdot$ $e2 = 0$
($e1$ :: ($'a$ :: {*real-inner*,*zero-neq-one*}) *hyperdual*) $\cdot$ $e12 = 0$
($e2$ :: ($'a$ :: {*real-inner*,*zero-neq-one*}) *hyperdual*) $\cdot$ $e12 = 0$
**by** (*simp-all add*: *hyperdual-bases inner-hyperdual-def*)

Hyperdual units of norm equal to 1.

**lemma** *hyperdual-bases-norm* [*simp*]:
  (*ba* :: ('*a* :: {*real-inner*,*real-normed-algebra-1*}) *hyperdual*) · *ba* = *1*
  (*e1* :: ('*a* :: {*real-inner*,*real-normed-algebra-1*}) *hyperdual*) · *e1* = *1*
  (*e2* :: ('*a* :: {*real-inner*,*real-normed-algebra-1*}) *hyperdual*) · *e2* = *1*
  (*e12* :: ('*a* :: {*real-inner*,*real-normed-algebra-1*}) *hyperdual*) · *e12* = *1*
  **by** (*simp-all add*: *hyperdual-bases inner-hyperdual-def norm-eq-1*[*symmetric*])

We can also express earlier operations in terms of the linear combination.

**lemma** *add-hyperdual-parts*:
  **fixes** $a\ b\ c\ d$ :: '*a* :: *ring-1*
  **shows** $(a *_H ba + b *_H e1 + c *_H e2 + d *_H e12) + (a' *_H ba + b' *_H e1 + c' *_H e2 + d' *_H e12) =$
    $(a + a') *_H ba + (b + b') *_H e1 + (c + c') *_H e2 + (d + d') *_H e12$
  **by** (*simp add*: *scaleH-add*(*1*))

**lemma** *times-hyperdual-parts*:
  **fixes** $a\ b\ c\ d$ :: '*a* :: *ring-1*
  **shows** $(a *_H ba + b *_H e1 + c *_H e2 + d *_H e12) * (a' *_H ba + b' *_H e1 + c' *_H e2 + d' *_H e12) =$
    $(a * a') *_H ba + (a * b' + b * a') *_H e1 + (a * c' + c * a') *_H e2 + (a * d' + b * c' + c * b' + d * a') *_H e12$
  **by** (*simp add*: *hyperdual-bases*)

**lemma** *inverse-hyperdual-parts*:
  **fixes** $a\ b\ c\ d$ :: '*a* :: {*inverse*,*ring-1*}
  **shows** *inverse* $(a *_H ba + b *_H e1 + c *_H e2 + d *_H e12) =$
    $(1 / a) *_H ba + (- b / a \hat{}\ 2) *_H e1 + (- c / a \hat{}\ 2) *_H e2 + (2 * (b * c / a \hat{}\ 3) - d / a \hat{}\ 2) *_H e12$
  **by** (*simp add*: *hyperdual-bases*)

Next we show that hyperduals form a euclidean space with the help of the basis we defined earlier and the above inner product if the component is an instance of *euclidean-space* and *real-algebra-1*. The basis of this space is each of the basis elements we defined scaled by each of the basis elements of the component type, representing the expansion of the space for each component of the hyperdual numbers.

**instantiation** *hyperdual* :: ({*euclidean-space*, *real-algebra-1*}) *euclidean-space*
**begin**

**definition** *Basis-hyperdual* :: ('*a hyperdual*) *set*
  **where** *Basis* = ($\bigcup$ *i*∈{*ba*,*e1*,*e2*,*e12*}. ($\lambda u.\ u *_H i$) ‘ *Basis*)

**instance**
**proof**
  **fix** *x y z* :: '*a hyperdual*
  **show** (*Basis* :: ('*a hyperdual*) *set*) ≠ {}
    **and** *finite* (*Basis* :: ('*a hyperdual*) *set*)
    **by** (*simp-all add*: *Basis-hyperdual-def*)

**show** $x \in Basis \Longrightarrow y \in Basis \Longrightarrow x \cdot y = (if\ x = y\ then\ 1\ else\ 0)$
**unfolding** *Basis-hyperdual-def inner-hyperdual-def hyperdual-bases*
**by** (*auto dest*: *inner-not-same-Basis*)
**show** $(\forall u \in Basis.\ x \cdot u = 0) = (x = 0)$
**by** (*auto simp add*: *Basis-hyperdual-def ball-Un inner-hyperdual-def hyperdual-bases euclidean-all-zero-iff*)
**qed**
**end**

## 1.7 Bounded Linear Projections

Now we can show that each projection to a basis element is a bounded linear map.

**lemma** *bounded-linear-Base*: *bounded-linear Base*
**proof**
**show** $\bigwedge b1\ b2.\ Base\ (b1 + b2) = Base\ b1 + Base\ b2$
**by** *simp*
**show** $\bigwedge r\ b.\ Base\ (r *_R b) = r *_R Base\ b$
**by** *simp*
**have** $\forall x.\ norm\ (Base\ x) \leq norm\ x$
**by** (*simp add*: *inner-hyperdual-def norm-eq-sqrt-inner*)
**then show** $\exists K.\ \forall x.\ norm\ (Base\ x) \leq norm\ x * K$
**by** (*metis mult.commute mult.left-neutral*)
**qed**
**lemma** *bounded-linear-Eps1*: *bounded-linear Eps1*
**proof**
**show** $\bigwedge b1\ b2.\ Eps1\ (b1 + b2) = Eps1\ b1 + Eps1\ b2$
**by** *simp*
**show** $\bigwedge r\ b.\ Eps1\ (r *_R b) = r *_R Eps1\ b$
**by** *simp*
**have** $\forall x.\ norm\ (Eps1\ x) \leq norm\ x$
**by** (*simp add*: *inner-hyperdual-def norm-eq-sqrt-inner*)
**then show** $\exists K.\ \forall x.\ norm\ (Eps1\ x) \leq norm\ x * K$
**by** (*metis mult.commute mult.left-neutral*)
**qed**
**lemma** *bounded-linear-Eps2*: *bounded-linear Eps2*
**proof**
**show** $\bigwedge b1\ b2.\ Eps2\ (b1 + b2) = Eps2\ b1 + Eps2\ b2$
**by** *simp*
**show** $\bigwedge r\ b.\ Eps2\ (r *_R b) = r *_R Eps2\ b$
**by** *simp*
**have** $\forall x.\ norm\ (Eps2\ x) \leq norm\ x$
**by** (*simp add*: *inner-hyperdual-def norm-eq-sqrt-inner*)
**then show** $\exists K.\ \forall x.\ norm\ (Eps2\ x) \leq norm\ x * K$
**by** (*metis mult.commute mult.left-neutral*)
**qed**
**lemma** *bounded-linear-Eps12*: *bounded-linear Eps12*
**proof**
**show** $\bigwedge b1\ b2.\ Eps12\ (b1 + b2) = Eps12\ b1 + Eps12\ b2$

**by** *simp*
**show** $\bigwedge r\ b.\ Eps12\ (r *_R b) = r *_R Eps12\ b$
**by** *simp*
**have** $\forall x.\ norm\ (Eps12\ x) \leq norm\ x$
**by** (*simp add*: *inner-hyperdual-def norm-eq-sqrt-inner*)
**then show** $\exists K.\ \forall x.\ norm\ (Eps12\ x) \leq norm\ x * K$
**by** (*metis mult.commute mult.left-neutral*)
**qed**

This bounded linearity gives us a range of useful theorems about limits, convergence and derivatives of these projections.

**lemmas** *tendsto-Base* = *bounded-linear.tendsto*[*OF bounded-linear-Base*]
**lemmas** *tendsto-Eps1* = *bounded-linear.tendsto*[*OF bounded-linear-Eps1*]
**lemmas** *tendsto-Eps2* = *bounded-linear.tendsto*[*OF bounded-linear-Eps2*]
**lemmas** *tendsto-Eps12* = *bounded-linear.tendsto*[*OF bounded-linear-Eps12*]

**lemmas** *has-derivative-Base* = *bounded-linear.has-derivative*[*OF bounded-linear-Base*]
**lemmas** *has-derivative-Eps1* = *bounded-linear.has-derivative*[*OF bounded-linear-Eps1*]
**lemmas** *has-derivative-Eps2* = *bounded-linear.has-derivative*[*OF bounded-linear-Eps2*]
**lemmas** *has-derivative-Eps12* = *bounded-linear.has-derivative*[*OF bounded-linear-Eps12*]

## 1.8 Convergence

**lemma** *inner-mult-le-mult-inner*:
**fixes** $a\ b$ :: $'a$ :: {*real-inner*,*real-normed-algebra*}
**shows** $((a * b) \cdot (a * b)) \leq (a \cdot a) * (b \cdot b)$
**by** (*metis real-sqrt-le-iff norm-eq-sqrt-inner real-sqrt-mult norm-mult-ineq*)

**lemma** *bounded-bilinear-scaleH*:
*bounded-bilinear* $((*_H)$ :: $('a$ :: {*real-normed-algebra-1*, *real-inner*}) $\Rightarrow$ $'a$ *hyperdual* $\Rightarrow$ $'a$ *hyperdual*)
**proof** (*auto simp add*: *bounded-bilinear-def scaleH-add scaleH-scaleR del*:*exI intro*!:*exI*)
**fix** $a$ :: $'a$
**and** $b$ :: $'a$ *hyperdual*
**have** $norm\ (a *_H b) = sqrt\ ((a * Base\ b) \cdot (a * Base\ b) + (a * Eps1\ b) \cdot (a * Eps1\ b) + (a * Eps2\ b) \cdot (a * Eps2\ b) + (a * Eps12\ b) \cdot (a * Eps12\ b))$
**by** (*simp add*: *norm-eq-sqrt-inner inner-hyperdual-def*)
**moreover have** $norm\ a * norm\ b = sqrt\ (a \cdot a * (Base\ b \cdot Base\ b + Eps1\ b \cdot Eps1\ b + Eps2\ b \cdot Eps2\ b + Eps12\ b \cdot Eps12\ b))$
**by** (*simp add*: *norm-eq-sqrt-inner inner-hyperdual-def real-sqrt-mult*)
**moreover have** $sqrt\ ((a * Base\ b) \cdot (a * Base\ b) + (a * Eps1\ b) \cdot (a * Eps1\ b) + (a * Eps2\ b) \cdot (a * Eps2\ b) + (a * Eps12\ b) \cdot (a * Eps12\ b)) \leq$
$sqrt\ (a \cdot a * (Base\ b \cdot Base\ b + Eps1\ b \cdot Eps1\ b + Eps2\ b \cdot Eps2\ b + Eps12\ b \cdot Eps12\ b))$
**by** (*simp add*: *distrib-left add-mono inner-mult-le-mult-inner*)
**ultimately show** $norm\ (a *_H b) \leq norm\ a * norm\ b * 1$
**by** *simp*
**qed**

**lemmas** *tendsto-scaleH* = *bounded-bilinear.tendsto*[*OF bounded-bilinear-scaleH*]

We describe how limits behave for general hyperdual-valued functions.

First we prove that we can go from convergence of the four component functions to the convergence of the hyperdual-valued function whose components they define.

**lemma** *tendsto-Hyperdual*:
  **fixes** *f* :: *'a* ⇒ (*'b* :: {*real-normed-algebra-1*, *real-inner*})
  **assumes** (*f* ⟶ *a*) *F*
    **and** (*g* ⟶ *b*) *F*
    **and** (*h* ⟶ *c*) *F*
    **and** (*i* ⟶ *d*) *F*
   **shows** ((λ*x*. *Hyperdual* (*f x*) (*g x*) (*h x*) (*i x*)) ⟶ *Hyperdual a b c d*) *F*
**proof** –
  **have** ((λ*x*. (*f x*) $*_H$ *ba*) ⟶ *a* $*_H$ *ba*) *F*
    ((λ*x*. (*g x*) $*_H$ *e1*) ⟶ *b* $*_H$ *e1*) *F*
    ((λ*x*. (*h x*) $*_H$ *e2*) ⟶ *c* $*_H$ *e2*) *F*
    ((λ*x*. (*i x*) $*_H$ *e12*) ⟶ *d* $*_H$ *e12*) *F*
   **by** (*rule tendsto-scaleH*[*OF - tendsto-const*], *rule assms*)+
  **then have** ((λ*x*. (*f x*) $*_H$ *ba* + (*g x*) $*_H$ *e1* + (*h x*) $*_H$ *e2* + (*i x*) $*_H$ *e12*) ⟶
      *a* $*_H$ *ba* + *b* $*_H$ *e1* + *c* $*_H$ *e2* + *d* $*_H$ *e12*) *F*
   **by** (*rule tendsto-add*[*OF tendsto-add*[*OF tendsto-add*]] ; *assumption*)
  **then show** *?thesis*
   **by** (*simp add*: *Hyperdual-eq*)
**qed**

Next we complete the equivalence by proving the other direction, from convergence of a hyperdual-valued function to the convergence of the projected component functions.

**lemma** *tendsto-hyperdual-iff*:
  **fixes** *f* :: *'a* ⇒ (*'b* :: {*real-normed-algebra-1*, *real-inner*}) *hyperdual*
  **shows** (*f* ⟶ *x*) *F* ⟷
   ((λ*x*. *Base* (*f x*)) ⟶ *Base x*) *F* ∧
   ((λ*x*. *Eps1* (*f x*)) ⟶ *Eps1 x*) *F* ∧
   ((λ*x*. *Eps2* (*f x*)) ⟶ *Eps2 x*) *F* ∧
   ((λ*x*. *Eps12* (*f x*)) ⟶ *Eps12 x*) *F*
**proof** *safe*
  **assume** (*f* ⟶ *x*) *F*
  **then show** ((λ*x*. *Base* (*f x*)) ⟶ *Base x*) *F*
    **and** ((λ*x*. *Eps1* (*f x*)) ⟶ *Eps1 x*) *F*
    **and** ((λ*x*. *Eps2* (*f x*)) ⟶ *Eps2 x*) *F*
    **and** ((λ*x*. *Eps12* (*f x*)) ⟶ *Eps12 x*) *F*
   **by** (*simp-all add*: *tendsto-Base tendsto-Eps1 tendsto-Eps2 tendsto-Eps12*)
**next**
  **assume** ((λ*x*. *Base* (*f x*)) ⟶ *Base x*) *F*
   **and** ((λ*x*. *Eps1* (*f x*)) ⟶ *Eps1 x*) *F*
   **and** ((λ*x*. *Eps2* (*f x*)) ⟶ *Eps2 x*) *F*

**and** (($\lambda x$. *Eps12* ($f\ x$)) $\longrightarrow$ *Eps12* $x$) $F$
**then show** ($f \longrightarrow x$) $F$
**using** *tendsto-Hyperdual*[*of* $\lambda x$. *Base* ($f\ x$) *Base* $x$ $F$ $\lambda x$. *Eps1* ($f\ x$) *Eps1* $x$ $\lambda x$. *Eps2* ($f\ x$) *Eps2* $x$ $\lambda x$. *Eps12* ($f\ x$) *Eps12* $x$]
**by** *simp*
**qed**

## 1.9 Derivatives

We describe how derivatives of hyperdual-valued functions behave. Due to hyperdual numbers not forming a normed field, the derivative relation we must use is the general Fréchet derivative (*has-derivative*).

The left to right implication of the following equivalence is easily proved by the known derivative behaviour of the projections. The other direction is more difficult, because we have to construct the two requirements of the (*has-derivative*) relation, the limit and the bounded linearity of the derivative. While the limit is simple to construct from the component functions by previous lemma, the bounded linearity is more involved.

**lemma** *has-derivative-hyperdual-iff*:
**fixes** $f$ :: ($'a$ :: *real-normed-vector*) $\Rightarrow$ ($'b$ :: {*real-normed-algebra-1*, *real-inner*}) *hyperdual*
**shows** ($f$ *has-derivative* $Df$) $F \longleftrightarrow$
(($\lambda x$. *Base* ($f\ x$)) *has-derivative* ($\lambda x$. *Base* ($Df\ x$))) $F \wedge$
(($\lambda x$. *Eps1* ($f\ x$)) *has-derivative* ($\lambda x$. *Eps1* ($Df\ x$))) $F \wedge$
(($\lambda x$. *Eps2* ($f\ x$)) *has-derivative* ($\lambda x$. *Eps2* ($Df\ x$))) $F \wedge$
(($\lambda x$. *Eps12* ($f\ x$)) *has-derivative* ($\lambda x$. *Eps12* ($Df\ x$))) $F$
**proof** *safe*
— Left to Right
**assume** *assm*: ($f$ *has-derivative* $Df$) $F$
**show** (($\lambda x$. *Base* ($f\ x$)) *has-derivative* ($\lambda x$. *Base* ($Df\ x$))) $F$
**using** *assm has-derivative-Base* **by** *blast*
**show** (($\lambda x$. *Eps1* ($f\ x$)) *has-derivative* ($\lambda x$. *Eps1* ($Df\ x$))) $F$
**using** *assm has-derivative-Eps1* **by** *blast*
**show** (($\lambda x$. *Eps2* ($f\ x$)) *has-derivative* ($\lambda x$. *Eps2* ($Df\ x$))) $F$
**using** *assm has-derivative-Eps2* **by** *blast*
**show** (($\lambda x$. *Eps12* ($f\ x$)) *has-derivative* ($\lambda x$. *Eps12* ($Df\ x$))) $F$
**using** *assm has-derivative-Eps12* **by** *blast*
**next**
— Right to Left
**assume** *assm*:
(($\lambda x$. *Base* ($f\ x$)) *has-derivative* ($\lambda x$. *Base* ($Df\ x$))) $F$
(($\lambda x$. *Eps1* ($f\ x$)) *has-derivative* ($\lambda x$. *Eps1* ($Df\ x$))) $F$
(($\lambda x$. *Eps2* ($f\ x$)) *has-derivative* ($\lambda x$. *Eps2* ($Df\ x$))) $F$
(($\lambda x$. *Eps12* ($f\ x$)) *has-derivative* ($\lambda x$. *Eps12* ($Df\ x$))) $F$

— First prove the limit from component function limits
**have** (($\lambda y$. *Base* ((($f\ y - f$ (*Lim* $F$ ($\lambda x$. $x$))) $-$ $Df$ ($y$ $-$ *Lim* $F$ ($\lambda x$. $x$))) $/_R$ *norm* ($y$ $-$ *Lim* $F$ ($\lambda x$. $x$)))) $\longrightarrow$ *Base* $0$) $F$

**using** *assm has-derivative-def*[*of* (λ*x*. *Base* (*f x*)) (λ*x*. *Base* (*Df x*)) *F*] **by** *simp*
**moreover have** ((λ*y*. *Eps1* (((*f y* − *f* (*Lim F* (λ*x*. *x*))) − *Df* (*y* − *Lim F* (λ*x*. *x*))) $/_R$ *norm* (*y* − *Lim F* (λ*x*. *x*)))) ⟶ *Base 0*) *F*
**using** *assm has-derivative-def*[*of* (λ*x*. *Eps1* (*f x*)) (λ*x*. *Eps1* (*Df x*)) *F*] **by** *simp*
**moreover have** ((λ*y*. *Eps2* (((*f y* − *f* (*Lim F* (λ*x*. *x*))) − *Df* (*y* − *Lim F* (λ*x*. *x*))) $/_R$ *norm* (*y* − *Lim F* (λ*x*. *x*)))) ⟶ *Base 0*) *F*
**using** *assm has-derivative-def*[*of* (λ*x*. *Eps2* (*f x*)) (λ*x*. *Eps2* (*Df x*)) *F*] **by** *simp*
**moreover have** ((λ*y*. *Eps12* (((*f y* − *f* (*Lim F* (λ*x*. *x*))) − *Df* (*y* − *Lim F* (λ*x*. *x*))) $/_R$ *norm* (*y* − *Lim F* (λ*x*. *x*)))) ⟶ *Base 0*) *F*
**using** *assm has-derivative-def*[*of* (λ*x*. *Eps12* (*f x*)) (λ*x*. *Eps12* (*Df x*)) *F*] **by** *simp*
**ultimately have** ((λ*y*. ((*f y* − *f* (*Lim F* (λ*x*. *x*))) − *Df* (*y* − *Lim F* (λ*x*. *x*))) $/_R$ *norm* (*y* − *Lim F* (λ*x*. *x*))) ⟶ *0*) *F*
**by** (*simp add*: *tendsto-hyperdual-iff*)

— Next prove bounded linearity of the composed derivative by proving each of that class' assumptions from bounded linearity of the component derivatives
**moreover have** *bounded-linear Df*
**proof**
**have** *bl*:
*bounded-linear* (λ*x*. *Base* (*Df x*))
*bounded-linear* (λ*x*. *Eps1* (*Df x*))
*bounded-linear* (λ*x*. *Eps2* (*Df x*))
*bounded-linear* (λ*x*. *Eps12* (*Df x*))
**using** *assm has-derivative-def* **by** *blast*+
**then have** *linear* (λ*x*. *Base* (*Df x*))
**and** *linear* (λ*x*. *Eps1* (*Df x*))
**and** *linear* (λ*x*. *Eps2* (*Df x*))
**and** *linear* (λ*x*. *Eps12* (*Df x*))
**using** *bounded-linear.linear* **by** *blast*+
**then show** ⋀*x y*. *Df* (*x* + *y*) = *Df x* + *Df y*
**and** ⋀*x y*. *Df* (*x* $*_R$ *y*) = *x* $*_R$ *Df y*
**using** *plus-hyperdual.code scaleR-hyperdual.code* **by** (*simp-all add*: *linear-iff*)

**show** ∃ *K*. ∀ *x*. *norm* (*Df x*) ≤ *norm x* ∗ *K*
**proof** −
**obtain** *k-re k-eps1 k-eps2 k-eps12*
**where** ∀ *x*. (*norm* (*Base* (*Df x*)))ˆ*2* ≤ (*norm x* ∗ *k-re*)ˆ*2*
**and** ∀ *x*. (*norm* (*Eps1* (*Df x*)))ˆ*2* ≤ (*norm x* ∗ *k-eps1*)ˆ*2*
**and** ∀ *x*. (*norm* (*Eps2* (*Df x*)))ˆ*2* ≤ (*norm x* ∗ *k-eps2*)ˆ*2*
**and** ∀ *x*. (*norm* (*Eps12* (*Df x*)))ˆ*2* ≤ (*norm x* ∗ *k-eps12*)ˆ*2*
**using** *bl bounded-linear.bounded norm-ge-zero power-mono* **by** *metis*
**moreover have** ∀ *x*. (*norm* (*Df x*))ˆ*2* = (*norm* (*Base* (*Df x*)))ˆ*2* + (*norm* (*Eps1* (*Df x*)))ˆ*2* + (*norm* (*Eps2* (*Df x*)))ˆ*2* + (*norm* (*Eps12* (*Df x*)))ˆ*2*
**using** *inner-hyperdual-def power2-norm-eq-inner* **by** *metis*
**ultimately have** ∀ *x*. (*norm* (*Df x*))ˆ*2* ≤ (*norm x* ∗ *k-re*)ˆ*2* + (*norm x* ∗ *k-eps1*)ˆ*2* + (*norm x* ∗ *k-eps2*)ˆ*2* + (*norm x* ∗ *k-eps12*)ˆ*2*
**by** *smt*
**then have** ∀ *x*. (*norm* (*Df x*))ˆ*2* ≤ (*norm x*)ˆ*2* ∗ (*k-re*ˆ*2* + *k-eps1*ˆ*2* +

*k-eps2^2 + k-eps12^2*)
**by** (*simp add*: *distrib-left power-mult-distrib*)
**then have** *final*: ∀ *x*. *norm* (*Df x*) ≤ *norm x* ∗ *sqrt*(*k-re^2 + k-eps1^2 + k-eps2^2 + k-eps12^2*)
**using** *real-le-rsqrt real-sqrt-mult real-sqrt-pow2* **by** *fastforce*
**then show** ∃ *K*. ∀ *x*. *norm* (*Df x*) ≤ *norm x* ∗ *K*
**by** *blast*
**qed**
**qed**
— Finally put the two together to finish the proof
**ultimately show** (*f has-derivative Df*) *F*
**by** (*simp add*: *has-derivative-def*)
**qed**

Stop automatically unfolding hyperduals into components outside this theory:

**lemmas** [*iff del*] = *hyperdual-eq-iff*

**end**

# 2 Twice Field Differentiable

**theory** *TwiceFieldDifferentiable*
**imports** *HOL−Analysis.Analysis*
**begin**

## 2.1 Differentiability on a Set

A function is differentiable on a set iff it is differentiable at any point within that set.

**definition** *field-differentiable-on* :: (*'a* ⇒ *'a*::*real-normed-field*) ⇒ *'a set* ⇒ *bool*
(**infix** *field'-differentiable'-on 50*)
**where** *f field-differentiable-on s* ≡ ∀ *x*∈*s*. *f field-differentiable* (*at x within s*)

This is preserved for subsets.

**lemma** *field-differentiable-on-subset*:
**assumes** *f field-differentiable-on S*
**and** *T* ⊆ *S*
**shows** *f field-differentiable-on T*
**by** (*meson assms field-differentiable-on-def field-differentiable-within-subset in-mono*)

## 2.2 Twice Differentiability

Informally, a function is twice differentiable at x iff it is differentiable on some neighbourhood of x and its derivative is differentiable at x.

**definition** *twice-field-differentiable-at* :: [*'a* ⇒ *'a*::*real-normed-field*, *'a* ] ⇒ *bool*
(**infixr** (*twice'-field'-differentiable'-at*) *50*)

**where** *f twice-field-differentiable-at x* ≡
∃ *S*. *f field-differentiable-on S* ∧ *x* ∈ *interior S* ∧ (*deriv f*) *field-differentiable* (*at x*)

**lemma** *once-field-differentiable-at*:
*f twice-field-differentiable-at x* ⟹ *f field-differentiable* (*at x*)
**by** (*metis at-within-interior field-differentiable-on-def interior-subset subsetD twice-field-differentiable-at-def*)

**lemma** *deriv-field-differentiable-at*:
*f twice-field-differentiable-at x* ⟹ *deriv f field-differentiable* (*at x*)
**using** *twice-field-differentiable-at-def* **by** *blast*

For a composition of two functions twice differentiable at x, the chain rule eventually holds on some neighbourhood of x.

**lemma** *eventually-deriv-compose*:
**assumes** ∃ *S*. *f field-differentiable-on S* ∧ *x* ∈ *interior S*
**and** *g twice-field-differentiable-at* (*f x*)
**shows** $\forall_F$ *x in nhds x*. *deriv* (λ*x*. *g* (*f x*)) *x* = *deriv g* (*f x*) ∗ *deriv f x*
**proof** −
**obtain** *S S′*
**where** *Df-on-S*: *f field-differentiable-on S* **and** *x-int-S*: *x* ∈ *interior S*
**and** *Dg-on-S′*: *g field-differentiable-on S′* **and** *fx-int-S′*: *f x* ∈ *interior S′*
**using** *assms twice-field-differentiable-at-def* **by** *blast*

**let** *?T* = {*x* ∈ *interior S*. *f x* ∈ *interior S′*}

**have** *continuous-on* (*interior S*) *f*
**by** (*meson Df-on-S continuous-on-eq-continuous-within continuous-on-subset field-differentiable-imp-continuous-at*
*field-differentiable-on-def interior-subset*)
**then have** *open* (*interior S* ∩ {*x*. *f x* ∈ *interior S′*})
**by** (*metis continuous-open-preimage open-interior vimage-def*)
**then have** *x-int-T*: *x* ∈ *interior ?T*
**by** (*metis* (*no-types*) *Collect-conj-eq Collect-mem-eq Int-Collect fx-int-S′ interior-eq x-int-S*)
**moreover have** *Dg-on-fT*: *g field-differentiable-on f‘?T*
**by** (*metis* (*no-types*, *lifting*) *Dg-on-S′ field-differentiable-on-subset image-Collect-subsetI interior-subset*)
**moreover have** *Df-on-T*: *f field-differentiable-on ?T*
**using** *field-differentiable-on-subset Df-on-S*
**by** (*metis Collect-subset interior-subset*)
**moreover have** ∀ *x* ∈ *interior ?T*. *deriv* (λ*x*. *g* (*f x*)) *x* = *deriv g* (*f x*) ∗ *deriv f x*
**proof**
**fix** *x*
**assume** *x-int-T*: *x* ∈ *interior ?T*
**have** *f field-differentiable at x*
**by** (*metis* (*no-types*, *lifting*) *Df-on-T at-within-interior field-differentiable-on-def*

*interior-subset subsetD x-int-T*)
**moreover have** *g field-differentiable at* (*f x*)
**by** (*metis* (*no-types*, *lifting*) *Dg-on-S′ at-within-interior field-differentiable-on-def interior-subset mem-Collect-eq subsetD x-int-T*)
**ultimately have** *deriv* (*g* ∘ *f*) *x* = *deriv g* (*f x*) ∗ *deriv f x*
**using** *deriv-chain*[*of f x g*] **by** *simp*
**then show** *deriv* (*λx. g* (*f x*)) *x* = *deriv g* (*f x*) ∗ *deriv f x*
**by** (*simp add*: *comp-def*)
**qed**
**ultimately show** *?thesis*
**using** *eventually-nhds* **by** *blast*
**qed**

**lemma** *eventually-deriv-compose′*:
**assumes** *f twice-field-differentiable-at x*
**and** *g twice-field-differentiable-at* (*f x*)
**shows** $\forall_F$ *x in nhds x. deriv* (*λx. g* (*f x*)) *x* = *deriv g* (*f x*) ∗ *deriv f x*
**using** *assms eventually-deriv-compose twice-field-differentiable-at-def* **by** *blast*

Composition of twice differentiable functions is twice differentiable.

**lemma** *twice-field-differentiable-at-compose*:
**assumes** *f twice-field-differentiable-at x*
**and** *g twice-field-differentiable-at* (*f x*)
**shows** (*λx. g* (*f x*)) *twice-field-differentiable-at x*
**proof** −
**obtain** *S S′*
**where** *Df-on-S*: *f field-differentiable-on S* **and** *x-int-S*: *x* ∈ *interior S*
**and** *Dg-on-S′*: *g field-differentiable-on S′* **and** *fx-int-S′*: *f x* ∈ *interior S′*
**using** *assms twice-field-differentiable-at-def* **by** *blast*

**let** *?T* = {*x* ∈ *interior S. f x* ∈ *interior S′*}

**have** *continuous-on* (*interior S*) *f*
**by** (*meson Df-on-S continuous-on-eq-continuous-within continuous-on-subset field-differentiable-imp-continuous-at field-differentiable-on-def interior-subset*)
**then have** *open* (*interior S* ∩ {*x. f x* ∈ *interior S′*})
**by** (*metis continuous-open-preimage open-interior vimage-def*)
**then have** *x-int-T*: *x* ∈ *interior ?T*
**by** (*metis* (*no-types*) *Collect-conj-eq Collect-mem-eq Int-Collect fx-int-S′ interior-eq x-int-S*)

**have** *Dg-on-fT*: *g field-differentiable-on f‘?T*
**by** (*metis* (*no-types*, *lifting*) *Dg-on-S′ field-differentiable-on-subset image-Collect-subsetI interior-subset*)

**have** *Df-on-T*: *f field-differentiable-on ?T*
**using** *field-differentiable-on-subset Df-on-S*
**by** (*metis Collect-subset interior-subset*)

**have** ($\lambda x.\ g\ (f\ x)$) *field-differentiable-on ?T*
**unfolding** *field-differentiable-on-def*
**proof**
**fix** $x$ **assume** *x-int*: $x \in \{x \in \mathit{interior}\ S.\ f\ x \in \mathit{interior}\ S'\}$
**have** *f field-differentiable at x*
**by** (*metis Df-on-S at-within-interior field-differentiable-on-def interior-subset mem-Collect-eq subsetD x-int*)
**moreover have** *g field-differentiable at* (*f x*)
**by** (*metis Dg-on-S′ at-within-interior field-differentiable-on-def interior-subset mem-Collect-eq subsetD x-int*)
**ultimately have** ($g \circ f$) *field-differentiable at x*
**by** (*simp add*: *field-differentiable-compose*)
**then have** ($\lambda x.\ g\ (f\ x)$) *field-differentiable at x*
**by** (*simp add*: *comp-def*)
**then show** ($\lambda x.\ g\ (f\ x)$) *field-differentiable at x within* $\{x \in \mathit{interior}\ S.\ f\ x \in \mathit{interior}\ S'\}$
**using** *field-differentiable-at-within* **by** *blast*
**qed**
**moreover have** *deriv* ($\lambda x.\ g\ (f\ x)$) *field-differentiable at x*
**proof** −
**have** ($\lambda x.$ *deriv g* ($f\ x$)) *field-differentiable at x*
**by** (*metis DERIV-chain2 assms deriv-field-differentiable-at field-differentiable-def once-field-differentiable-at*)
**then have** ($\lambda x.$ *deriv g* ($f\ x$) $*$ *deriv f x*) *field-differentiable at x*
**using** *assms field-differentiable-mult*[*of* $\lambda x.$ *deriv g* ($f\ x$)]
**by** (*simp add*: *deriv-field-differentiable-at*)
**moreover have** *deriv* (*deriv* ($\lambda x.\ g\ (f\ x)$)) $x$ = *deriv* ($\lambda x.$ *deriv g* ($f\ x$) $*$ *deriv f x*) $x$
**using** *assms Df-on-S x-int-S deriv-cong-ev eventually-deriv-compose* **by** *fastforce*
**ultimately show** *?thesis*
**using** *assms eventually-deriv-compose DERIV-deriv-iff-field-differentiable Df-on-S x-int-S*
*DERIV-cong-ev*[*of x x deriv* ($\lambda x.\ g\ (f\ x)$) $\lambda x.$ *deriv g* ($f\ x$) $*$ *deriv f x*]
**by** *blast*
**qed**
**ultimately show** *?thesis*
**using** *twice-field-differentiable-at-def x-int-T* **by** *blast*
**qed**

### 2.2.1 Constant

**lemma** *twice-field-differentiable-at-const* [*simp*, *intro*]:
($\lambda x.\ a$) *twice-field-differentiable-at x*
**by** (*auto intro*: *exI* [*of - UNIV*] *simp add*: *twice-field-differentiable-at-def field-differentiable-on-def*)

### 2.2.2 Identity

**lemma** *twice-field-differentiable-at-ident* [*simp*, *intro*]:

($\lambda x.\ x$) *twice-field-differentiable-at x*
**proof** –
**have** $\forall x \in UNIV$. ($\lambda x.\ x$) *field-differentiable at x*
**and** *deriv* (($\lambda x.\ x$)) *field-differentiable at x*
**by** *simp-all*
**then show** *?thesis*
**unfolding** *twice-field-differentiable-at-def field-differentiable-on-def*
**by** *fastforce*
**qed**

### 2.2.3 Constant Multiplication

**lemma** *twice-field-differentiable-at-cmult* [*simp*, *intro*]:
(∗) *k twice-field-differentiable-at x*
**proof** –
**have** $\forall x \in UNIV$. (∗) *k field-differentiable at x*
**by** *simp*
**moreover have** *deriv* ((∗) *k*) *field-differentiable at x*
**by** *simp*
**ultimately show** *?thesis*
**unfolding** *twice-field-differentiable-at-def field-differentiable-on-def*
**by** *fastforce*
**qed**

**lemma** *twice-field-differentiable-at-uminus* [*simp*, *intro*]:
*uminus twice-field-differentiable-at x*
**proof** –
**have** $\forall x \in UNIV$. *uminus field-differentiable at x*
**by** (*simp add*: *field-differentiable-minus*)
**moreover have** *deriv uminus field-differentiable at x*
**by** *simp*
**ultimately show** *?thesis*
**unfolding** *twice-field-differentiable-at-def field-differentiable-on-def*
**by** *auto*
**qed**

**lemma** *twice-field-differentiable-at-uminus-fun* [*intro*]:
**assumes** *f twice-field-differentiable-at x*
**shows** ($\lambda x.\ -\ f\ x$) *twice-field-differentiable-at x*
**by** (*simp add*: *assms twice-field-differentiable-at-compose*)

### 2.2.4 Real Scaling

**lemma** *deriv-scaleR-right-id* [*simp*]:
(*deriv* (($*_R$) *k*)) = ($\lambda z.\ k *_R 1$)
**using** *DERIV-imp-deriv has-field-derivative-scaleR-right DERIV-ident* **by** *blast*

**lemma** *deriv-deriv-scaleR-right-id* [*simp*]:
*deriv* (*deriv* (($*_R$) *k*)) = ($\lambda z.\ 0$)
**by** *simp*

**lemma** *deriv-scaleR-right*:
*f field-differentiable* (*at z*) $\Longrightarrow$ *deriv* ($\lambda x.$ *k* $*_R$ *f x*) *z* = *k* $*_R$ *deriv f z*
**by** (*simp add*: *DERIV-imp-deriv field-differentiable-derivI has-field-derivative-scaleR-right*)

**lemma** *field-differentiable-scaleR-right* [*intro*]:
*f field-differentiable F* $\Longrightarrow$ ($\lambda x.$ *c* $*_R$ *f x*) *field-differentiable F*
**using** *field-differentiable-def has-field-derivative-scaleR-right* **by** *blast*

**lemma** *has-field-derivative-scaleR-deriv-right*:
**assumes** *f twice-field-differentiable-at z*
**shows** (($\lambda x.$ *k* $*_R$ *deriv f x*) *has-field-derivative* *k* $*_R$ *deriv* (*deriv f*) *z*) (*at z*)
**by** (*simp add*: *DERIV-deriv-iff-field-differentiable assms deriv-field-differentiable-at has-field-derivative-scaleR-right*)

**lemma** *deriv-scaleR-deriv-right*:
**assumes** *f twice-field-differentiable-at z*
**shows** *deriv* ($\lambda x.$ *k* $*_R$ *deriv f x*) *z* = *k* $*_R$ *deriv* (*deriv f*) *z*
**using** *assms deriv-scaleR-right twice-field-differentiable-at-def* **by** *blast*

**lemma** *twice-field-differentiable-at-scaleR* [*simp*, *intro*]:
($*_R$) *k twice-field-differentiable-at x*
**proof** −
**have** $\forall x \in UNIV$. ($*_R$) *k field-differentiable at x*
**by** (*simp add*: *field-differentiable-scaleR-right*)
**moreover have** *deriv* (($*_R$) *k*) *field-differentiable at x*
**by** *simp*
**ultimately show** *?thesis*
**unfolding** *twice-field-differentiable-at-def field-differentiable-on-def*
**by** *auto*
**qed**

**lemma** *twice-field-differentiable-at-scaleR-fun* [*simp*, *intro*]:
**assumes** *f twice-field-differentiable-at x*
**shows** ($\lambda x.$ *k* $*_R$ *f x*) *twice-field-differentiable-at x*
**by** (*simp add*: *assms twice-field-differentiable-at-compose*)

### 2.2.5 Addition

**lemma** *eventually-deriv-add*:
**assumes** *f twice-field-differentiable-at x*
**and** *g twice-field-differentiable-at x*
**shows** $\forall_F$ *x in nhds x*. *deriv* ($\lambda x.$ *f x* + *g x*) *x* = *deriv f x* + *deriv g x*
**proof** −
**obtain** *S* **where** *Df-on-S*: *f field-differentiable-on S* **and** *x-int-S*: *x* $\in$ *interior S*
**using** *assms twice-field-differentiable-at-def* **by** *blast*
**obtain** *S'* **where** *Dg-on-S*: *g field-differentiable-on S'* **and** *x-int-S'*: *x* $\in$ *interior S'*
**using** *assms twice-field-differentiable-at-def* **by** *blast*

**have** $x \in$ *interior* $(S \cap S')$
**by** (*simp add*: *x-int-S x-int-S'*)
**moreover have** *Df-on-SS'*: *f field-differentiable-on* $(S \cap S')$
**by** (*meson Df-on-S IntD1 field-differentiable-on-def field-differentiable-within-subset inf-sup-ord*(*1*))
**moreover have** *Dg-on-SS'*: *g field-differentiable-on* $(S \cap S')$
**by** (*meson Dg-on-S IntD2 field-differentiable-on-def field-differentiable-within-subset inf-le2*)
**moreover have** *open* (*interior* $(S \cap S')$)
**by** *blast*
**moreover have** $\forall x \in$ *interior* $(S \cap S')$. *deriv* $(\lambda x.\ f\ x + g\ x)\ x =$ *deriv* $f\ x +$ *deriv* $g\ x$
**by** (*metis* (*full-types*) *Df-on-SS' Dg-on-SS' at-within-interior deriv-add field-differentiable-on-def in-mono interior-subset*)
**ultimately show** *?thesis*
**using** *eventually-nhds* **by** *blast*
**qed**

**lemma** *twice-field-differentiable-at-add* [*intro*]:
**assumes** *f twice-field-differentiable-at x*
**and** *g twice-field-differentiable-at x*
**shows** $(\lambda x.\ f\ x + g\ x)$ *twice-field-differentiable-at x*
**proof** −
**obtain** $S\ S'$
**where** *Df-on-S*: *f field-differentiable-on S* **and** *x-int-S*: $x \in$ *interior S*
**and** *Dg-on-S'*: *g field-differentiable-on S'* **and** *x-int-S'*: $x \in$ *interior S'*
**using** *assms twice-field-differentiable-at-def* **by** *blast*

**let** *?T* = *interior* $(S \cap S')$

**have** *x-int-T*: $x \in$ *interior ?T*
**by** (*simp add*: *x-int-S x-int-S'*)

**have** *Df-on-T*: *f field-differentiable-on ?T*
**by** (*meson Df-on-S field-differentiable-on-subset inf-sup-ord*(*1*) *interior-subset*)
**have** *Dg-on-fT*: *g field-differentiable-on ?T*
**by** (*meson Dg-on-S' field-differentiable-on-subset interior-subset le-infE*)

**have** $(\lambda x.\ f\ x + g\ x)$ *field-differentiable-on ?T*
**unfolding** *field-differentiable-on-def*
**proof**
**fix** *x* **assume** *x-in-T*: $x \in$ *?T*
**have** *f field-differentiable at x*
**by** (*metis x-in-T Df-on-T at-within-open field-differentiable-on-def open-interior*)
**moreover have** *g field-differentiable at x*
**by** (*metis x-in-T Dg-on-fT at-within-open field-differentiable-on-def open-interior*)
**ultimately have** $(\lambda x.\ f\ x + g\ x)$ *field-differentiable at x*
**by** (*simp add*: *field-differentiable-add*)
**then show** $(\lambda x.\ f\ x + g\ x)$ *field-differentiable at x within ?T*

```
    using field-differentiable-at-within by blast
  qed
  moreover have deriv (λx. f x + g x) field-differentiable at x
  proof -
    have deriv (λx. f x + g x) x = deriv f x + deriv g x
      by (simp add: assms once-field-differentiable-at)
    moreover have (λx. deriv f x + deriv g x) field-differentiable at x
      by (simp add: field-differentiable-add assms deriv-field-differentiable-at)
    ultimately show ?thesis
      using assms DERIV-deriv-iff-field-differentiable
            DERIV-cong-ev[of x x deriv (λx. f x + g x) λx. deriv f x + deriv g x]
      by (simp add: eventually-deriv-add field-differentiable-def)
  qed
  ultimately show ?thesis
    using twice-field-differentiable-at-def x-int-T by blast
qed

lemma deriv-add-id-const [simp]:
  deriv (λx. x + a) = (λz. 1)
  using ext trans[OF deriv-add] by force

lemma deriv-deriv-add-id-const [simp]:
  deriv (deriv (λx. x + a)) z = 0
  by simp

lemma twice-field-differentiable-at-cadd [simp]:
  (λx. x + a) twice-field-differentiable-at x
proof -
  have ∀x∈UNIV. (λx. x + a) field-differentiable at x
    by (simp add: field-differentiable-add)
  moreover have deriv ((λx. x + a)) field-differentiable at x
    by (simp add: ext)
  ultimately show ?thesis
    unfolding twice-field-differentiable-at-def field-differentiable-on-def
    by auto
qed
```

### 2.2.6 Linear Function

```
lemma twice-field-differentiable-at-linear [simp, intro]:
  (λx. k ∗ x + a) twice-field-differentiable-at x
proof -
  have ∀x∈UNIV. (λx. k ∗ x + a) field-differentiable at x
    by (simp add: field-differentiable-add)
  moreover have deriv ((λx. k ∗ x + a)) field-differentiable at x
  proof -
    have deriv ((λx. k ∗ x + a)) = (λx. k)
      by (simp add: ext)
    then show ?thesis
```

**by** *simp*
**qed**
**ultimately show** *?thesis*
**unfolding** *twice-field-differentiable-at-def field-differentiable-on-def*
**by** *auto*
**qed**

**lemma** *twice-field-differentiable-at-linearR* [*simp*, *intro*]:
($\lambda x.\ k *_R x + a$) *twice-field-differentiable-at x*
**proof** –
**have** $\forall x \in UNIV$. ($\lambda x.\ k *_R x + a$) *field-differentiable at x*
**by** (*simp add*: *field-differentiable-scaleR-right field-differentiable-add*)
**moreover have** *deriv* (($\lambda x.\ k *_R x + a$)) *field-differentiable at x*
**proof** –
**have** *deriv* (($\lambda x.\ k *_R x + a$)) = ($\lambda x.\ k *_R 1$)
**by** (*simp add*: *ext once-field-differentiable-at*)
**then show** *?thesis*
**by** *simp*
**qed**
**ultimately show** *?thesis*
**unfolding** *twice-field-differentiable-at-def field-differentiable-on-def*
**by** *auto*
**qed**

### 2.2.7 Multiplication

**lemma** *eventually-deriv-mult*:
**assumes** *f twice-field-differentiable-at x*
**and** *g twice-field-differentiable-at x*
**shows** $\forall_F$ *x in nhds x*. *deriv* ($\lambda x.\ f\ x * g\ x$) $x = f\ x *$ *deriv g x* + *deriv f x* $* g\ x$
**proof** –
**obtain** $S$ **and** $S'$
**where** *f field-differentiable-on S* **and** *in-S*: $x \in$ *interior S*
**and** *g field-differentiable-on* $S'$ **and** *in-S'*: $x \in$ *interior* $S'$
**using** *assms twice-field-differentiable-at-def* **by** *blast*
**then have** *Df-on-SS'*: *f field-differentiable-on* ($S \cap S'$)
**and** *Dg-on-SS'*: *g field-differentiable-on* ($S \cap S'$)
**using** *field-differentiable-on-subset* **by** *blast+*

**have** $\forall x \in$ *interior* ($S \cap S'$). *deriv* ($\lambda x.\ f\ x * g\ x$) $x = f\ x *$ *deriv g x* + *deriv f x* $* g\ x$
**proof**
**fix** $x$ **assume** $x \in$ *interior* ($S \cap S'$)
**then have** *f field-differentiable* (*at x*)
**and** *g field-differentiable* (*at x*)
**using** *Df-on-SS' Dg-on-SS' field-differentiable-on-def at-within-interior interior-subset subsetD* **by** *metis+*
**then show** *deriv* ($\lambda x.\ f\ x * g\ x$) $x = f\ x *$ *deriv g x* + *deriv f x* $* g\ x$
**by** *simp*

**qed**
**moreover have** *x* ∈ *interior* (*S* ∩ *S*′)
**by** (*simp add*: *in-S in-S*′)
**moreover have** *open* (*interior* (*S* ∩ *S*′))
**by** *blast*
**ultimately show** *?thesis*
**using** *eventually-nhds* **by** *blast*
**qed**

**lemma** *twice-field-differentiable-at-mult* [*intro*]:
**assumes** *f twice-field-differentiable-at x*
**and** *g twice-field-differentiable-at x*
**shows** (*λx. f x* ∗ *g x*) *twice-field-differentiable-at x*
**proof** −
**obtain** *S S*′
**where** *Df-on-S*: *f field-differentiable-on S* **and** *x-int-S*: *x* ∈ *interior S*
**and** *Dg-on-S*′: *g field-differentiable-on S*′ **and** *x-int-S*′: *x* ∈ *interior S*′
**using** *assms twice-field-differentiable-at-def* **by** *blast*

**let** *?T* = *interior* (*S* ∩ *S*′)

**have** *x-int-T*: *x* ∈ *interior ?T*
**by** (*simp add*: *x-int-S x-int-S*′)

**have** *Df-on-T*: *f field-differentiable-on ?T*
**by** (*meson Df-on-S field-differentiable-on-subset inf-sup-ord*(*1*) *interior-subset*)
**have** *Dg-on-fT*: *g field-differentiable-on ?T*
**by** (*meson Dg-on-S*′ *field-differentiable-on-subset interior-subset le-infE*)

**have** (*λx. f x* ∗ *g x*) *field-differentiable-on ?T*
**unfolding** *field-differentiable-on-def*
**proof**
**fix** *x* **assume** *x-in-T*: *x* ∈ *?T*
**have** *f field-differentiable at x*
**by** (*metis x-in-T Df-on-T at-within-open field-differentiable-on-def open-interior*)
**moreover have** *g field-differentiable at x*
**by** (*metis x-in-T Dg-on-fT at-within-open field-differentiable-on-def open-interior*)
**ultimately have** (*λx. f x* ∗ *g x*) *field-differentiable at x*
**by** (*simp add*: *field-differentiable-mult*)
**then show** (*λx. f x* ∗ *g x*) *field-differentiable at x within ?T*
**using** *field-differentiable-at-within* **by** *blast*
**qed**
**moreover have** *deriv* (*λx. f x* ∗ *g x*) *field-differentiable at x*
**proof** −
**have** *deriv* (*λx. f x* ∗ *g x*) *x* = *f x* ∗ *deriv g x* + *deriv f x* ∗ *g x*
**by** (*simp add*: *assms once-field-differentiable-at*)
**moreover have** (*λx. f x* ∗ *deriv g x* + *deriv f x* ∗ *g x*) *field-differentiable at x*
**by** (*rule field-differentiable-add*, *simp-all add*: *field-differentiable-mult assms once-field-differentiable-at deriv-field-differentiable-at*)

**ultimately show** *?thesis*
**using** *assms DERIV-deriv-iff-field-differentiable*
*DERIV-cong-ev*[*of x x deriv* (*λx. f x ∗ g x*) *λx. f x ∗ deriv g x* + *deriv f x ∗ g x*]
**by** (*simp add*: *eventually-deriv-mult field-differentiable-def*)
**qed**
**ultimately show** *?thesis*
**using** *twice-field-differentiable-at-def x-int-T* **by** *blast*
**qed**

### 2.2.8 Sine and Cosine

**lemma** *deriv-sin* [*simp*]: *deriv sin a* = *cos a*
**by** (*simp add*: *DERIV-imp-deriv*)

**lemma** *deriv-sinf* [*simp*]: *deriv sin* = (*λx. cos x*)
**by** *auto*

**lemma** *deriv-cos* [*simp*]: *deriv cos a* = − *sin a*
**by** (*simp add*: *DERIV-imp-deriv*)

**lemma** *deriv-cosf* [*simp*]: *deriv cos* = (*λx.* − *sin x*)
**by** *auto*

**lemma** *deriv-sin-minus* [*simp*]:
*deriv* (*λx.* − *sin x*) *a* = − *deriv* (*λx. sin x*) *a*
**by** (*simp add*: *DERIV-imp-deriv Deriv.field-differentiable-minus*)

**lemma** *twice-field-differentiable-at-sin* [*simp*, *intro*]:
*sin twice-field-differentiable-at x*
**by** (*auto intro*!: *exI* [*of - UNIV*] *simp add*: *field-differentiable-at-sin field-differentiable-on-def twice-field-differentiable-at-def field-differentiable-at-cos*)

**lemma** *twice-field-differentiable-at-sin-fun* [*intro*]:
**assumes** *f twice-field-differentiable-at x*
**shows** (*λx. sin* (*f x*)) *twice-field-differentiable-at x*
**by** (*simp add*: *assms twice-field-differentiable-at-compose*)

**lemma** *twice-field-differentiable-at-cos* [*simp*, *intro*]:
*cos twice-field-differentiable-at x*
**by** (*auto intro*!: *exI* [*of - UNIV*] *simp add*: *field-differentiable-within-sin field-differentiable-minus field-differentiable-on-def twice-field-differentiable-at-def field-differentiable-at-cos*)

**lemma** *twice-field-differentiable-at-cos-fun* [*intro*]:
**assumes** *f twice-field-differentiable-at x*
**shows** (*λx. cos* (*f x*)) *twice-field-differentiable-at x*
**by** (*simp add*: *assms twice-field-differentiable-at-compose*)

### 2.2.9 Exponential

**lemma** *deriv-exp* [*simp*]: *deriv exp x* = *exp x*
  **using** *DERIV-exp DERIV-imp-deriv* **by** *blast*

**lemma** *deriv-expf* [*simp*]: *deriv exp* = *exp*
  **by** (*simp add*: *ext*)

**lemma** *deriv-deriv-exp* [*simp*]: *deriv* (*deriv exp*) *x* = *exp x*
  **by** *simp*

**lemma** *twice-field-differentiable-at-exp* [*simp*, *intro*]:
  *exp twice-field-differentiable-at x*
**proof** −
  **have** ∀ *x*∈*UNIV*. *exp field-differentiable at x*
   **and** *deriv exp field-differentiable at x*
    **by** (*simp-all add*: *field-differentiable-within-exp*)
  **then show** *?thesis*
    **unfolding** *twice-field-differentiable-at-def field-differentiable-on-def*
    **by** *auto*
**qed**

**lemma** *twice-field-differentiable-at-exp-fun* [*simp*, *intro*]:
  **assumes** *f twice-field-differentiable-at x*
  **shows** (*λx*. *exp* (*f x*)) *twice-field-differentiable-at x*
  **by** (*simp add*: *assms twice-field-differentiable-at-compose*)

### 2.2.10 Square Root

**lemma** *deriv-real-sqrt* [*simp*]: *x* > *0* ⟹ *deriv sqrt x* = *inverse* (*sqrt x*) / *2*
  **using** *DERIV-imp-deriv DERIV-real-sqrt* **by** *blast*

**lemma** *has-real-derivative-inverse-sqrt*:
  **assumes** *x* > *0*
  **shows** ((*λx*. *inverse* (*sqrt x*) / *2*) *has-real-derivative* − (*inverse* (*sqrt x* ^ *3*) / *4*)) (*at x*)
**proof** −
  **have** *inv-sqrt-mult*: (*inverse* (*sqrt x*)/*2*) ∗ (*sqrt x* ∗ *2*) = *1*
    **using** *assms* **by** *simp*
  **have** *inv-sqrt-mult2*: (− *inverse* ((*sqrt x*)^*3*)/*2*)∗ *x* ∗ (*sqrt x* ∗ *2*) = −*1*
    **using** *assms* **by** (*simp add*: *field-simps power3-eq-cube*)
  **then show** *?thesis*
    **using** *assms* **by** (*safe intro*!: *DERIV-imp-deriv derivative-eq-intros*)
      (*auto intro*: *derivative-eq-intros inv-sqrt-mult* [*THEN ssubst*] *inv-sqrt-mult2* [*THEN ssubst*]
        *simp add*: *divide-simps power3-eq-cube*)
**qed**

**lemma** *deriv-deriv-real-sqrt′*:
  **assumes** *x* > *0*

**shows** *deriv* (*λx. inverse* (*sqrt x*) / *2*) *x* = − *inverse* ((*sqrt x*)^*3*)/*4*
**by** (*simp add*: *DERIV-imp-deriv assms has-real-derivative-inverse-sqrt*)

**lemma** *has-real-derivative-deriv-sqrt*:
**assumes** *x* > *0*
**shows** (*deriv sqrt has-real-derivative* − *inverse* (*sqrt x* ^ *3*) / *4*) (*at x*)
**proof** −
**have** ((*λx. inverse* (*sqrt x*) / *2*) *has-real-derivative* − *inverse* (*sqrt x* ^ *3*) / *4*) (*at x*)
**using** *assms has-real-derivative-inverse-sqrt* **by** *auto*
**moreover**
{**fix** *xa* :: *real*
**assume** *xa* ∈ {*0*<..}
**then have** *inverse* (*sqrt xa*) / *2* = *deriv sqrt xa*
**by** *simp*
}
**ultimately show** *?thesis*
**using** *has-field-derivative-transform-within-open* [**where** *S*={*0* <..} **and** *f*=(*λx. inverse* (*sqrt x*) / *2*)]
**by** (*meson assms greaterThan-iff open-greaterThan*)
**qed**

**lemma** *deriv-deriv-real-sqrt* [*simp*]:
**assumes** *x* > *0*
**shows** *deriv*(*deriv sqrt*) *x* = − *inverse* ((*sqrt x*)^*3*)/*4*
**using** *DERIV-imp-deriv assms has-real-derivative-deriv-sqrt* **by** *blast*

**lemma** *twice-field-differentiable-at-sqrt* [*simp*, *intro*]:
**assumes** *x* > *0*
**shows** *sqrt twice-field-differentiable-at x*
**proof** −
**have** *sqrt field-differentiable-on* {*0*<..}
**by** (*metis DERIV-real-sqrt at-within-open field-differentiable-def field-differentiable-on-def greaterThan-iff open-greaterThan*)
**moreover have** *x* ∈ *interior* {*0*<..}
**by** (*metis assms greaterThan-iff interior-interior interior-real-atLeast*)
**moreover have** *deriv sqrt field-differentiable at x*
**using** *assms field-differentiable-def has-real-derivative-deriv-sqrt* **by** *blast*
**ultimately show** *?thesis*
**using** *twice-field-differentiable-at-def* **by** *blast*
**qed**

**lemma** *twice-field-differentiable-at-sqrt-fun* [*intro*]:
**assumes** *f twice-field-differentiable-at x*
**and** *f x* > *0*
**shows** (*λx. sqrt* (*f x*)) *twice-field-differentiable-at x*
**by** (*simp add*: *assms*(*1*) *assms*(*2*) *twice-field-differentiable-at-compose*)

### 2.2.11 Natural Power

**lemma** *field-differentiable-power* [*simp*]:
  ($\lambda x.\ x$ ^ $n$) *field-differentiable at x*
  **using** *DERIV-power DERIV-ident field-differentiable-def*
  **by** *blast*

**lemma** *deriv-power-fun* [*simp*]:
  **assumes** *f field-differentiable at x*
    **shows** *deriv* ($\lambda x.$ *f x* ^ *n*) *x* = *of-nat n* ∗ *deriv f x* ∗ *f x* ^ (*n* − *1*)
  **using** *DERIV-power*[*of f deriv f x*]
  **by** (*simp add*: *DERIV-imp-deriv assms field-differentiable-derivI mult.assoc* [*symmetric*])

**lemma** *deriv-power* [*simp*]:
  *deriv* ($\lambda x.$ *x* ^ *n*) *x* = *of-nat n* ∗ *x* ^ (*n* − *1*)
  **using** *DERIV-power*[*of* $\lambda x.$ *x 1*] *DERIV-imp-deriv* **by** *force*

**lemma** *deriv-deriv-power* [*simp*]:
  *deriv* (*deriv* ($\lambda x.$ *x* ^ *n*)) *x* = *of-nat n* ∗ *of-nat* (*n* − *Suc 0*) ∗ *x* ^ (*n* − *2*)
**proof** −
  **have** ($\lambda x.$ *x* ^ (*n* − *1*)) *field-differentiable at x*
    **by** *simp*
  **then have** *deriv* ($\lambda x.$ *of-nat n* ∗ *x* ^ (*n* − *1*)) *x* = *of-nat n* ∗ *of-nat* (*n* − *Suc 0*) ∗ *x* ^ (*n* − *2*)
    **by** (*simp add*: *diff-diff-add mult.assoc numeral-2-eq-2*)
  **then show** *?thesis*
    **by** (*simp add*: *ext*[*OF deriv-power*])
**qed**

**lemma** *twice-field-differentiable-at-power* [*simp*, *intro*]:
  ($\lambda x.$ *x* ^ *n*) *twice-field-differentiable-at x*
**proof** −
  **have** ∀ *x*∈*UNIV*. ($\lambda x.$ *x* ^ *n*) *field-differentiable at x*
    **by** *simp*
  **moreover have** *deriv* (($\lambda x.$ *x* ^ *n*)) *field-differentiable at x*
  **proof** −
    **have** *deriv* (($\lambda x.$ *x* ^ *n*)) = ($\lambda x.$ *of-nat n* ∗ *x* ^ (*n* − *1*))
      **by** (*simp add*: *ext*)
    **then show** *?thesis*
      **using** *field-differentiable-mult*[*of* $\lambda x.$ *of-nat n x UNIV* $\lambda x.$ *x* ^ (*n* − *1*)]
      **by** (*simp add*: *field-differentiable-caratheodory-at*)
  **qed**
  **ultimately show** *?thesis*
    **unfolding** *twice-field-differentiable-at-def field-differentiable-on-def*
    **by** *force*
**qed**

**lemma** *twice-field-differentiable-at-power-fun* [*intro*]:
  **assumes** *f twice-field-differentiable-at x*

**shows** ($\lambda x.$ *f x* ^ *n*) *twice-field-differentiable-at x*
**by** (*blast intro*: *assms twice-field-differentiable-at-compose* [*OF* - *twice-field-differentiable-at-power*])

### 2.2.12 Inverse

**lemma** *eventually-deriv-inverse*:
**assumes** $x \neq 0$
**shows** $\forall_F$ *x in nhds x. deriv inverse x* = − *1* / (*x* ^ *2*)
**proof** −
**obtain** *T* **where** *open-T*: *open T* **and** $\forall z \in T.\ z \neq 0$ **and** *x-in-T*: $x \in T$
**using** *assms t1-space* **by** *blast*

**then have** $\forall x \in T$. *deriv inverse x* = − *1* / (*x* ^ *2*)
**by** *simp*
**then show** *?thesis*
**using** *eventually-nhds open-T x-in-T* **by** *blast*
**qed**

**lemma** *deriv-deriv-inverse* [*simp*]:
**assumes** $x \neq 0$
**shows** *deriv* (*deriv inverse*) *x* = *2* ∗ *inverse* (*x* ^ *3*)
**proof** −
**have** *deriv* ($\lambda x.$ *inverse* (*x* ^ *2*)) *x* = − (*of-nat 2* ∗ *x*) / ((*x* ^ *2*) ^ *2*)
**using** *assms* **by** *simp*
**moreover have** ($\lambda x.$ *inverse* (*x* ^ *2*)) *field-differentiable at x*
**using** *assms* **by** (*simp add*: *field-differentiable-inverse*)
**ultimately have** *deriv* ($\lambda x.$ − (*inverse* (*x* ^ *2*))) *x* = *of-nat 2* ∗ *x* / (*x* ^ *4*)
**using** *deriv-chain*[*of* $\lambda x.$ *inverse* (*x* ^ *2*) *x*]
**by** (*simp add*: *comp-def field-differentiable-minus field-simps*)
**then have** *deriv* ($\lambda x.$ − *1* / (*x* ^ *2*)) *x* = *2* ∗ *inverse* (*x* ^ *3*)
**by** (*simp add*: *power4-eq-xxxx power3-eq-cube field-simps*)
**then show** *?thesis*
**using** *assms eventually-deriv-inverse deriv-cong-ev* **by** *fastforce*
**qed**

**lemma** *twice-field-differentiable-at-inverse* [*simp*, *intro*]:
**assumes** $x \neq 0$
**shows** *inverse twice-field-differentiable-at x*
**proof** −
**obtain** *T* **where** *zero-T*: $0 \notin T$ **and** *x-in-T*: $x \in T$ **and** *open-T*: *open T*
**using** *assms t1-space* **by** *blast*
**then have** $T \subseteq \{z.\ z \neq 0\}$
**by** *blast*
**then have** $\forall x \in T$. *inverse field-differentiable at x within T*
**using** *DERIV-inverse field-differentiable-def* **by** *blast*
**moreover have** *deriv inverse field-differentiable at x*
**proof** −
**have** ($\lambda x.$ − *inverse* (*x* ^ *2*)) *field-differentiable at x*
**using** *assms* **by** (*simp add*: *field-differentiable-inverse field-differentiable-minus*)

**then have** ($\lambda x.\ -\ 1\ /\ (x\ \hat{}\ 2)$) *field-differentiable at x*
**by** (*simp add*: *inverse-eq-divide*)
**then show** *?thesis*
**using** *eventually-deriv-inverse*[*OF assms*]
**by** (*simp add*: *DERIV-cong-ev field-differentiable-def*)
**qed**
**moreover have** $x \in$ *interior T*
**by** (*simp add*: *x-in-T open-T interior-open*)
**ultimately show** *?thesis*
**unfolding** *twice-field-differentiable-at-def field-differentiable-on-def*
**by** *blast*
**qed**

**lemma** *twice-field-differentiable-at-inverse-fun* [*simp*, *intro*]:
**assumes** *f twice-field-differentiable-at x*
*f x* $\neq$ *0*
**shows** ($\lambda x.$ *inverse* (*f x*)) *twice-field-differentiable-at x*
**by** (*simp add*: *assms twice-field-differentiable-at-compose*)

**lemma** *twice-field-differentiable-at-divide* [*intro*]:
**assumes** *f twice-field-differentiable-at x*
**and** *g twice-field-differentiable-at x*
**and** *g x* $\neq$ *0*
**shows** ($\lambda x.\ f\ x\ /\ g\ x$) *twice-field-differentiable-at x*
**by** (*simp add*: *assms divide-inverse twice-field-differentiable-at-mult*)

### 2.2.13 Polynomial

**lemma** *twice-field-differentiable-at-polyn* [*simp*, *intro*]:
**fixes** *coef* :: *nat* $\Rightarrow$ *'a* :: {*real-normed-field*}
**and** *n* :: *nat*
**shows** ($\lambda x.\ \sum i<n.\ coef\ i * x\ \hat{}\ i$) *twice-field-differentiable-at x*
**proof** (*induction n*)
**case** *0*
**then show** *?case*
**by** *simp*
**next**
**case** *hyp*: (*Suc n*)
**show** *?case*
**proof** (*simp*, *rule twice-field-differentiable-at-add*)
**show** ($\lambda x.\ \sum i<n.\ coef\ i * x\ \hat{}\ i$) *twice-field-differentiable-at x*
**by** (*rule hyp*)
**show** ($\lambda x.\ coef\ n * x\ \hat{}\ n$) *twice-field-differentiable-at x*
**using** *twice-field-differentiable-at-compose*[*of* $\lambda x.\ x\ \hat{}\ n\ x\ (*)\ (coef\ n)$]
**by** *simp*
**qed**
**qed**

**lemma** *twice-field-differentiable-at-polyn-fun* [*simp*]:

**fixes** *coef* :: *nat* ⇒ *'a* :: {*real-normed-field*}
**and** *n* :: *nat*
**assumes** *f twice-field-differentiable-at x*
**shows** (*λx*. ∑ *i*<*n*. *coef i* ∗ *f x* ^ *i*) *twice-field-differentiable-at x*
**by** (*blast intro*: *assms twice-field-differentiable-at-compose* [*OF* - *twice-field-differentiable-at-polyn*])

**end**

# 3 Hyperdual Extension of Functions

**theory** *HyperdualFunctionExtension*
**imports** *Hyperdual TwiceFieldDifferentiable*
**begin**

The following is an important fact in the derivation of the hyperdual extension.

**lemma**
**fixes** *x* :: (*'a* :: *comm-ring-1*) *hyperdual* **and** *n* :: *nat*
**assumes** *Base x* = *0*
**shows** *x* ^ (*n* + *3*) = *0*
**proof** (*induct n*)
**case** *0*
**then show** *?case*
**using** *assms hyperdual-power*[*of x 3*] **by** *simp*
**next**
**case** (*Suc n*)
**then show** *?case*
**using** *assms power-Suc*[*of x n* + *3*] *mult-zero-right add-Suc* **by** *simp*
**qed**

We define the extension of a function to the hyperdual numbers.

**primcorec** *hypext* :: ((*'a* :: *real-normed-field*) ⇒ *'a*) ⇒ *'a hyperdual* ⇒ *'a hyperdual* (‹∗*h*∗ -› [*80*] *80*)
**where**
*Base* ((∗*h*∗ *f*) *x*) = *f* (*Base x*)
| *Eps1* ((∗*h*∗ *f*) *x*) = *Eps1 x* ∗ *deriv f* (*Base x*)
| *Eps2* ((∗*h*∗ *f*) *x*) = *Eps2 x* ∗ *deriv f* (*Base x*)
| *Eps12* ((∗*h*∗ *f*) *x*) = *Eps12 x* ∗ *deriv f* (*Base x*) + *Eps1 x* ∗ *Eps2 x* ∗ *deriv* (*deriv f*) (*Base x*)

This has the expected behaviour when expressed in terms of the units.

**lemma** *hypext-Hyperdual-eq*:
(∗*h*∗ *f*) (*Hyperdual a b c d*) =
*Hyperdual* (*f a*) (*b* ∗ *deriv f a*) (*c* ∗ *deriv f a*) (*d* ∗ *deriv f a* + *b* ∗ *c* ∗ *deriv* (*deriv f*) *a*)
**by** (*simp add*: *hypext.code*)

**lemma** *hypext-Hyperdual-eq-parts*:

$(*h*\ f)\ (Hyperdual\ a\ b\ c\ d) =$
$f\ a *_H\ ba + (b * deriv\ f\ a) *_H\ e1 + (c * deriv\ f\ a) *_H\ e2 +$
$(d * deriv\ f\ a + b * c * deriv\ (deriv\ f)\ a) *_H\ e12$
**by** (*metis Hyperdual-eq hypext-Hyperdual-eq*)

The extension can be used to extract the function value, and first and second derivatives at x when applied to $x *_H\ re + e1 + e2 + (0::'a) *_H\ e12$, which we denote by $\beta\ x$.

**definition** *hyperdualx* :: $('a :: real\text{-}normed\text{-}field) \Rightarrow 'a\ hyperdual\ (\beta)$
**where** $\beta\ x = (Hyperdual\ x\ 1\ 1\ 0)$

**lemma** *hyperdualx-sel* [*simp*]:
**shows** $Base\ (\beta\ x) = x$
**and** $Eps1\ (\beta\ x) = 1$
**and** $Eps2\ (\beta\ x) = 1$
**and** $Eps12\ (\beta\ x) = 0$
**by** (*simp-all add*: *hyperdualx-def*)

**lemma** *hypext-extract-eq*:
$(*h*\ f)\ (\beta\ x) = f\ x *_H\ ba + deriv\ f\ x *_H\ e1 + deriv\ f\ x *_H\ e2 + deriv\ (deriv\ f)\ x *_H\ e12$
**by** (*simp add*: *hypext-Hyperdual-eq-parts hyperdualx-def*)

**lemma** *Base-hypext*:
$Base\ ((*h*\ f)\ (\beta\ x)) = f\ x$
**by** (*simp add*: *hyperdualx-def*)

**lemma** *Eps1-hypext*:
$Eps1\ ((*h*\ f)\ (\beta\ x)) = deriv\ f\ x$
**by** (*simp add*: *hyperdualx-def*)

**lemma** *Eps2-hypext*:
$Eps2\ ((*h*\ f)\ (\beta\ x)) = deriv\ f\ x$
**by** (*simp add*: *hyperdualx-def*)

**lemma** *Eps12-hypext*:
$Eps12\ ((*h*\ f)\ (\beta\ x)) = deriv\ (deriv\ f)\ x$
**by** (*simp add*: *hyperdualx-def*)

### 3.0.1 Convenience Interface

Define a datatype to hold the function value, and the first and second derivative values.

**datatype** (*'a* :: *real-normed-field*) *derivs* = *Derivs* (*Value*: *'a*) (*First*: *'a*) (*Second*: *'a*)

Then we convert a hyperdual number to derivative values by extracting the base component, one of the first-order components, and the second-order component.

**fun** *hyperdual-to-derivs* :: (*'a* :: *real-normed-field*) *hyperdual* ⇒ *'a derivs*
**where** *hyperdual-to-derivs x* = *Derivs* (*Base x*) (*Eps1 x*) (*Eps12 x*)

Finally we define way of converting any compatible function into one that yields the value and the derivatives.

**fun** *autodiff* :: (*'a* :: *real-normed-field* ⇒ *'a*) ⇒ *'a* ⇒ *'a derivs*
**where** *autodiff f* = (λ*x*. *hyperdual-to-derivs* ((∗*h*∗ *f*) (β *x*)))

**lemma** *autodiff-sel*:
*Value* (*autodiff f x*) = *Base* ((∗*h*∗ *f*) (β *x*))
*First* (*autodiff f x*) = *Eps1* ((∗*h*∗ *f*) (β *x*))
*Second* (*autodiff f x*) = *Eps12* ((∗*h*∗ *f*) (β *x*))
**by** *simp-all*

The result contains the expected values.

**lemma** *autodiff-extract-value*:
*Value* (*autodiff f x*) = *f x*
**by** (*simp del*: *hypext.simps add*: *Base-hypext*)

**lemma** *autodiff-extract-first*:
*First* (*autodiff f x*) = *deriv f x*
**by** (*simp del*: *hypext.simps add*: *Eps1-hypext*)

**lemma** *autodiff-extract-second*:
*Second* (*autodiff f x*) = *deriv* (*deriv f*) *x*
**by** (*simp del*: *hypext.simps add*: *Eps12-hypext*)

The derivative components of the result are actual derivatives if the function is sufficiently differentiable on that argument.

**lemma** *autodiff-first-derivative*:
**assumes** *f field-differentiable* (*at x*)
**shows** (*f has-field-derivative First* (*autodiff f x*)) (*at x*)
**by** (*simp add*: *autodiff-extract-first DERIV-deriv-iff-field-differentiable assms*)

**lemma** *autodiff-second-derivative*:
**assumes** *f twice-field-differentiable-at x*
**shows** ((*deriv f*) *has-field-derivative Second* (*autodiff f x*)) (*at x*)
**by** (*simp add*: *autodiff-extract-second DERIV-deriv-iff-field-differentiable assms deriv-field-differentiable-at*)

### 3.0.2 Composition

Composition of hyperdual extensions is the hyperdual extension of composition:

**lemma** *hypext-compose*:
**assumes** *f twice-field-differentiable-at* (*Base x*)

**and** *g twice-field-differentiable-at* (*f* (*Base x*))
**shows** (*∗h∗* (λ*x*. *g* (*f x*))) *x* = (*∗h∗ g*) ((*∗h∗ f*) *x*)
**proof** (*simp add*: *hyperdual-eq-iff*, *intro conjI disjI2*)
**show** *goal1*: *deriv* (λ*x*. *g* (*f x*)) (*Base x*) = *deriv f* (*Base x*) ∗ *deriv g* (*f* (*Base x*))
**proof** −
**have** *deriv* (λ*x*. *g* (*f x*)) (*Base x*) = *deriv* (*g* ∘ *f*) (*Base x*)
**by** (*simp add*: *comp-def*)
**also have** ... = *deriv g* (*f* (*Base x*)) ∗ *deriv f* (*Base x*)
**using** *assms* **by** (*simp add*: *deriv-chain once-field-differentiable-at*)
**finally show** *?thesis*
**by** (*simp add*: *mult.commute deriv-chain*)
**qed**
**then show** *deriv* (λ*x*. *g* (*f x*)) (*Base x*) = *deriv f* (*Base x*) ∗ *deriv g* (*f* (*Base x*)) **.**

**have** *first-diff*: (λ*x*. *deriv g* (*f x*)) *field-differentiable at* (*Base x*)
**by** (*metis DERIV-chain2 assms deriv-field-differentiable-at field-differentiable-def once-field-differentiable-at*)

**have** *deriv* (*deriv g* ∘ *f*) (*Base x*) = *deriv* (*deriv g*) (*f* (*Base x*)) ∗ *deriv f* (*Base x*)
**using** *deriv-chain assms once-field-differentiable-at deriv-field-differentiable-at*
**by** *blast*
**then have** *deriv-deriv-comp*: *deriv* (λ*x*. *deriv g* (*f x*)) (*Base x*) = *deriv* (*deriv g*) (*f* (*Base x*)) ∗ *deriv f* (*Base x*)
**by** (*simp add*: *comp-def*)

**have** *deriv* (*deriv* (λ*x*. *g* (*f x*))) (*Base x*) = *deriv* ((λ*x*. *deriv f x* ∗ *deriv g* (*f x*))) (*Base x*)
**using** *assms eventually-deriv-compose′*[*of f Base x g*]
**by** (*simp add*: *mult.commute deriv-cong-ev*)
**also have** ... = *deriv f* (*Base x*) ∗ *deriv* (λ*x*. *deriv g* (*f x*)) (*Base x*) + *deriv* (*deriv f*) (*Base x*) ∗ *deriv g* (*f* (*Base x*))
**using** *assms*(*1*) *first-diff* **by** (*simp add*: *deriv-field-differentiable-at*)
**also have** ... = *deriv f* (*Base x*) ∗ *deriv* (*deriv g*) (*f* (*Base x*)) ∗ *deriv f* (*Base x*) + *deriv* (*deriv f*) (*Base x*) ∗ *deriv g* (*f* (*Base x*))
**using** *deriv-deriv-comp* **by** *simp*
**finally show** *Eps12 x* ∗ *deriv* (λ*x*. *g* (*f x*)) (*Base x*) + *Eps1 x* ∗ *Eps2 x* ∗ *deriv* (*deriv* (λ*x*. *g* (*f x*))) (*Base x*) =
(*Eps12 x* ∗ *deriv f* (*Base x*) + *Eps1 x* ∗ *Eps2 x* ∗ *deriv* (*deriv f*) (*Base x*)) ∗ *deriv g* (*f* (*Base x*)) +
*Eps1 x* ∗ *deriv f* (*Base x*) ∗ (*Eps2 x* ∗ *deriv f* (*Base x*)) ∗ *deriv* (*deriv g*) (*f* (*Base x*))
**by** (*simp add*: *goal1 field-simps*)
**qed**

## 3.1 Concrete Instances

### 3.1.1 Constant

Component embedding is an extension of the constant function.

**lemma** *hypext-const* [*simp*]:
 $(*h* (\lambda x.\ a))\ x = \textit{of-comp}\ a$
 **by** (*simp add*: *of-comp-def hyperdual-eq-iff*)

**lemma** *autodiff* $(\lambda x.\ a) = (\lambda x.\ \textit{Derivs}\ a\ 0\ 0)$
 **by** *simp*

### 3.1.2 Identity

Identity is an extension of the component identity.

**lemma** *hypext-ident*:
 $(*h* (\lambda x.\ x))\ x = x$
 **by** (*simp add*: *hyperdual-eq-iff*)

### 3.1.3 Component Scalar Multiplication

Component scaling is an extension of component constant multiplication:

**lemma** *hypext-scaleH*:
 $(*h* (\lambda x.\ k * x))\ x = k *_H x$
 **by** (*simp add*: *hyperdual-eq-iff*)

**lemma** *hypext-fun-scaleH*:
 **assumes** *f twice-field-differentiable-at* (*Base x*)
 **shows** $(*h* (\lambda x.\ k * f\ x))\ x = k *_H (*h* f)\ x$
 **using** *assms* **by** (*simp add*: *hypext-compose hypext-scaleH*)

Unary minus is just an instance of constant multiplication:

**lemma** *hypext-uminus*:
 $(*h*\ \textit{uminus})\ x = -\ x$
 **using** *hypext-scaleH*[*of* $-1$ *x*] **by** *simp*

### 3.1.4 Real Scalar Multiplication

Real scaling is an extension of component real scaling:

**lemma** *hypext-scaleR*:
 $(*h* (\lambda x.\ k *_R x))\ x = k *_R x$
 **by** (*auto simp add*: *hyperdual-eq-iff*)

**lemma** *hypext-fun-scaleR*:
 **assumes** *f twice-field-differentiable-at* (*Base x*)
 **shows** $(*h* (\lambda x.\ k *_R f\ x))\ x = k *_R (*h* f)\ x$
 **using** *assms* **by** (*simp add*: *hypext-compose hypext-scaleR*)

### 3.1.5 Addition

Addition of hyperdual extensions is a hyperdual extension of addition of functions.

**lemma** *hypext-fun-add*:
  **assumes** *f twice-field-differentiable-at* (*Base x*)
    **and** *g twice-field-differentiable-at* (*Base x*)
   **shows** (*∗h∗* (*λx. f x* + *g x*)) *x* = (*∗h∗ f*) *x* + (*∗h∗ g*) *x*
**proof** (*simp add*: *hyperdual-eq-iff distrib-left*[*symmetric*], *intro conjI disjI2*)
  **show** *goal1*: *deriv* (*λx. f x* + *g x*) (*Base x*) = *deriv f* (*Base x*) + *deriv g* (*Base x*)
   **by** (*simp add*: *assms once-field-differentiable-at distrib-left*)
  **then show** *deriv* (*λx. f x* + *g x*) (*Base x*) = *deriv f* (*Base x*) + *deriv g* (*Base x*) **.**

  **have** *deriv* (*deriv* (*λx. f x* + *g x*)) (*Base x*) = *deriv* (*λw. deriv f w* + *deriv g w*) (*Base x*)
   **by** (*simp add*: *assms deriv-cong-ev eventually-deriv-add*)
  **moreover have** *Eps12 x* ∗ *deriv f* (*Base x*) + *Eps12 x* ∗ *deriv g* (*Base x*) = *Eps12 x* ∗ *deriv* (*λx. f x* + *g x*) (*Base x*)
   **by** (*metis distrib-left goal1*)
  **ultimately show** *Eps12 x* ∗ *deriv* (*λx. f x* + *g x*) (*Base x*) +
   *Eps1 x* ∗ *Eps2 x* ∗ *deriv* (*deriv* (*λx. f x* + *g x*)) (*Base x*) =
   *Eps12 x* ∗ *deriv f* (*Base x*) + *Eps1 x* ∗ *Eps2 x* ∗ *deriv* (*deriv f*) (*Base x*) +
   (*Eps12 x* ∗ *deriv g* (*Base x*) + *Eps1 x* ∗ *Eps2 x* ∗ *deriv* (*deriv g*) (*Base x*))
   **using** *deriv-add*[*OF deriv-field-differentiable-at deriv-field-differentiable-at*, *OF assms*]
   **by** (*simp add*: *distrib-left add.left-commute*)
**qed**

**lemma** *hypext-cadd* [*simp*]:
  (*∗h∗* (*λx. x* + *a*)) *x* = *x* + *of-comp a*
  **by** (*auto simp add*: *hyperdual-eq-iff of-comp-def*)

**lemma** *hypext-fun-cadd*:
  **assumes** *f twice-field-differentiable-at* (*Base x*)
  **shows** (*∗h∗* (*λx. f x* + *a*)) *x* = (*∗h∗ f*) *x* + *of-comp a*
  **using** *assms hypext-compose*[*of f x λx. x* + *a*] **by** *simp*

### 3.1.6 Component Linear Function

Hyperdual linear function is an extension of the component linear function:

**lemma** *hypext-linear*:
  (*∗h∗* (*λx. k* ∗ *x* + *a*)) *x* = *k* $*_H$ *x* + *of-comp a*
  **using** *hypext-fun-add*[*of* (∗) *k x λx. a*]
  **by** (*simp add*: *hypext-scaleH*)

**lemma** *hypext-fun-linear*:
  **assumes** *f twice-field-differentiable-at* (*Base x*)

**shows** (*∗h∗* (*λx. k ∗ f x + a*)) *x* = *k* $*_H$ (*∗h∗ f*) *x* + *of-comp a*
**using** *assms hypext-compose*[*of f x λx. k ∗ x + a*] **by** (*simp add*: *hypext-linear*)

### 3.1.7 Real Linear Function

We have the same for real scaling instead of component multiplication:

**lemma** *hypext-linearR*:
(*∗h∗* (*λx. k* $*_R$ *x + a*)) *x* = *k* $*_R$ *x* + *of-comp a*
**using** *hypext-fun-add*[*of* ($*_R$) *k x λx. a*]
**by** (*simp add*: *hypext-scaleR*)

**lemma** *hypext-fun-linearR*:
**assumes** *f twice-field-differentiable-at* (*Base x*)
**shows** (*∗h∗* (*λx. k* $*_R$ *f x + a*)) *x* = *k* $*_R$ (*∗h∗ f*) *x* + *of-comp a*
**using** *assms hypext-compose*[*of f x λx. k* $*_R$ *x + a*] **by** (*simp add*: *hypext-linearR*)

### 3.1.8 Multiplication

Extension of multiplication is multiplication of the functions' extensions.

**lemma** *hypext-fun-mult*:
**assumes** *f twice-field-differentiable-at* (*Base x*)
**and** *g twice-field-differentiable-at* (*Base x*)
**shows** (*∗h∗* (*λz. f z ∗ g z*)) *x* = (*∗h∗ f*) *x* ∗ (*∗h∗ g*) *x*
**proof** (*simp add*: *hyperdual-eq-iff distrib-left*[*symmetric*], *intro conjI*)
**show** *Eps1 x ∗ deriv* (*λz. f z ∗ g z*) (*Base x*) =
*f* (*Base x*) ∗ (*Eps1 x ∗ deriv g* (*Base x*)) + *Eps1 x ∗ deriv f* (*Base x*) ∗ *g* (*Base x*)
**and** *Eps2 x ∗ deriv* (*λz. f z ∗ g z*) (*Base x*) =
*f* (*Base x*) ∗ (*Eps2 x ∗ deriv g* (*Base x*)) + *Eps2 x ∗ deriv f* (*Base x*) ∗ *g* (*Base x*)
**using** *assms* **by** (*simp-all add*: *once-field-differentiable-at distrib-left*)

**have**
*deriv* (*deriv* (*λz. f z ∗ g z*)) (*Base x*) =
*f* (*Base x*) ∗ *deriv* (*deriv g*) (*Base x*) + *2* ∗ *deriv f* (*Base x*) ∗ *deriv g* (*Base x*) + *deriv* (*deriv f*) (*Base x*) ∗ *g* (*Base x*)
**proof** −
**have** *deriv* (*deriv* (*λz. f z ∗ g z*)) (*Base x*) = *deriv* (*λz. f z ∗ deriv g z* + *deriv f z ∗ g z*) (*Base x*)
**using** *assms* **by** (*simp add*: *eventually-deriv-mult deriv-cong-ev*)
**also have** ... = (*λz. f z ∗ deriv* (*deriv g*) *z* + *deriv f z ∗ deriv g z* + *deriv f z ∗ deriv g z* + *deriv* (*deriv f*) *z ∗ g z*) (*Base x*)
**by** (*simp add*: *assms deriv-field-differentiable-at field-differentiable-mult once-field-differentiable-at*)
**finally show** *?thesis*
**by** *simp*
**qed**
**then show**

*Eps12 x* ∗ *deriv* (λ*z*. *f z* ∗ *g z*) (*Base x*) + *Eps1 x* ∗ *Eps2 x* ∗ *deriv* (*deriv* (λ*z*. *f z* ∗ *g z*)) (*Base x*) =
*2* ∗ (*Eps1 x* ∗ (*Eps2 x* ∗ (*deriv f* (*Base x*) ∗ *deriv g* (*Base x*)))) +
*f* (*Base x*) ∗ (*Eps12 x* ∗ *deriv g* (*Base x*) + *Eps1 x* ∗ *Eps2 x* ∗ *deriv* (*deriv g*) (*Base x*)) +
(*Eps12 x* ∗ *deriv f* (*Base x*) + *Eps1 x* ∗ *Eps2 x* ∗ *deriv* (*deriv f*) (*Base x*)) ∗ *g* (*Base x*)
**using** *assms* **by** (*simp add*: *once-field-differentiable-at field-simps*)
**qed**

### 3.1.9 Sine and Cosine

The extended sin and cos at an arbitrary hyperdual.

**lemma** *hypext-sin-Hyperdual*:
(∗*h*∗ *sin*) (*Hyperdual a b c d*) = *sin a* $*_H$ *ba* + (*b* ∗*cos a*) $*_H$ *e1* + (*c* ∗ *cos a*) $*_H$ *e2* + (*d* ∗ *cos a* − *b* ∗ *c* ∗ *sin a*) $*_H$ *e12*
**by** (*simp add*: *hypext-Hyperdual-eq-parts*)

**lemma** *hypext-cos-Hyperdual*:
(∗*h*∗ *cos*) (*Hyperdual a b c d*) = *cos a* $*_H$ *ba* − (*b* ∗ *sin a*) $*_H$ *e1* − (*c* ∗ *sin a*) $*_H$ *e2* − (*d* ∗ *sin a* + *b* ∗ *c* ∗ *cos a*) $*_H$ *e12*
**proof** −
**have** *of-comp* (− (*d* ∗ *sin a*) − *b* ∗ *c* ∗ *cos a*) ∗ *e12* = − (*of-comp* (*d* ∗ *sin a* + *b* ∗ *c* ∗ *cos a*) ∗ *e12*)
**by** (*metis add-uminus-conv-diff minus-add-distrib mult-minus-left of-comp-minus*)
**then show** *?thesis*
**by** (*simp add*: *hypext-Hyperdual-eq-parts of-comp-minus scaleH-times*)
**qed**

**lemma** *Eps1-hypext-sin* [*simp*]:
*Eps1* ((∗*h*∗ *sin*) *x*) = *Eps1 x* ∗ *cos* (*Base x*)
**by** *simp*

**lemma** *Eps2-hypext-sin* [*simp*]:
*Eps2* ((∗*h*∗ *sin*) *x*) = *Eps2 x* ∗ *cos* (*Base x*)
**by** *simp*

**lemma** *Eps12-hypext-sin* [*simp*]:
*Eps12* ((∗*h*∗ *sin*) *x*) = *Eps12 x* ∗ *cos* (*Base x*) − *Eps1 x* ∗ *Eps2 x* ∗ *sin* (*Base x*)
**by** *simp*

**lemma** *hypext-sin-e1* [*simp*]:
(∗*h*∗ *sin*) (*x* ∗ *e1*) = *e1* ∗ *x*
**by** (*simp add*: *e1-def hyperdual-eq-iff one-hyperdual-def*)

**lemma** *hypext-sin-e2* [*simp*]:
(∗*h*∗ *sin*) (*x* ∗ *e2*) = *e2* ∗ *x*
**by** (*simp add*: *e2-def hyperdual-eq-iff one-hyperdual-def*)

**lemma** *hypext-sin-e12* [*simp*]:
$(*h* \; sin) \; (x * e12) = e12 * x$
**by** (*simp add*: *e12-def hyperdual-eq-iff one-hyperdual-def*)

**lemma** *hypext-cos-e1* [*simp*]:
$(*h* \; cos) \; (x * e1) = 1$
**by** (*simp add*: *e1-def hyperdual-eq-iff one-hyperdual-def*)

**lemma** *hypext-cos-e2* [*simp*]:
$(*h* \; cos) \; (x * e2) = 1$
**by** (*simp add*: *e2-def hyperdual-eq-iff one-hyperdual-def*)

**lemma** *hypext-cos-e12* [*simp*]:
$(*h* \; cos) \; (x * e12) = 1$
**by** (*simp add*: *e12-def hyperdual-eq-iff one-hyperdual-def*)

The extended sin and cos at $\beta \; x$.

**lemma** *hypext-sin-extract*:
$(*h* \; sin) \; (\beta \; x) = sin \; x *_H \; ba + cos \; x *_H \; e1 + cos \; x *_H \; e2 - sin \; x *_H \; e12$
**by** (*simp add*: *hypext-sin-Hyperdual of-comp-minus scaleH-times hyperdualx-def*)

**lemma** *hypext-cos-extract*:
$(*h* \; cos) \; (\beta \; x) = cos \; x *_H \; ba - sin \; x *_H \; e1 - sin \; x *_H \; e2 - cos \; x *_H \; e12$
**by** (*simp add*: *hypext-cos-Hyperdual hyperdualx-def*)

Extracting the extended sin components at $\beta \; x$.

**lemma** *Base-hypext-sin-extract* [*simp*]:
$Base \; ((*h* \; sin) \; (\beta \; x)) = sin \; x$
**by** (*rule Base-hypext*)

**lemma** *Eps2-hypext-sin-extract* [*simp*]:
$Eps2 \; ((*h* \; sin) \; (\beta \; x)) = cos \; x$
**using** *Eps2-hypext*[*of sin*] **by** *simp*

**lemma** *Eps12-hypext-sin-extract* [*simp*]:
$Eps12 \; ((*h* \; sin) \; (\beta \; x)) = - \; sin \; x$
**using** *Eps12-hypext*[*of sin*] **by** *simp*

Extracting the extended cos components at $\beta \; x$.

**lemma** *Base-hypext-cos-extract* [*simp*]:
$Base \; ((*h* \; cos) \; (\beta \; x)) = cos \; x$
**by** (*rule Base-hypext*)

**lemma** *Eps2-hypext-cos-extract* [*simp*]:
$Eps2 \; ((*h* \; cos) \; (\beta \; x)) = - \; sin \; x$
**using** *Eps2-hypext*[*of cos*] **by** *simp*

**lemma** *Eps12-hypext-cos-extract* [*simp*]:
$Eps12 \; ((*h* \; cos) \; (\beta \; x)) = - \; cos \; x$

**using** *Eps12-hypext*[*of cos*] **by** *simp*

We get one of the key trigonometric properties for the extensions of sin and cos.

**lemma** $((*h*\ sin)\ x)^2 + ((*h*\ cos)\ x)^2 = 1$
**by** (*simp add*: *hyperdual-eq-iff one-hyperdual-def power2-eq-square field-simps*)

**lemma** $(*h*\ sin)\ x + (*h*\ cos)\ x = (*h*\ (\lambda x.\ sin\ x + cos\ x))\ x$
**by** (*simp add*: *hypext-fun-add*)

### 3.1.10 Exponential

The exponential function extension behaves as expected.

**lemma** *hypext-exp-Hyperdual*:
$(*h*\ exp)\ (Hyperdual\ a\ b\ c\ d) =$
$exp\ a *_H ba + (b * exp\ a) *_H e1 + (c * exp\ a) *_H e2 + (d * exp\ a + b * c * exp\ a) *_H e12$
**by** (*simp add*: *hypext-Hyperdual-eq-parts*)

**lemma** *hypext-exp-extract*:
$(*h*\ exp)\ (\beta\ x) = exp\ x *_H ba + exp\ x *_H e1 + exp\ x *_H e2 + exp\ x *_H e12$
**by** (*simp add*: *hypext-extract-eq*)

**lemma** *hypext-exp-e1* [*simp*]:
$(*h*\ exp)\ (x * e1) = 1 + e1 * x$
**by** (*simp add*: *e1-def hyperdual-eq-iff*)

**lemma** *hypext-exp-e2* [*simp*]:
$(*h*\ exp)\ (x * e2) = 1 + e2 * x$
**by** (*simp add*: *e2-def hyperdual-eq-iff*)

**lemma** *hypext-exp-e12* [*simp*]:
$(*h*\ exp)\ (x * e12) = 1 + e12 * x$
**by** (*simp add*: *e12-def hyperdual-eq-iff*)

Extracting the parts for the exponential function extension.

**lemma** *Eps1-hypext-exp-extract* [*simp*]:
$Eps1\ ((*h*\ exp)\ (\beta\ x)) = exp\ x$
**using** *Eps1-hypext*[*of exp*] **by** *simp*

**lemma** *Eps2-hypext-exp-extract* [*simp*]:
$Eps2\ ((*h*\ exp)\ (\beta\ x)) = exp\ x$
**using** *Eps2-hypext*[*of exp*] **by** *simp*

**lemma** *Eps12-hypext-exp-extract* [*simp*]:
$Eps12\ ((*h*\ exp)\ (\beta\ x)) = exp\ x$
**using** *Eps12-hypext*[*of exp*] **by** *simp*

### 3.1.11 Square Root

Square root function extension.

**lemma** *hypext-sqrt-Hyperdual-Hyperdual*:
 **assumes** *a* > *0*
 **shows** (∗*h*∗ *sqrt*) (*Hyperdual a b c d*) =
  *Hyperdual* (*sqrt a*) (*b* ∗ *inverse* (*sqrt a*) / *2*) (*c* ∗ *inverse* (*sqrt a*) / *2*)
   (*d* ∗ *inverse* (*sqrt a*) / *2* − *b* ∗ *c* ∗ *inverse* (*sqrt a* ^ *3*) / *4*)
 **by** (*simp add*: *assms hypext-Hyperdual-eq*)

**lemma** *hypext-sqrt-Hyperdual*:
 *a* > *0* $\Longrightarrow$ (∗*h*∗ *sqrt*) (*Hyperdual a b c d*) =
  *sqrt a* $*_H$ *ba* + (*b* ∗ *inverse* (*sqrt a*) / *2*) $*_H$ *e1* + (*c* ∗ *inverse* (*sqrt a*) / *2*) $*_H$ *e2* +
  (*d* ∗ *inverse* (*sqrt a*) / *2* − *b* ∗ *c* ∗ *inverse* (*sqrt a* ^ *3*) / *4*) $*_H$ *e12*
 **by** (*auto simp add*: *hypext-Hyperdual-eq-parts*)

**lemma** *hypext-sqrt-extract*:
 *x* > *0* $\Longrightarrow$ (∗*h*∗ *sqrt*) (β *x*) = *sqrt x* $*_H$ *ba* + (*inverse* (*sqrt x*) / *2*) $*_H$ *e1* +
  (*inverse* (*sqrt x*) / *2*) $*_H$ *e2* − (*inverse* (*sqrt x* ^ *3*) / *4*) $*_H$ *e12*
 **by** (*simp add*: *hypext-sqrt-Hyperdual hyperdualx-def of-comp-minus scaleH-times*)

Extracting the parts for the square root extension.

**lemma** *Eps1-hypext-sqrt-extract* [*simp*]:
 *x* > *0* $\Longrightarrow$ *Eps1* ((∗*h*∗ *sqrt*) (β *x*)) = *inverse* (*sqrt x*) / *2*
 **using** *Eps1-hypext*[*of sqrt*] **by** *simp*

**lemma** *Eps2-hypext-sqrt-extract* [*simp*]:
 *x* > *0* $\Longrightarrow$ *Eps2* ((∗*h*∗ *sqrt*) (β *x*)) = *inverse* (*sqrt x*) / *2*
 **using** *Eps2-hypext*[*of sqrt*] **by** *simp*

**lemma** *Eps12-hypext-sqrt-extract* [*simp*]:
 *x* > *0* $\Longrightarrow$ *Eps12* ((∗*h*∗ *sqrt*) (β *x*)) = − (*inverse* (*sqrt x* ^ *3*) / *4*)
 **using** *Eps12-hypext*[*of sqrt*] **by** *simp*

**lemma** *Base x* > *0* $\Longrightarrow$ (∗*h*∗ *sin*) *x* + (∗*h*∗ *sqrt*) *x* = (∗*h*∗ (λ*x*. *sin x* + *sqrt x*)) *x*
 **by** (*simp add*: *hypext-fun-add*)

### 3.1.12 Natural Power

**lemma** *hypext-power*:
 (∗*h*∗ (λ*x*. *x* ^ *n*)) *x* = *x* ^ *n*
 **by** (*simp add*: *hyperdual-eq-iff hyperdual-power*)

**lemma** *hypext-fun-power*:
 **assumes** *f twice-field-differentiable-at* (*Base x*)
  **shows** (∗*h*∗ (λ*x*. (*f x*) ^ *n*)) *x* = ((∗*h*∗ *f*) *x*) ^ *n*
 **using** *assms hypext-compose*[*of f x* λ*x*. *x* ^ *n*] **by** (*simp add*: *hypext-power*)

**lemma** *hypext-power-Hyperdual*:
(*∗h∗* (*λx. x* ^ *n*)) (*Hyperdual a b c d*) =
*a* ^ *n* $*_H$ *ba* + (*of-nat n* ∗ *b* ∗ *a* ^ (*n* − *1*)) $*_H$ *e1* + (*of-nat n* ∗ *c* ∗ *a* ^ (*n* − *1*)) $*_H$ *e2* +
(*d* ∗ (*of-nat n* ∗ *a* ^ (*n* − *1*)) + *b* ∗ *c* ∗ (*of-nat n* ∗ *of-nat* (*n* − *1*) ∗ *a* ^ (*n* − *2*))) $*_H$ *e12*
**by** (*simp add*: *hypext-Hyperdual-eq-parts algebra-simps*)

**lemma** *hypext-power-Hyperdual-parts*:
(*∗h∗* (*λx. x* ^ *n*)) (*a* $*_H$ *ba* + *b* $*_H$ *e1* + *c* $*_H$ *e2* + *d* $*_H$ *e12*) =
*a* ^ *n* $*_H$ *ba* + (*of-nat n* ∗ *b* ∗ *a* ^ (*n* − *1*)) $*_H$ *e1* + (*of-nat n* ∗ *c* ∗ *a* ^ (*n* − *1*)) $*_H$ *e2* +
(*d* ∗ (*of-nat n* ∗ *a* ^ (*n* − *1*)) + *b* ∗ *c* ∗ (*of-nat n* ∗ *of-nat* (*n* − *1*) ∗ *a* ^ (*n* − *2*))) $*_H$ *e12*
**by** (*simp add*: *Hyperdual-eq* [*symmetric*] *hypext-power-Hyperdual*)

**lemma** *hypext-power-extract*:
(*∗h∗* (*λx. x* ^ *n*)) (*β x*) =
*x* ^ *n* $*_H$ *ba* + (*of-nat n* ∗ *x* ^ (*n* − *1*)) $*_H$ *e1* + (*of-nat n* ∗ *x* ^ (*n* − *1*)) $*_H$ *e2* +
(*of-nat n* ∗ *of-nat* (*n* − *1*) ∗ *x* ^ (*n* − *2*)) $*_H$ *e12*
**by** (*simp add*: *hypext-extract-eq*)

**lemma** *Eps1-hypext-power* [*simp*]:
*Eps1* ((*∗h∗* (*λx. x* ^ *n*)) *x*) = *of-nat n* ∗ *Eps1 x* ∗ (*Base x*) ^ (*n* − *1*)
**by** *simp*

**lemma** *Eps2-hypext-power* [*simp*]:
*Eps2* ((*∗h∗* (*λx. x* ^ *n*)) *x*) = *of-nat n* ∗ *Eps2 x* ∗ (*Base x*) ^ (*n* − *1*)
**by** *simp*

**lemma** *Eps12-hypext-power* [*simp*]:
*Eps12* ((*∗h∗* (*λx. x* ^ *n*)) *x*) =
*Eps12 x* ∗ (*of-nat n* ∗ *Base x* ^ (*n* − *1*)) + *Eps1 x* ∗ *Eps2 x* ∗ (*of-nat n* ∗ *of-nat* (*n* − *1*) ∗ *Base x* ^ (*n* − *2*))
**by** *simp*

### 3.1.13 Inverse

**lemma** *hypext-inverse*:
**assumes** *Base x* ≠ *0*
**shows** (*∗h∗ inverse*) *x* = *inverse x*
**using** *assms* **by** (*simp add*: *hyperdual-eq-iff inverse-eq-divide*)

**lemma** *hypext-fun-inverse*:
**assumes** *f twice-field-differentiable-at* (*Base x*)
**and** *f* (*Base x*) ≠ *0*
**shows** (*∗h∗* (*λx. inverse* (*f x*))) *x* = *inverse* ((*∗h∗ f*) *x*)

**using** *assms hypext-compose*[*of f x inverse*] **by** (*simp add*: *hypext-inverse*)

**lemma** *hypext-inverse-Hyperdual*:
$a \neq 0 \Longrightarrow$
$(*h*\ inverse)\ (Hyperdual\ a\ b\ c\ d) =$
$Hyperdual\ (inverse\ a)\ (-\ (b\ /\ a^2))\ (-\ (c\ /\ a^2))\ (2 * b * c\ /\ (a\ \widehat{}\ 3) - d\ /\ a^2)$
**by** (*simp add*: *hypext-Hyperdual-eq divide-inverse*)

**lemma** *hypext-inverse-Hyperdual-parts*:
$a \neq 0 \Longrightarrow$
$(*h*\ inverse)\ (a *_H ba + b *_H e1 + c *_H e2 + d *_H e12) =$
$inverse\ a *_H ba + -\ (b\ /\ a^2) *_H e1 + -\ (c\ /\ a^2) *_H e2 + (2 * b * c\ /\ a\ \widehat{}\ 3$
$-\ d\ /\ a^2) *_H e12$
**by** (*metis Hyperdual-eq hypext-inverse-Hyperdual*)

**lemma** *inverse-Hyperdual-parts*:
$(a::'a::real\text{-}normed\text{-}field) \neq 0 \Longrightarrow$
$inverse\ (a *_H ba + b *_H e1 + c *_H e2 + d *_H e12) =$
$inverse\ a *_H ba + -\ (b\ /\ a^2) *_H e1 + -\ (c\ /\ a^2) *_H e2 + (2 * b * c\ /\ a\ \widehat{}\ 3$
$-\ d\ /\ a^2) *_H e12$
**by** (*metis Hyperdual-eq hyperdual.sel*(*1*) *hypext-inverse hypext-inverse-Hyperdual-parts*)

**lemma** *hypext-inverse-extract*:
$x \neq 0 \Longrightarrow (*h*\ inverse)\ (\beta\ x) = inverse\ x *_H ba - (1\ /\ x^2) *_H e1 - (1\ /\ x^2)$
$*_H e2 + (2\ /\ x\ \widehat{}\ 3) *_H e12$
**by** (*simp add*: *hypext-extract-eq divide-inverse of-comp-minus scaleH-times*)

**lemma** *inverse-extract*:
$x \neq 0 \Longrightarrow inverse\ (\beta\ x) = inverse\ x *_H ba - (1\ /\ x^2) *_H e1 - (1\ /\ x^2) *_H e2$
$+ (2\ /\ x\ \widehat{}\ 3) *_H e12$
**by** (*metis hyperdual.sel*(*1*) *hyperdualx-def hypext-inverse hypext-inverse-extract*)

**lemma** *Eps1-hypext-inverse* [*simp*]:
$Base\ x \neq 0 \Longrightarrow Eps1\ ((*h*\ inverse)\ x) = -\ Eps1\ x * (1\ /\ (Base\ x)^2)$
**by** *simp*
**lemma** *Eps1-inverse* [*simp*]:
$Base\ (x::'a::real\text{-}normed\text{-}field\ hyperdual) \neq 0 \Longrightarrow Eps1\ (inverse\ x) = -\ Eps1\ x *$
$(1\ /\ (Base\ x)^2)$
**by** *simp*

**lemma** *Eps2-hypext-inverse* [*simp*]:
$Base\ (x::'a::real\text{-}normed\text{-}field\ hyperdual) \neq 0 \Longrightarrow Eps2\ (inverse\ x) = -\ Eps2\ x *$
$(1\ /\ (Base\ x)^2)$
**by** *simp*

**lemma** *Eps12-hypext-inverse* [*simp*]:
$Base\ (x::'a::real\text{-}normed\text{-}field\ hyperdual) \neq 0$
$\Longrightarrow Eps12\ (inverse\ x) = Eps1\ x * Eps2\ x * (2/\ (Base\ x\ \widehat{}\ 3)) - Eps12\ x\ /$
$(Base\ x)^2$

**by** *simp*

### 3.1.14 Division

**lemma** *hypext-fun-divide*:
**assumes** *f twice-field-differentiable-at* (*Base x*)
**and** *g twice-field-differentiable-at* (*Base x*)
**and** *g* (*Base x*) $\neq$ *0*
**shows** ($*h*$ ($\lambda x.$ *f x* / *g x*)) *x* = ($*h*$ *f*) *x* / ($*h*$ *g*) *x*
**proof** –
**have** ($\lambda x.$ *inverse* (*g x*)) *twice-field-differentiable-at Base x*
**by** (*simp add*: *assms*(*2*) *assms*(*3*) *twice-field-differentiable-at-compose*)
**moreover have** ($*h*$ *f*) *x* $*$ ($*h*$ ($\lambda x.$ *inverse* (*g x*))) *x* = ($*h*$ *f*) *x* $*$ *inverse* (($*h*$ *g*) *x*)
**by** (*simp add*: *assms*(*2*) *assms*(*3*) *hypext-fun-inverse*)
**ultimately have** ($*h*$ ($\lambda x.$ *f x* $*$ *inverse* (*g x*))) *x* = ($*h*$ *f*) *x* $*$ *inverse* (($*h*$ *g*) *x*)
**by** (*simp add*: *assms*(*1*) *hypext-fun-mult*)
**then show** *?thesis*
**by** (*simp add*: *divide-inverse hyp-divide-inverse*)
**qed**

### 3.1.15 Polynomial

**lemma** *hypext-polyn*:
**fixes** *coef* :: *nat* $\Rightarrow$ *'a* :: {*real-normed-field*}
**and** *n* :: *nat*
**shows** ($*h*$ ($\lambda x.$ $\sum$ *i*<*n*. *coef i* $*$ *x*^*i*)) *x* = ($\sum$ *i*<*n*. (*coef i*) $*_H$ (*x*^*i*))
**proof** (*induction n*)
**case** *0*
**then show** *?case*
**by** (*simp add*: *zero-hyperdual-def*)
**next**
**case** *hyp*: (*Suc n*)

**have** ($\lambda x.$ $\sum$ *i*<*Suc n*. *coef i* $*$ *x* ^ *i*) = ($\lambda x.$ ($\sum$ *i*<*n*. *coef i* $*$ *x* ^ *i*) + *coef n* $*$ *x* ^ *n*)
**and** ($\lambda x.$ $\sum$ *i*<*Suc n*. *coef i* $*_H$ *x* ^ *i*) = ($\lambda x.$ ($\sum$ *i*<*n*. *coef i* $*_H$ *x* ^ *i*) + *coef n* $*_H$ *x* ^ *n*)
**by** (*simp-all add*: *field-simps*)

**then show** *?case*
**proof** (*simp*)
**have** ($\lambda x.$ *coef n* $*$ *x* ^ *n*) *twice-field-differentiable-at Base x*
**using** *twice-field-differentiable-at-compose*[*of* $\lambda x.$ *x* ^ *n Base x* ($*$) (*coef n*)]
**by** *simp*
**then have** ($*h*$ ($\lambda x.$ ($\sum$ *i*<*n*. *coef i* $*$ *x* ^ *i*) + *coef n* $*$ *x* ^ *n*)) *x* =
($*h*$ ($\lambda x.$ ($\sum$ *i*<*n*. *coef i* $*$ *x* ^ *i*))) *x* + ($*h*$ ($\lambda x.$ *coef n* $*$ *x* ^ *n*)) *x*
**by** (*simp add*: *hypext-fun-add*)
**moreover have** ($*h*$ ($\lambda x.$ *coef n* $*$ *x* ^ *n*)) *x* = *coef n* $*_H$ *x* ^ *n*

**by** (*simp add*: *hypext-fun-scaleH hypext-power*)
**ultimately have** ($*h*$ ($\lambda x$. ($\sum i<n$. *coef* $i * x$ ^ $i$) + *coef* $n * x$ ^ $n$)) $x$ = ($\sum i<n$. *coef* $i *_H x$ ^ $i$) + *coef* $n *_H x$ ^ $n$
**using** *hyp* **by** *simp*
**then show** ($*h*$ ($\lambda x$. ($\sum i<n$. *coef* $i * x$ ^ $i$) + *coef* $n * x$ ^ $n$)) $x$ = ($\sum i<n$. *coef* $i *_H x$ ^ $i$) + *coef* $n *_H x$ ^ $n$
**by** *simp*
**qed**
**qed**

**lemma** *hypext-fun-polyn*:
**fixes** *coef* :: *nat* ⇒ *'a* :: {*real-normed-field*}
**and** *n* :: *nat*
**assumes** *f twice-field-differentiable-at* (*Base x*)
**shows** ($*h*$ ($\lambda x$. $\sum i<n$. *coef* $i *$ ($f$ $x$)^$i$)) $x$ = ($\sum i<n$. (*coef* $i$) $*_H$ ((($*h*$ $f$) $x$)^$i$))
**using** *assms hypext-compose*[*of f x* $\lambda x$. ($\sum i<n$. *coef* $i * x$^$i$)] **by** (*simp add*: *hypext-polyn*)

**end**

**theory** *LogisticFunction*
**imports** *HyperdualFunctionExtension*
**begin**

## 3.2 Logistic Function

Define the standard logistic function and its hyperdual variant:

**definition** *logistic* :: *real* ⇒ *real*
**where** *logistic x* = *inverse* (*1* + *exp* ($-x$))
**definition** *hyp-logistic* :: *real hyperdual* ⇒ *real hyperdual*
**where** *hyp-logistic x* = *inverse* (*1* + ($*h*$ *exp*) ($-x$))

Hyperdual extension of the logistic function is its hyperdual variant:

**lemma** *hypext-logistic*:
($*h*$ *logistic*) $x$ = *hyp-logistic x*
**proof** −
**have** ($*h*$ ($\lambda x$. *exp* ($-$ $x$) + *1*)) $x$ = ($*h*$ *exp*) ($-$ $x$) + *of-comp 1*
**by** (*simp add*: *hypext-compose hypext-uminus hypext-fun-cadd twice-field-differentiable-at-compose*)
**then have** ($*h*$ ($\lambda x$. *1* + *exp* ($-$ $x$))) $x$ = *1* + ($*h*$ *exp*) ($-$ $x$)
**by** (*simp add*: *one-hyperdual-def add.commute*)
**moreover have** *1* + *exp* ($-$ *Base x*) ≠ *0*
**by** (*metis exp-ge-zero add-eq-0-iff neg-0-le-iff-le not-one-le-zero*)
**moreover have** ($\lambda x$. *1* + *exp* ($-$ $x$)) *twice-field-differentiable-at Base x*
**proof** −
**have** ($\lambda x$. *exp* ($-$ $x$)) *twice-field-differentiable-at Base x*
**by** (*simp add*: *twice-field-differentiable-at-compose*)
**then have** ($\lambda x$. *exp* ($-$ $x$) + *1*) *twice-field-differentiable-at Base x*

**using** *twice-field-differentiable-at-compose*[*of* λ*x*. *exp* (− *x*) *Base x* λ*x*. *x* + *1*]
**by** *simp*
**then show** *?thesis*
**by** (*simp add*: *add.commute*)
**qed**
**ultimately have** (∗*h*∗ (λ*x*. *inverse* (*1* + *exp* (− *x*)))) *x* = *inverse* (*1* + (∗*h*∗ *exp*) (− *x*))
**by** (*simp add*: *hypext-fun-inverse*)
**then show** *?thesis*
**unfolding** *logistic-def hyp-logistic-def* **.**
**qed**

From properties of autodiff we know it gives us the derivative:

**lemma** *Eps1* (*hyp-logistic* (β *x*)) = *deriv logistic x*
**by** (*metis Eps1-hypext hypext-logistic*)

which is equal to the known derivative of the standard logistic function:

**lemma** *First* (*autodiff logistic x*) = *exp* (− *x*) / (*1* + *exp* (− *x*)) ^ *2*

**apply** (*simp only*: *autodiff.simps hyperdual-to-derivs.simps derivs.sel hypext-logistic*)

**apply** (*simp only*: *hyp-logistic-def inverse-hyperdual.code hyperdual.sel*)

**apply** (*simp add*: *hyperdualx-def hypext-exp-Hyperdual hyperdual-bases*)
**done**

Similarly we can get the second derivative:

**lemma** *Second* (*autodiff logistic x*) = *deriv* (*deriv logistic*) *x*
**by** (*rule autodiff-extract-second*)

and derive its value:

**lemma** *Second* (*autodiff logistic x*) = ((*exp* (− *x*) − *1*) ∗ *exp* (− *x*)) / ((*1* + *exp* (− *x*)) ^ *3*)

**apply** (*simp only*: *autodiff.simps hyperdual-to-derivs.simps derivs.sel hypext-logistic*)

**apply** (*simp only*: *hyp-logistic-def inverse-hyperdual.code hyperdual.sel*)

**apply** (*simp add*: *hyperdualx-def hypext-exp-Hyperdual hyperdual-bases*)

**proof** −
**have**
*2* ∗ (*exp* (− *x*) ∗ *exp* (− *x*)) / (*1* + *exp* (− *x*)) ^ *3* − *exp* (− *x*) / (*1* + *exp* (− *x*)) ^ *2* =
(*2* ∗ *exp* (− *x*) / (*1* + *exp* (− *x*)) ^ *3* − *1* / (*1* + *exp* (− *x*)) ^ *2*) ∗ *exp* (− *x*)
**by** (*simp add*: *field-simps*)
**also have** ... = (*2* ∗ *exp* (− *x*) / (*1* + *exp* (− *x*)) ^ *3* − (*1* + *exp* (− *x*)) / (*1* + *exp* (− *x*)) ^ *3*) ∗ *exp* (− *x*)

```
  proof −
    have inverse ((1 + exp (− x)) ^ 2) = inverse (1 + exp (− x)) ^ 2
      by (simp add: power-inverse)
    also have ... = (1 + exp (− x)) ∗ inverse (1 + exp (− x)) ∗ inverse (1 + exp
(− x)) ^ 2
      by (simp add: inverse-eq-divide)
    also have ... = (1 + exp (− x)) ∗ inverse (1 + exp (− x)) ^ 3
      by (simp add: power2-eq-square power3-eq-cube)
    finally have inverse ((1 + exp (− x)) ^ 2) = (1 + exp (− x)) ∗ inverse ((1 +
exp (− x)) ^ 3)
      by (simp add: power-inverse)
    then show ?thesis
      by (simp add: inverse-eq-divide)
  qed
  also have ... = (2 ∗ exp (− x) − (1 + exp (− x))) / (1 + exp (− x)) ^ 3 ∗ exp
(− x)
    by (metis diff-divide-distrib)
  finally show
    2 ∗ (exp (− x) ∗ exp (− x)) / (1 + exp (− x)) ^ 3 − exp (− x) / (1 + exp (−
x))² =
     (exp (− x) − 1) ∗ exp (− x) / (1 + exp (− x)) ^ 3
    by (simp add: field-simps)
qed

end
```

```
theory AnalyticTestFunction
  imports HyperdualFunctionExtension HOL−Decision-Procs.Approximation
begin
```

### 3.3 Analytic Test Function

We investigate the analytic test function used by Fike and Alonso in their 2011 paper [1] as a relatively non-trivial example. The function is defined as: $\lambda x.\ exp\ x\ /\ sqrt\ ((sin\ x)^3 + (cos\ x)^3)$.

```
definition fa-test :: real ⇒ real
  where fa-test x = exp x / (sqrt (sin x ^ 3 + cos x ^ 3))
```

We define the same composition of functions but using the relevant hyperdual versions. Note that we implicitly use the facts that hyperdual extensions of plus, times and inverse are the same operations on hyperduals.

```
definition hyp-fa-test :: real hyperdual ⇒ real hyperdual
  where hyp-fa-test x = ((∗h∗ exp) x) / ((∗h∗ sqrt) (((∗h∗ sin) x) ^ 3 + ((∗h∗
cos) x) ^ 3))
```

We prove lemmas useful to show when this function is well defined.

```
lemma sin-cube-plus-cos-cube:
```

```
  sin x ^ 3 + cos x ^ 3 = 1/2 * (sin x + cos x) * (2 − sin (2 * x))
  for x :: 'a::{real-normed-field,banach}
proof −
  have sin x ^ 3 + cos x ^ 3 = (sin x + cos x) * (cos x ^ 2 − cos x * sin x + sin
x ^ 2)
    by (smt (z3) add.commute combine-common-factor diff-add-cancel distrib-left
        mult.commute mult.left-commute power2-eq-square power3-eq-cube)
  also have ... = (sin x + cos x) * (1 − cos x * sin x)
    by simp
  finally show ?thesis
   by (smt (z3) mult.commute mult.left-neutral mult-2 nonzero-mult-div-cancel-left
        right-diff-distrib' sin-add times-divide-eq-left zero-neq-numeral)
qed

lemma sin-cube-plus-cos-cube-gt-zero-iff:
  (sin x ^ 3 + cos x ^ 3 > 0) = (sin x + cos x > 0)
  for x ::real
 by (smt (verit, best) cos-zero power3-eq-cube power-zero-numeral sin-cube-plus-cos-cube
      sin-le-one sin-zero zero-less-mult-iff)

lemma sin-plus-cos-eq-45:
  sin x + cos x = sqrt 2 * sin (x + pi/4)
  apply (simp add: sin-add sin-45 cos-45 )
  by (simp add: field-simps)

lemma sin-cube-plus-cos-cube-gt-zero-iff':
  (sin x ^ 3 + cos x ^ 3 > 0) = (sin (x + pi/4) > 0)
  by (smt (verit, best) mult-pos-pos real-sqrt-gt-0-iff
      sin-cube-plus-cos-cube-gt-zero-iff sin-plus-cos-eq-45 zero-less-mult-pos)

lemma sin-less-zero-pi:
  ⟦− pi < x; x < 0⟧ ⟹ sin x < 0
 by (metis add.inverse-inverse add.inverse-neutral neg-less-iff-less sin-gt-zero sin-minus)

lemma sin-45-positive-intervals:
 (sin (x + pi/4) > 0) = (x ∈ (⋃ n::int. {−pi/4 + 2*pi*n <..< 3*pi/4 + 2*pi*n}))
proof (standard ; (elim UnionE rangeE | −))
  obtain y :: real and n :: int
    where x = y + 2*pi*n and − pi ≤ y and y ≤ pi
    using sincos-principal-value sin-cos-eq-iff less-eq-real-def by metis
  note yn = this

  assume 0 < sin (x + pi / 4)
  then have a: 0 < sin (y + pi / 4)
    using yn by (metis sin-add sin-cos-eq-iff)
  then have y ∈ {− pi / 4<..<3 * pi / 4}
  proof (unfold greaterThanLessThan-iff, safe)
    show − pi / 4 < y
    proof (rule ccontr)
```

```
      assume ¬ − pi / 4 < y
      then have y < − pi / 4 ∨ y = − pi / 4
        by (simp add: not-less le-less)
      then show False
        using a sin-less-zero-pi[where x = y + pi/4] yn sin-zero
        by force
    qed
    show y < 3 ∗ pi / 4
    proof (rule ccontr)
      assume ¬ y < 3 ∗ pi / 4
      then have pi ≤ y + pi/4
        by (simp add: not-less)
      then show False
        using a sin-le-zero yn pi-ge-two by fastforce
    qed
  qed
  then have x ∈ {− pi / 4 + 2∗pi∗n<..<3 ∗ pi / 4 + 2∗pi∗n}
    using yn greaterThanLessThan-iff by simp
  then show x ∈ (⋃n::int. {− pi / 4 + 2∗pi∗n<..<3 ∗ pi / 4 + 2∗pi∗n})
    by blast
next
  fix X and n :: int
  assume x ∈ X
     and X = {− pi / 4 + 2∗pi∗n<..<3 ∗ pi / 4 + 2∗pi∗n}
  then have x ∈ {− pi / 4 + 2∗pi∗n<..<3 ∗ pi / 4 + 2∗pi∗n}
    by simp
  then obtain y :: real and n :: int
    where x = y + 2∗pi∗n and − pi / 4 < y and y < 3 ∗ pi / 4
    by (smt (z3) greaterThanLessThan-iff)
  note yn = this

  have 0 < sin (y + pi / 4)
    using sin-gt-zero yn by force
  then show 0 < sin (x + pi / 4)
    using yn sin-cos-eq-iff[of x + pi / 4 y + pi / 4] by simp
qed
```

When the function is well defined our hyperdual definition is equal to the hyperdual extension.

```
lemma hypext-fa-test:
  assumes Base x ∈ (⋃n::int. {−pi/4 + 2∗pi∗n <..< 3∗pi/4 + 2∗pi∗n})
  shows (∗h∗ fa-test) x = hyp-fa-test x
proof −
  have inverse-sqrt-valid: 0 < sin (Base x) ^ 3 + cos (Base x) ^ 3
   using assms sin-45-positive-intervals sin-cube-plus-cos-cube-gt-zero-iff' by force

  have ⋀f. (λx. (sin x) ^ 3) twice-field-differentiable-at Base x
   and ⋀f. (λx. (cos x) ^ 3) twice-field-differentiable-at Base x
   by (simp-all add: twice-field-differentiable-at-compose[OF - twice-field-differentiable-at-power])
```

**then have** (*h* (λx. sin x ^ 3 + cos x ^ 3)) x = (*h* sin) x ^ 3 + (*h* cos) x ^ 3
**and** d-sincos: (λx. sin x ^ 3 + cos x ^ 3) twice-field-differentiable-at Base x
**using** hypext-fun-add[of λx. sin x ^ 3 x λx. cos x ^ 3]
**by** (simp-all add: hypext-fun-power twice-field-differentiable-at-add)
**then have** (*h* (λx. sqrt (sin x ^ 3 + cos x ^ 3))) x = (*h* sqrt) ((*h* sin) x ^ 3 + (*h* cos) x ^ 3)
**using** inverse-sqrt-valid hypext-compose[of λx. sin x ^ 3 + cos x ^ 3 x] **by** simp
**moreover have** d-sqrt: (λx. sqrt (sin x ^ 3 + cos x ^ 3)) twice-field-differentiable-at Base x
**using** inverse-sqrt-valid d-sincos twice-field-differentiable-at-compose twice-field-differentiable-at-sqrt
**by** blast
**ultimately have** (*h* (λx. inverse (sqrt (sin x ^ 3 + cos x ^ 3)))) x = inverse ((*h* sqrt) ((*h* sin) x ^ 3 + (*h* cos) x ^ 3))
**using** inverse-sqrt-valid hypext-fun-inverse[of λx. sqrt (sin x ^ 3 + cos x ^ 3) x]
**by** simp
**moreover have** (λx. inverse (sqrt (sin x ^ 3 + cos x ^ 3))) twice-field-differentiable-at Base x
**using** inverse-sqrt-valid d-sqrt real-sqrt-eq-zero-cancel-iff
twice-field-differentiable-at-compose twice-field-differentiable-at-inverse
less-numeral-extra(3)
**by** force
**ultimately have**
(*h* (λx. exp x * inverse (sqrt (sin x ^ 3 + cos x ^ 3)))) x =
(*h* exp) x * inverse ((*h* sqrt) ((*h* sin) x ^ 3 + (*h* cos) x ^ 3))
**by** (simp add: hypext-fun-mult)
**then have**
(*h* (λx. exp x / sqrt (sin x ^ 3 + cos x ^ 3))) x =
(*h* exp) x / (*h* sqrt) ((*h* sin) x ^ 3 + (*h* cos) x ^ 3)
**by** (simp add: inverse-eq-divide hyp-divide-inverse)
**then show** ?thesis
**unfolding** fa-test-def hyp-fa-test-def **.**
**qed**

We can show that our hyperdual extension gives (approximately) the same values as those found by Fike and Alonso when evaluated at (15::'a) / (10::'a).

**lemma**
**assumes** x = hyp-fa-test (β 1.5)
**shows** |Base x − 4.4978| ≤ 0.00005
**and** |Eps1 x − 4.0534| ≤ 0.00005
**and** |Eps12 x − 9.4631| ≤ 0.00005
**proof** −
**show** |Base x − 4.4978| ≤ 0.00005
**by** (simp add: assms hyp-fa-test-def, approximation 20)
**have** d: 0 < sin (3/2 :: real) ^ 3 + cos (3/2 :: real) ^ 3
**using** sin-cube-plus-cos-cube-gt-zero-iff' sin-gt-zero pos-add-strict pi-gt3 **by** force
**show** |Eps1 x − 4.0534| ≤ 0.00005

**by** (*simp add*: *assms hyp-fa-test-def d*, *approximation 22*)
**show** $|Eps12\ x - 9.4631| \leq 0.00005$
**by** (*simp add*: *assms hyp-fa-test-def d*, *approximation 24*)
**qed**

A number of additional lemmas that will be required to prove the derivatives:

**lemma** *hypext-sqrt-hyperdual-parts*:
$a > 0 \Longrightarrow (*h*\ sqrt)\ (a *_H ba + b *_H e1 + c *_H e2 + d *_H e12) =$
$sqrt\ a *_H ba + (b * inverse\ (sqrt\ a)\ /\ 2) *_H e1 + (c * inverse\ (sqrt\ a)\ /\ 2)$
$*_H e2 +$
$(d * inverse\ (sqrt\ a)\ /\ 2 - b * c * inverse\ (sqrt\ a\ \hat{}\ 3)\ /\ 4) *_H e12$
**by** (*metis Hyperdual-eq hypext-sqrt-Hyperdual*)

**lemma** *cos-multiple*: $cos\ (n * x) = 2 * cos\ x * cos\ ((n - 1) * x) - cos\ ((n - 2)$
$* x)$
**for** $x :: 'a :: \{banach,real\text{-}normed\text{-}field\}$
**proof** −
**have** $cos\ ((n - 1) * x + x) + cos\ ((n - 1) * x - x) = 2 * cos\ ((n - 1) * x) *$
$cos\ x$
**by** (*simp add*: *cos-add cos-diff*)
**then show** *?thesis*
**by** (*simp add*: *left-diff-distrib′ eq-diff-eq*)
**qed**

**lemma** *sin-multiple*: $sin\ (n * x) = 2 * cos\ x * sin\ ((n - 1) * x) - sin\ ((n - 2) *$
$x)$
**for** $x :: 'a :: \{banach,real\text{-}normed\text{-}field\}$
**proof** −
**have** $sin\ ((n - 1) * x + x) + sin\ ((n - 1) * x - x) = 2 * cos\ x * sin\ ((n - 1)$
$* x)$
**by** (*simp add*: *sin-add sin-diff*)
**then show** *?thesis*
**by** (*simp add*: *left-diff-distrib′ eq-diff-eq*)
**qed**

**lemma** *power5*:
**fixes** $z :: 'a :: monoid\text{-}mult$
**shows** $z\ \hat{}\ 5 = z * z * z * z * z$
**by** (*simp add*: *mult.assoc power2-eq-square power-numeral-odd*)

**lemma** *power6*:
**fixes** $z :: 'a :: monoid\text{-}mult$
**shows** $z\ \hat{}\ 6 = z * z * z * z * z * z$
**by** (*simp add*: *mult.assoc power3-eq-cube power-numeral-even*)

We find the derivatives of *fa-test* by applying a Wengert list approach, as done by Fike and Alonso, to make the same composition but in hyperduals. We know that this is equal to the hyperdual extension which in turn gives us the derivatives.

**lemma** *Wengert-autodiff-fa-test*:
**assumes** *x* ∈ (⋃*n*::*int*. {−*pi*/*4* + *2*∗*pi*∗*n* <..< *3*∗*pi*/*4* + *2*∗*pi*∗*n*})
**shows** *First* (*autodiff fa-test x*) =
(*exp x* ∗ (*3* ∗ *cos x* + *5* ∗ *cos* (*3* ∗ *x*) + *9* ∗ *sin x* + *sin* (*3* ∗ *x*))) /
(*8* ∗ (*sqrt* (*sin x* ^ *3* + *cos x* ^ *3*)) ^ *3*)
**and** *Second* (*autodiff fa-test x*) =
(*exp x* ∗ (*130* − *12* ∗ *cos* (*2* ∗ *x*) +
*30* ∗ *cos* (*4* ∗ *x*) + *12* ∗ *cos* (*6* ∗ *x*) −
*111* ∗ *sin* (*2* ∗ *x*) +
*48* ∗ *sin* (*4* ∗ *x*) + *5* ∗ *sin* (*6* ∗ *x*))) /
(*64* ∗ (*sqrt* (*sin x* ^ *3* + *cos x* ^ *3*)) ^ *5*)
**proof** −
**have** *s3-c3-gt-zero*: (*sin x*) ^ *3* + (*cos x*) ^ *3* > *0*
**using** *assms sin-45-positive-intervals sin-cube-plus-cos-cube-gt-zero-iff′ sin-gt-zero*
**by** *force*
— Work out *hyp-fa-test* as Wengert list of basic operations
**let** *?w0* = *β x*
**have** *w0*: *?w0* = *x* ∗$_H$ *ba* + *1* ∗$_H$ *e1* + *1* ∗$_H$ *e2* + *0* ∗$_H$ *e12*
**by** (*simp add*: *Hyperdual-eq hyperdualx-def*)

**let** *?w1* = (∗*h*∗ *exp*) *?w0*
**have** *w1*: *?w1* = *exp x* ∗$_H$ *ba* + *exp x* ∗$_H$ *e1* + *exp x* ∗$_H$ *e2* + *exp x* ∗$_H$ *e12*
**using** *hypext-exp-extract* **by** *blast*

**let** *?w2* = (∗*h*∗ *sin*) *?w0*
**have** *w2*: *?w2* = *sin x* ∗$_H$ *ba* + *cos x* ∗$_H$ *e1* + *cos x* ∗$_H$ *e2* + − *sin x* ∗$_H$ *e12*
**by** (*simp add*: *hypext-sin-extract of-comp-minus scaleH-times*)

**let** *?w3* = (∗*h*∗ (λ*x*. *x* ^ *3*)) *?w2*
**have** *w3*: *?w3* = (*sin x*) ^ *3* ∗$_H$ *ba* + (*3* ∗ *cos x* ∗ (*sin x*)$^2$) ∗$_H$ *e1* + (*3* ∗ *cos x* ∗ (*sin x*)$^2$) ∗$_H$ *e2* + − (*3*/*4* ∗ (*sin x* − *3* ∗ *sin* (*3* ∗ *x*))) ∗$_H$ *e12*
**by** (*simp add*: *w2 hypext-power-Hyperdual-parts power2-eq-square cos-times-cos sin-times-sin*
*sin-times-cos distrib-left right-diff-distrib′ divide-simps*)

**let** *?w4* = (∗*h*∗ *cos*) *?w0*
**have** *w4*: *?w4* = *cos x* ∗$_H$ *ba* + − *sin x* ∗$_H$ *e1* + − *sin x* ∗$_H$ *e2* + − *cos x* ∗$_H$ *e12*
**by** (*simp add*: *hypext-cos-extract of-comp-minus scaleH-times*)

**let** *?w5* = (∗*h*∗ (λ*x*. *x* ^ *3*)) *?w4*
**have** *w5*: *?w5* = (*cos x*) ^ *3* ∗$_H$ *ba* + − (*3* ∗ *sin x* ∗ (*cos x*)$^2$) ∗$_H$ *e1* + − (*3* ∗ *sin x* ∗ (*cos x*)$^2$) ∗$_H$ *e2* + − (*3*/*4* ∗ (*cos x* + *3* ∗ *cos* (*3* ∗ *x*))) ∗$_H$ *e12*
**by** (*simp add*: *w4 hypext-power-Hyperdual-parts sin-times-sin right-diff-distrib′ cos-times-cos*
*power2-eq-square distrib-left sin-times-cos divide-simps*)

**let** *?w6* = *?w3* + *?w5*
**have** *sqrt-pos*: *Base ?w6* > *0*

**using** *s3-c3-gt-zero* **by** *auto*
**have** *w6*: *?w6 = (sin x ^ 3 + cos x ^ 3)* $*_H$ *ba +*
*(3 * cos x * sin x * (sin x − cos x))* $*_H$ *e1 +*
*(3 * cos x * sin x * (sin x − cos x))* $*_H$ *e2 +*
*− (3/4 * (sin x + cos x + 3 * cos (3 * x) − 3 * sin (3 * x)))* $*_H$ *e12*
**by** (*auto simp add*: *w3 w5 add-hyperdual-parts power2-eq-square right-diff-distrib′ divide-simps*)

**let** *?w7 = inverse ((*h* sqrt) ?w6)*
**have** *w7*: *?w7 = inverse(sqrt(sin x ^ 3 + cos x ^ 3))* $*_H$ *ba +*
*− ((3 * cos x * sin x * (sin x − cos x))/(2 * (sqrt (sin x ^ 3 + cos x ^ 3)) ^ 3))* $*_H$ *e1 +*
*− ((3 * cos x * sin x * (sin x − cos x))/(2 * (sqrt (sin x ^ 3 + cos x ^ 3)) ^ 3))* $*_H$ *e2 +*
*((3 * (30 + 2 * cos (4 * x) − 41 * sin (2 * x) + 3 * sin (6 * x)))/(64 * (sqrt (sin x ^ 3 + cos x ^ 3)) ^ 5))* $*_H$ *e12*
— Apply the functions in two steps, then simplify the result to match
**proof** −
**let** *?w7a = (*h* sqrt) ?w6*
**have** *w7a*: *?w7a =*
*sqrt (sin x ^ 3 + cos x ^ 3)* $*_H$ *ba +*
*((3 * (cos x * (sin x * (sin x − cos x)))) * inverse (sqrt (sin x ^ 3 + cos x ^ 3)) / 2)* $*_H$ *e1 +*
*((3 * (cos x * (sin x * (sin x − cos x)))) * inverse (sqrt (sin x ^ 3 + cos x ^ 3)) / 2)* $*_H$ *e2 +*
*(− ((3 * sin x + 3 * cos x + 9 * cos (3 * x) − 9 * sin (3 * x)) * inverse (sqrt (sin x ^ 3 + cos x ^ 3)) / 8) +*
*− 9 * (cos x * (sin x * ((sin x − cos x) * (cos x * (sin x * (sin x − cos x)))))) * inverse (sqrt (sin x ^ 3 + cos x ^ 3) ^ 3) / 4)* $*_H$ *e12*
**unfolding** *w6*
**using** *sqrt-pos* **by** (*simp add*: *hypext-sqrt-hyperdual-parts mult.assoc*)

**let** *?w7b = inverse ?w7a*
**have** *?w7b =*
*(1 / sqrt (sin x ^ 3 + cos x ^ 3))* $*_H$ *ba +*
*− (3 * (cos x * (sin x * (sin x − cos x))) * inverse (sqrt (sin x ^ 3 + cos x ^ 3)) / (2 * (sqrt (sin x ^ 3 + cos x ^ 3))*$^2$*))* $*_H$ *e1 +*
*− (3 * (cos x * (sin x * (sin x − cos x))) * inverse (sqrt (sin x ^ 3 + cos x ^ 3)) / (2 * (sqrt (sin x ^ 3 + cos x ^ 3))*$^2$*))* $*_H$ *e2 +*
*(9 * (cos x * (cos x * (sin x * (sin x * ((sin x − cos x) * ((sin x − cos x) * (inverse (sqrt (sin x ^ 3 + cos x ^ 3)) * inverse (sqrt (sin x ^ 3 + cos x ^ 3))))))))) / (2 * sqrt (sin x ^ 3 + cos x ^ 3) ^ 3) −*
*(− ((3 * sin x + 3 * cos x + 9 * cos (3 * x) − 9 * sin (3 * x)) * inverse (sqrt (sin x ^ 3 + cos x ^ 3)) / 8) −*
*9 * (cos x * (sin x * ((sin x − cos x) * (cos x * (sin x * (sin x − cos x)))))) * inverse (sqrt (sin x ^ 3 + cos x ^ 3) ^ 3) / 4) /*
*(sqrt (sin x ^ 3 + cos x ^ 3))*$^2$*)* $*_H$ *e12*
— Push the inverse in
**by** (*simp add*: *w7a inverse-hyperdual-parts*)

**then have** *w7b*: *?w7b* =
(*inverse* (*sqrt* (*sin x* ^ *3* + *cos x* ^ *3*))) $*_H$ *ba* +
− (*3* * *cos x* * *sin x* * (*sin x* − *cos x*) / (*2* * (*sqrt* (*sin x* ^ *3* + *cos x* ^ *3*)) ^ *3*)) $*_H$ *e1* +
− (*3* * *cos x* * *sin x* * (*sin x* − *cos x*) / (*2* * (*sqrt* (*sin x* ^ *3* + *cos x* ^ *3*)) ^ *3*)) $*_H$ *e2* +
(*9* * *cos x* * *cos x* * *sin x* * *sin x* * ((*sin x* − *cos x*) * (*sin x* − *cos x*) / ((*sqrt* (*sin x* ^ *3* + *cos x* ^ *3*)) ^ *2*)) / (*2* * *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*) ^ *3*) −
(− ((*3* * *sin x* + *3* * *cos x* + *9* * *cos* (*3* * *x*) − *9* * *sin* (*3* * *x*)) / (*8* * *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*))) −
*9* * *cos x* * *sin x* * (*sin x* − *cos x*) * *cos x* * *sin x* * (*sin x* − *cos x*) / (*4* * *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*) ^ *3*))
/ (*sqrt* (*sin x* ^ *3* + *cos x* ^ *3*))$^2$) $*_H$ *e12*
— Simplify powers and parents
**by** (*simp add*: *power2-eq-square power3-eq-cube field-simps*)

**have**
*9* * *cos x* * *cos x* * *sin x* * *sin x* * ((*sin x* − *cos x*) * (*sin x* − *cos x*)) / ((*sqrt* (*sin x* ^ *3* + *cos x* ^ *3*))$^2$ * (*2* * *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*) ^ *3*)) −
(− ((*3* * *sin x* + *3* * *cos x* + *9* * *cos* (*3* * *x*) − *9* * *sin* (*3* * *x*)) / (*8* * *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*))) −
*9* * *cos x* * *sin x* * (*sin x* − *cos x*) * *cos x* * *sin x* * (*sin x* − *cos x*) / (*4* * *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*) ^ *3*)) /
(*sqrt* (*sin x* ^ *3* + *cos x* ^ *3*))$^2$ =
(*90* + *6* * *cos* (*4* * *x*) − *123* * *sin* (*2* * *x*) + *9* * *sin* (*6* * *x*)) / (*64* * *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*) ^ *5*)
— Equate the last component's coefficients
**proof** −
**have**
*9* * *cos x* * *cos x* * *sin x* * *sin x* * ((*sin x* − *cos x*) * (*sin x* − *cos x*)) / ((*sqrt* (*sin x* ^ *3* + *cos x* ^ *3*))$^2$ * (*2* * *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*) ^ *3*)) −
(− ((*3* * *sin x* + *3* * *cos x* + *9* * *cos* (*3* * *x*) − *9* * *sin* (*3* * *x*)) / (*8* * *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*))) −
*9* * *cos x* * *sin x* * (*sin x* − *cos x*) * *cos x* * *sin x* * (*sin x* − *cos x*) / (*4* * *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*) ^ *3*)) /
(*sqrt* (*sin x* ^ *3* + *cos x* ^ *3*))$^2$ =
*9* * *cos x* * *cos x* * *sin x* * *sin x* * (*sin x* − *cos x*) * (*sin x* − *cos x*) / (*2* * *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*) ^ *5*) +
( (*3* * *sin x* + *3* * *cos x* + *9* * *cos* (*3* * *x*) − *9* * *sin* (*3* * *x*)) / (*8* * *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*)) +
*9* * *cos x* * *sin x* * (*sin x* − *cos x*) * *cos x* * *sin x* * (*sin x* − *cos x*) / (*4* * *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*) ^ *3*)) /
(*sqrt* (*sin x* ^ *3* + *cos x* ^ *3*)) ^ *2*
**by** (*simp add*: *divide-simps*)
**also have** ... =
*9* * *cos x* * *cos x* * *sin x* * *sin x* * (*sin x* − *cos x*) * (*sin x* − *cos x*) / (*2* * *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*) ^ *5*) +
( (*3* * *sin x* + *3* * *cos x* + *9* * *cos* (*3* * *x*) − *9* * *sin* (*3* * *x*)) / (*8* * (*sqrt* (*sin x* ^ *3* + *cos x* ^ *3*)) ^ *3*) +

*9 ∗ cos x ∗ sin x ∗ (sin x − cos x) ∗ cos x ∗ sin x ∗ (sin x − cos x) / (4 ∗ sqrt (sin x ^ 3 + cos x ^ 3) ^ 3 ∗ (sqrt (sin x ^ 3 + cos x ^ 3)) ^ 2))*
**by** (*simp add*: *add-divide-distrib power2-eq-square power3-eq-cube mult.commute mult.left-commute*)
**also have** ... =
*9 ∗ cos x ∗ cos x ∗ sin x ∗ sin x ∗ (sin x − cos x) ∗ (sin x − cos x) / (2 ∗ sqrt (sin x ^ 3 + cos x ^ 3) ^ 5) +*
*( (3 ∗ sin x + 3 ∗ cos x + 9 ∗ cos (3 ∗ x) − 9 ∗ sin (3 ∗ x)) / (8 ∗ (sqrt (sin x ^ 3 + cos x ^ 3)) ^ 3) +*
*9 ∗ cos x ∗ sin x ∗ (sin x − cos x) ∗ cos x ∗ sin x ∗ (sin x − cos x) / (4 ∗ sqrt (sin x ^ 3 + cos x ^ 3) ^ 5))*
**by** (*simp add*: *mult.commute mult.left-commute*)
**also have** ... =
*9 ∗ cos x ∗ cos x ∗ sin x ∗ sin x ∗ (sin x − cos x) ∗ (sin x − cos x) / (2 ∗ sqrt (sin x ^ 3 + cos x ^ 3) ^ 5) +*
*( (3 ∗ sin x + 3 ∗ cos x + 9 ∗ cos (3 ∗ x) − 9 ∗ sin (3 ∗ x)) ∗ (sin x ^ 3 + cos x ^ 3) / (8 ∗ (sqrt (sin x ^ 3 + cos x ^ 3)) ^ 5) +*
*9 ∗ cos x ∗ sin x ∗ (sin x − cos x) ∗ cos x ∗ sin x ∗ (sin x − cos x) / (4 ∗ sqrt (sin x ^ 3 + cos x ^ 3) ^ 5))*
**proof** −
**have** (*sin x ^ 3 + cos x ^ 3*) ∗ *inverse* ((*sqrt* (*sin x ^ 3 + cos x ^ 3*)) ^ *2*) = *1*
**using** *s3-c3-gt-zero* **by** *auto*
**then have**
*1 / (8 ∗ (sqrt (sin x ^ 3 + cos x ^ 3)) ^ 3) =*
*(sin x ^ 3 + cos x ^ 3) / ((sqrt (sin x ^ 3 + cos x ^ 3)) ^ 2) / (8 ∗ (sqrt (sin x ^ 3 + cos x ^ 3)) ^ 3)*
**by** (*simp add*: *field-simps*)
**then have**
*1 / (8 ∗ (sqrt (sin x ^ 3 + cos x ^ 3)) ^ 3) =*
*(sin x ^ 3 + cos x ^ 3) / (8 ∗ (sqrt (sin x ^ 3 + cos x ^ 3)) ^ 5)*
**by** *auto*
**then show** *?thesis*
**by** (*metis* (*mono-tags*, *lifting*) *divide-real-def inverse-eq-divide times-divide-eq-right*)
**qed**
**also have** ... =
*( 288 ∗ cos x ∗ cos x ∗ sin x ∗ sin x ∗ (sin x − cos x) ∗ (sin x − cos x) +*
*(3 ∗ sin x + 3 ∗ cos x + 9 ∗ cos (3 ∗ x) − 9 ∗ sin (3 ∗ x)) ∗ (sin x ^ 3 + cos x ^ 3) ∗ 8 +*
*144 ∗ cos x ∗ sin x ∗ (sin x − cos x) ∗ cos x ∗ sin x ∗ (sin x − cos x))*
*/ (64 ∗ (sqrt (sin x ^ 3 + cos x ^ 3)) ^ 5)*
**by** (*simp add*: *divide-simps*)
**also have** ... =
*( 432 ∗ cos x ∗ cos x ∗ sin x ∗ sin x ∗ (sin x − cos x) ∗ (sin x − cos x) +*
*(24 ∗ sin x + 24 ∗ cos x + 72 ∗ cos (3 ∗ x) − 72 ∗ sin (3 ∗ x)) ∗ (sin x ^ 3 + cos x ^ 3))*
*/ (64 ∗ (sqrt (sin x ^ 3 + cos x ^ 3)) ^ 5)*
**by** (*simp add*: *divide-simps*)
**finally show** *?thesis*

— Having matched the denominators, prove the numerators equal
**by** (*simp*, *force intro*: *disjI2 simp add*: *power2-eq-square power4-eq-xxxx power3-eq-cube cos-times-sin sin-times-cos cos-times-cos sin-times-sin right-diff-distrib power6 power5 distrib-left distrib-right left-diff-distrib divide-simps*)
**qed**
**then show** *?thesis*
— Put it all together
**by** (*simp add*: *w7b*)
**qed**

**let** *?w8* = *?w1* ∗ *?w7*
**have** *w8*: *?w8* =
((*exp x*)/(*sqrt*(*sin x* ^ *3* + *cos x* ^ *3*))) $*_H$ *ba* +
((*exp x* ∗ (*3* ∗ *cos x* + *5* ∗ *cos* (*3* ∗ *x*) + *9* ∗ *sin x* + *sin* (*3* ∗ *x*)))/(*8* ∗ (*sqrt* (*sin x* ^ *3* + *cos x* ^ *3*)) ^ *3*)) $*_H$ *e1* +
((*exp x* ∗ (*3* ∗ *cos x* + *5* ∗ *cos* (*3* ∗ *x*) + *9* ∗ *sin x* + *sin* (*3* ∗ *x*)))/(*8* ∗ (*sqrt* (*sin x* ^ *3* + *cos x* ^ *3*)) ^ *3*)) $*_H$ *e2* +
((*exp x* ∗ (*130* − *12* ∗ *cos* (*2* ∗ *x*) + *30* ∗ *cos* (*4* ∗ *x*) + *12* ∗ *cos* (*6* ∗ *x*) − *111* ∗ *sin* (*2* ∗ *x*) + *48* ∗ *sin* (*4* ∗ *x*) +
*5* ∗ *sin* (*6* ∗ *x*)))/(*64* ∗ (*sqrt* (*sin x* ^ *3* + *cos x* ^ *3*)) ^ *5*)) $*_H$ *e12*
**proof** (*auto simp add*: *w7 w1 times-hyperdual-parts*)
**show** *exp x* ∗ *inverse* (*sqrt* (*sin x* ^ *3* + *cos x* ^ *3*)) = *exp x* / *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*)
**by** (*simp add*: *divide-inverse*)

**have** *sqrt-sc3*: *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*) ^ *3* = (*sin x* ^ *3* + *cos x* ^ *3*) ∗ *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*)
**using** *s3-c3-gt-zero*
**by** (*simp add*: *power3-eq-cube*)
**then have** *inverse* (*sqrt* (*sin x* ^ *3* + *cos x* ^ *3*)) −
(*3* ∗ *cos x* ∗ *sin x* ∗ (*sin x* − *cos x*)) / (*2* ∗ *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*) ^ *3*) =
(*3* ∗ *cos x* + *5* ∗ *cos* (*3* ∗ *x*) + *9* ∗ *sin x* + *sin* (*3* ∗ *x*)) / (*8* ∗ *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*) ^ *3*)
**using** *sqrt-pos*
**apply** (*simp add*: *right-diff-distrib′ sin-times-sin cos-times-cos sin-times-cos cos-times-sin power3-eq-cube left-diff-distrib′ divide-simps*)
**by** (*simp add*: *distrib-right cos-times-cos*)
**then show** *exp x* ∗ *inverse* (*sqrt* (*sin x* ^ *3* + *cos x* ^ *3*)) −
*exp x* ∗ (*3* ∗ *cos x* ∗ *sin x* ∗ (*sin x* − *cos x*)) / (*2* ∗ *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*) ^ *3*) =
*exp x* ∗ (*3* ∗ *cos x* + *5* ∗ *cos* (*3* ∗ *x*) + *9* ∗ *sin x* + *sin* (*3* ∗ *x*)) / (*8* ∗ *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*) ^ *3*)
**by** (*simp add*: *algebra-simps*)

**have** *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*) ^ *8* = (*sin x* ^ *3* + *cos x* ^ *3*) ^ *4*
**using** *s3-c3-gt-zero*
**by** (*smt mult-2 numeral-Bit0 power2-eq-square power-even-eq real-sqrt-mult-self*)

**moreover have** *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*) ^ *5* = (*sin x* ^ *3* + *cos x* ^ *3*) ^ *2* ∗ *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*)
**by** (*simp add*: *mult.assoc power2-eq-square power5*)
**ultimately**
**have** (*90* + *6* ∗ *cos* (*4* ∗ *x*) − *123* ∗ *sin* (*2* ∗ *x*) + *9* ∗ *sin* (*6* ∗ *x*)) / (*64* ∗ *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*) ^ *5*) −
*3* ∗ (*cos x* ∗ ((*sin x* ∗ (*sin x* − *cos x*)))) / *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*) ^ *3* +
*inverse* (*sqrt* (*sin x* ^ *3* + *cos x* ^ *3*)) =
(*130* − *12* ∗ *cos* (*2* ∗ *x*) + *30* ∗ *cos* (*4* ∗ *x*) + *12* ∗ *cos* (*6* ∗ *x*) − *111* ∗ *sin* (*2* ∗ *x*) + *48* ∗ *sin* (*4* ∗ *x*) + *5* ∗ *sin* (*6* ∗ *x*)) /
(*64* ∗ *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*) ^ *5*)
**using** *sqrt-pos*
**apply** (*simp add*: *sqrt-sc3 power2-eq-square divide-simps*)
**by** (*simp add*: *distrib-right distrib-left power3-eq-cube sin-times-sin sin-times-cos cos-times-cos cos-times-sin right-diff-distrib′ left-diff-distrib′ divide-simps*)
**moreover have** ∀ *r a b c*. (*a*::*real*) ∗ *b* − *c* ∗ (*a* ∗ *r*) = *a* ∗ (*b* − *c* ∗ *r*)
**by** (*simp add*: *right-diff-distrib*)
**ultimately show** *exp x* ∗ (*90* + *6* ∗ *cos* (*4* ∗ *x*) − *123* ∗ *sin* (*2* ∗ *x*) + *9* ∗ *sin* (*6* ∗ *x*)) / (*64* ∗ *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*) ^ *5*) −
*3* ∗ (*cos x* ∗ (*exp x* ∗ (*sin x* ∗ (*sin x* − *cos x*)))) / *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*) ^ *3* +
*exp x* ∗ *inverse* (*sqrt* (*sin x* ^ *3* + *cos x* ^ *3*)) =
*exp x* ∗
(*130* − *12* ∗ *cos* (*2* ∗ *x*) + *30* ∗ *cos* (*4* ∗ *x*) + *12* ∗ *cos* (*6* ∗ *x*) − *111* ∗ *sin* (*2* ∗ *x*) + *48* ∗ *sin* (*4* ∗ *x*) + *5* ∗ *sin* (*6* ∗ *x*)) /
(*64* ∗ *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*) ^ *5*)
**by** (*metis* (*mono-tags*, *opaque-lifting*) *distrib-left mult.left-commute times-divide-eq-right*)
**qed**

— Check that we have indeed computed the hyperdual extension of our analytic test function
**moreover have** *w8-eq-hyp-fa-test*: *?w8* = (∗*h*∗ *fa-test*) (*β x*)
**using** *assms*
**by** (*simp add*: *hyp-divide-inverse hyp-fa-test-def hypext-fa-test hypext-power*)

— Now simply show that "extraction" of first and second derivatives are as expected
**ultimately**
**show** *First* (*autodiff fa-test x*) =
(*exp x* ∗ (*3* ∗ *cos x* + *5* ∗ *cos* (*3* ∗ *x*) + *9* ∗ *sin x* + *sin* (*3* ∗ *x*))) /
(*8* ∗ (*sqrt* (*sin x* ^ *3* + *cos x* ^ *3*)) ^ *3*)
**and**
*Second* (*autodiff fa-test x*) =
*exp x* ∗ (*130* − *12* ∗ *cos* (*2* ∗ *x*) +
*30* ∗ *cos* (*4* ∗ *x*) + *12* ∗ *cos* (*6* ∗ *x*) −
*111* ∗ *sin* (*2* ∗ *x*) + *48* ∗ *sin* (*4* ∗ *x*) + *5* ∗ *sin* (*6* ∗ *x*)) /
(*64* ∗ *sqrt* (*sin x* ^ *3* + *cos x* ^ *3*) ^ *5*)
**by** (*metis autodiff-sel hyperdual-comb-sel*)+

**qed**

**end**

# References

[1] J. A. Fike and J. J. Alonso. The development of hyper-dual numbers for exact second-derivative calculations. In *AIAA paper 2011-886, 49th AIAA Aerospace Sciences Meeting*, 2011.