Formal Verification of an Explicit Counterexample to the Jacobian Conjecture

Arthur Freitas Ramos David Barros Hulak
Ruy J. G. B. de Queiroz

July 21, 2026

Abstract

The Jacobian conjecture asks whether a polynomial self-map of affine space over a field of characteristic zero has a polynomial inverse whenever its Jacobian determinant is a nonzero constant [4, 5]. This entry gives an independent Isabelle/HOL verification of the explicit three-dimensional map announced by Levent Alpöge on July 20, 2026 [1]. The announcement credits Akhil Mathew with prompting the question and the AI system Claude Fable with work leading to the map; stable Lean and independent verification repositories appeared the same day [2, 3].

Isabelle proves that the formal Jacobian entries are the corresponding complex analytic partial derivatives and that the determinant is identically -2. It verifies that three distinct rational points have the same image. Scaling the first output coordinate yields determinant exactly 1 while preserving noninjectivity. The theory defines polynomial invertibility and proves that the map has no polynomial inverse. It also constructs identity-padded polynomial maps, proves their block-Jacobian determinant is unchanged, and thereby obtains counterexamples in every finite dimension at least three. The development makes no claim about the two-dimensional conjecture.

Contents

theory *Jacobian-Counterexample*
imports
Complex-Main
HOL−Analysis.Derivative
HOL−Analysis.Determinants
HOL−Decision-Procs.Commutative-Ring
HOL−Library.Cardinality
begin

1 Formal multivariate polynomials

We use a small expression datatype for multivariate polynomials. This makes polynomiality syntactic and gives a direct definition of formal partial differentiation, independent of analytic differentiability.

datatype $('v, 'a)$ *poly-expr* =
PConst $'a$
| *PVar* $'v$
| *PAdd* $('v, 'a)$ *poly-expr* $('v, 'a)$ *poly-expr*
| *PMul* $('v, 'a)$ *poly-expr* $('v, 'a)$ *poly-expr*
| *PPow* $('v, 'a)$ *poly-expr nat*

instantiation *poly-expr* :: (*type*, *zero*) *zero*
begin

definition *0* = *PConst 0*

instance ..

end

fun *eval-poly-expr* ::
$('v \Rightarrow 'a$::*semiring-1*$) \Rightarrow ('v, 'a)$ *poly-expr* $\Rightarrow 'a$ **where**
eval-poly-expr α (*PConst* c) = c
| *eval-poly-expr* α (*PVar* v) = $\alpha\ v$
| *eval-poly-expr* α (*PAdd* $p\ q$) =
eval-poly-expr $\alpha\ p$ + *eval-poly-expr* $\alpha\ q$
| *eval-poly-expr* α (*PMul* $p\ q$) =
eval-poly-expr $\alpha\ p$ $*$ *eval-poly-expr* $\alpha\ q$
| *eval-poly-expr* α (*PPow* $p\ n$) = *eval-poly-expr* $\alpha\ p\ \hat{}\ n$

fun *partial-poly-expr* ::
$'v \Rightarrow ('v, 'a$::*semiring-1*$)$ *poly-expr* $\Rightarrow ('v, 'a)$ *poly-expr* **where**
partial-poly-expr v (*PConst* c) = *PConst 0*
| *partial-poly-expr* v (*PVar* w) = *PConst* (*if* $v = w$ *then 1 else 0*)
| *partial-poly-expr* v (*PAdd* $p\ q$) =
PAdd (*partial-poly-expr* $v\ p$) (*partial-poly-expr* $v\ q$)
| *partial-poly-expr* v (*PMul* $p\ q$) =
PAdd (*PMul* (*partial-poly-expr* $v\ p$) q)

```
      (PMul p (partial-poly-expr v q))
| partial-poly-expr v (PPow p n) =
    PMul (PMul (PConst (of-nat n)) (PPow p (n − 1)))
      (partial-poly-expr v p)

lemma eval-partial-poly-expr-has-field-derivative:
  fixes α :: 'v ⇒ complex
  shows ((λu. eval-poly-expr (α(v := u)) p) has-field-derivative
    eval-poly-expr (α(v := u)) (partial-poly-expr v p)) (at u)
proof (induction p)
  case (PConst c)
  then show ?case
    by (auto intro!: derivative-eq-intros)
next
  case (PVar w)
  then show ?case
    by (cases v = w) (auto intro!: derivative-eq-intros)
next
  case (PAdd p q)
  then show ?case
    by (auto intro!: derivative-eq-intros)
next
  case (PMul p q)
  then show ?case
    by (auto intro!: derivative-eq-intros)
next
  case (PPow p n)
  then show ?case
    by (auto intro!: derivative-eq-intros)
qed

definition eval-poly-map ::
    (('n, 'a::semiring-1) poly-expr)^'n ⇒ 'a^'n ⇒ 'a^'n where
  eval-poly-map P x = (χ i. eval-poly-expr (λj. x $ j) (P $ i))

definition jacobian-matrix ::
    (('n, 'a::comm-ring-1) poly-expr)^'n ⇒ 'a^'n ⇒ 'a^'n^'n where
  jacobian-matrix P x =
    (χ i j. eval-poly-expr (λk. x $ k) (partial-poly-expr j (P $ i)))

lemma jacobian-matrix-entry-has-field-derivative:
  fixes P :: (('n, complex) poly-expr)^'n
  shows ((λu. eval-poly-expr ((λk. x $ k)(j := u)) (P $ i))
    has-field-derivative jacobian-matrix P x $ i $ j) (at (x $ j))
  unfolding jacobian-matrix-def
  using eval-partial-poly-expr-has-field-derivative[
    where α = λk. x $ k and v = j and p = P $ i and u = x $ j]
  by simp
```

definition *polynomially-invertible* ::
  $((\prime n, \prime a{::}\mathit{semiring\text{-}1})\ \mathit{poly\text{-}expr)^{\frown\prime}n \Rightarrow \mathit{bool}}$ **where**
 polynomially-invertible $P \longleftrightarrow$
  $(\exists\, Q.$
   $(\forall\, x.\ \mathit{eval\text{-}poly\text{-}map}\ Q\ (\mathit{eval\text{-}poly\text{-}map}\ P\ x) = x) \wedge$
   $(\forall\, x.\ \mathit{eval\text{-}poly\text{-}map}\ P\ (\mathit{eval\text{-}poly\text{-}map}\ Q\ x) = x))$

lemma *polynomially-invertible-imp-inj*:
 assumes *polynomially-invertible* P
 shows *inj* (*eval-poly-map* P)
proof (*rule injI*)
 fix $x\ y$
 assume *same*: *eval-poly-map* $P\ x =$ *eval-poly-map* $P\ y$
 from *assms* **obtain** Q **where** *left*:
   $\bigwedge z.$ *eval-poly-map* Q (*eval-poly-map* $P\ z) = z$
  unfolding *polynomially-invertible-def* **by** *blast*
 have $x =$ *eval-poly-map* Q (*eval-poly-map* $P\ x$)
  by (*simp add*: *left*)
 also have $\ldots =$ *eval-poly-map* Q (*eval-poly-map* $P\ y$)
  by (*simp add*: *same*)
 also have $\ldots = y$
  by (*rule left*)
 finally show $x = y$ **.**
qed

definition *keller-map* ::
  $((\prime n, \prime a{::}\mathit{comm\text{-}ring\text{-}1})\ \mathit{poly\text{-}expr)^{\frown\prime}n \Rightarrow \mathit{bool}}$ **where**
 keller-map $P \longleftrightarrow$
  $(\exists\, d.\ d \neq 0 \wedge (\forall\, x.\ \mathit{det}\ (\mathit{jacobian\text{-}matrix}\ P\ x) = d))$

definition *normalized-keller-map* ::
  $((\prime n, \prime a{::}\mathit{comm\text{-}ring\text{-}1})\ \mathit{poly\text{-}expr)^{\frown\prime}n \Rightarrow \mathit{bool}}$ **where**
 normalized-keller-map $P \longleftrightarrow$
  $(\forall\, x.\ \mathit{det}\ (\mathit{jacobian\text{-}matrix}\ P\ x) = 1)$

definition *strong-keller-counterexample* ::
  $((\prime n, \prime a{::}\mathit{comm\text{-}ring\text{-}1})\ \mathit{poly\text{-}expr)^{\frown\prime}n \Rightarrow \mathit{bool}}$ **where**
 strong-keller-counterexample $P \longleftrightarrow$
  keller-map $P \wedge \neg$ *inj* (*eval-poly-map* P)

definition *keller-counterexample* ::
  $((\prime n, \prime a{::}\mathit{comm\text{-}ring\text{-}1})\ \mathit{poly\text{-}expr)^{\frown\prime}n \Rightarrow \mathit{bool}}$ **where**
 keller-counterexample $P \longleftrightarrow$
  keller-map $P \wedge \neg$ *polynomially-invertible* P

definition *strong-normalized-keller-counterexample* ::
  $((\prime n, \prime a{::}\mathit{comm\text{-}ring\text{-}1})\ \mathit{poly\text{-}expr)^{\frown\prime}n \Rightarrow \mathit{bool}}$ **where**
 strong-normalized-keller-counterexample $P \longleftrightarrow$
  normalized-keller-map $P \wedge \neg$ *inj* (*eval-poly-map* P)

definition *normalized-keller-counterexample* ::
  $(('n, 'a{::}comm\text{-}ring\text{-}1)\ poly\text{-}expr)^{\frown}'n \Rightarrow bool$ **where**
 normalized-keller-counterexample P $\longleftrightarrow$
  normalized-keller-map P $\wedge \neg$ *polynomially-invertible P*

lemma *strong-keller-counterexample-imp-keller-counterexample*:
 strong-keller-counterexample P $\Longrightarrow$ *keller-counterexample P*
 unfolding *strong-keller-counterexample-def keller-counterexample-def*
 using *polynomially-invertible-imp-inj* **by** *blast*

lemma *strong-normalized-keller-counterexample-imp-counterexample*:
 strong-normalized-keller-counterexample P $\Longrightarrow$
  normalized-keller-counterexample P
 unfolding *strong-normalized-keller-counterexample-def*
  normalized-keller-counterexample-def
 using *polynomially-invertible-imp-inj* **by** *blast*

2 The announced polynomial map

Alpöge announced the following three-coordinate polynomial map on July 20, 2026 [1]. The announcement credits Akhil Mathew with prompting the question and the AI system Claude Fable with work leading to the example; stable independent verification repositories appeared the same day [2, 3]. The definitions below encode the map syntactically, so the subsequent Jacobian calculation is a theorem about formal polynomial differentiation rather than an imported computer-algebra result.

definition *x-poly* :: $(3, 'a{::}comm\text{-}ring\text{-}1)\ poly\text{-}expr$ **where**
 x-poly = *PVar 1*

definition *y-poly* :: $(3, 'a{::}comm\text{-}ring\text{-}1)\ poly\text{-}expr$ **where**
 y-poly = *PVar 2*

definition *z-poly* :: $(3, 'a{::}comm\text{-}ring\text{-}1)\ poly\text{-}expr$ **where**
 z-poly = *PVar 3*

definition *one-plus-xy-poly* :: $(3, 'a{::}comm\text{-}ring\text{-}1)\ poly\text{-}expr$ **where**
 one-plus-xy-poly = *PAdd* (*PConst 1*) (*PMul x-poly y-poly*)

definition *four-plus-three-xy-poly* :: $(3, 'a{::}comm\text{-}ring\text{-}1)\ poly\text{-}expr$ **where**
 four-plus-three-xy-poly =
  PAdd (*PConst 4*) (*PMul* (*PConst 3*) (*PMul x-poly y-poly*))

definition *component-a-poly* :: $(3, 'a{::}comm\text{-}ring\text{-}1)\ poly\text{-}expr$ **where**
 component-a-poly =
  PAdd (*PMul* (*PPow one-plus-xy-poly 3*) *z-poly*)
   (*PMul* (*PMul* (*PPow y-poly 2*) *one-plus-xy-poly*)

```
    four-plus-three-xy-poly)

definition component-b-poly :: (3, 'a::comm-ring-1) poly-expr where
  component-b-poly =
    PAdd y-poly
     (PAdd (PMul (PMul (PConst 3) x-poly)
        (PMul (PPow one-plus-xy-poly 2) z-poly))
      (PMul (PMul (PMul (PConst 3) x-poly) (PPow y-poly 2))
       four-plus-three-xy-poly))

definition component-c-poly :: (3, 'a::comm-ring-1) poly-expr where
  component-c-poly =
    PAdd (PMul (PConst 2) x-poly)
     (PAdd (PMul (PConst (-3)) (PMul (PPow x-poly 2) y-poly))
      (PMul (PConst (-1)) (PMul (PPow x-poly 3) z-poly)))

definition alpoge-map :: ((3, 'a::comm-ring-1) poly-expr)^3 where
  alpoge-map =
    vector [component-a-poly, component-b-poly, component-c-poly]

lemma eval-alpoge-map:
  eval-poly-map alpoge-map (vector [x, y, z]) =
   vector [
    (1 + x * y)^3 * z + y^2 * (1 + x * y) * (4 + 3 * x * y),
    y + 3 * x * (1 + x * y)^2 * z + 3 * x * y^2 * (4 + 3 * x * y),
    2 * x - 3 * x^2 * y - x^3 * z]
  unfolding eval-poly-map-def alpoge-map-def component-a-poly-def
   component-b-poly-def component-c-poly-def one-plus-xy-poly-def
   four-plus-three-xy-poly-def x-poly-def y-poly-def z-poly-def
  by (simp add: vec-eq-iff vector-def forall-3 algebra-simps)

corollary alpoge-jacobian-entry-has-field-derivative:
  fixes x :: complex^3
  shows ((λu.
      eval-poly-map (alpoge-map :: ((3, complex) poly-expr)^3)
       (χ k. if k = j then u else x $ k) $ i)
    has-field-derivative
      jacobian-matrix (alpoge-map :: ((3, complex) poly-expr)^3) x $ i $ j)
    (at (x $ j))
  using jacobian-matrix-entry-has-field-derivative[
    where P = alpoge-map :: ((3, complex) poly-expr)^3
      and x = x and i = i and j = j]
  by (simp add: eval-poly-map-def fun-upd-def)

lemma jacobian-alpoge-map:
  jacobian-matrix alpoge-map (vector [x, y, z]) =
   vector [
    vector [
      3 * y * (1 + x * y)^2 * z +
```

```
    y^3 * (4 + 3 * x * y) + 3 * y^3 * (1 + x * y),
   3 * x * (1 + x * y)^2 * z +
    2 * y * (1 + x * y) * (4 + 3 * x * y) +
    x * y^2 * (4 + 3 * x * x * y) + 3 * x * y^2 * (1 + x * y),
   (1 + x * y)^3],
  vector [
   3 * (1 + x * y)^2 * z + 6 * x * y * (1 + x * y) * z +
    3 * y^2 * (4 + 3 * x * y) + 9 * x * y^3,
   1 + 6 * x^2 * (1 + x * y) * z +
    6 * x * y * (4 + 3 * x * y) + 9 * x^2 * y^2,
   3 * x * (1 + x * y)^2],
  vector [
   2 - 6 * x * y - 3 * x^2 * z,
   -3 * x^2,
   -(x^3)]]
 unfolding jacobian-matrix-def alpoge-map-def component-a-poly-def
  component-b-poly-def component-c-poly-def one-plus-xy-poly-def
  four-plus-three-xy-poly-def x-poly-def y-poly-def z-poly-def
 apply (simp add: vec-eq-iff vector-def forall-3 algebra-simps)
 apply (intro conjI)
 apply ring+
 done

lemma alpoge-jacobian-det-coordinates:
 det (jacobian-matrix alpoge-map (vector [x, y, z])) = -2
 apply (subst jacobian-alpoge-map)
 apply (simp add: det-3 vector-def)
 apply ring
 done

theorem alpoge-jacobian-det:
 det (jacobian-matrix alpoge-map p) = -2
proof -
 have p: p = vector [p $ 1, p $ 2, p $ 3]
  by (simp add: vec-eq-iff vector-def forall-3)
 show ?thesis
  apply (subst p)
  apply (rule alpoge-jacobian-det-coordinates)
  done
qed
```

3 An explicit three-point fibre

```
definition collision-point-0 :: 'a::field-char-0^3 where
 collision-point-0 = vector [0, 0, -1 / 4]

definition collision-point-pos :: 'a::field-char-0^3 where
 collision-point-pos = vector [1, -3 / 2, 13 / 2]
```

definition *collision-point-neg* :: *'a*::*field-char-0*^*3* **where**
collision-point-neg = *vector* [−*1*, *3* / *2*, *13* / *2*]

definition *collision-value* :: *'a*::*field-char-0*^*3* **where**
collision-value = *vector* [−*1* / *4*, *0*, *0*]

lemma *collision-points-distinct*:
distinct [*collision-point-0*, *collision-point-pos*, *collision-point-neg*]
unfolding *collision-point-0-def collision-point-pos-def collision-point-neg-def*
by (*simp add*: *vec-eq-iff vector-def forall-3*)

lemma *alpoge-collision*:
eval-poly-map alpoge-map collision-point-0 = *collision-value*
eval-poly-map alpoge-map collision-point-pos = *collision-value*
eval-poly-map alpoge-map collision-point-neg = *collision-value*
unfolding *collision-point-0-def collision-point-pos-def collision-point-neg-def*
collision-value-def eval-alpoge-map
by (*simp-all add*: *vec-eq-iff vector-def forall-3 field-simps algebra-simps*)

lemma *alpoge-map-not-injective*:
¬ *inj*
(*eval-poly-map* (*alpoge-map* :: ((*3*, *'a*::*field-char-0*) *poly-expr*)^*3*))
proof
assume *inj*: *inj*
(*eval-poly-map* (*alpoge-map* :: ((*3*, *'a*) *poly-expr*)^*3*))
have *same*:
eval-poly-map (*alpoge-map* :: ((*3*, *'a*) *poly-expr*)^*3*)
(*collision-point-0* :: *'a*^*3*) =
eval-poly-map alpoge-map (*collision-point-pos* :: *'a*^*3*)
by (*simp only*: *alpoge-collision*)
have (*collision-point-0* :: *'a*^*3*) = *collision-point-pos*
using *inj same* **by** (*rule injD*)
moreover have (*collision-point-0* :: *'a*^*3*) ≠ *collision-point-pos*
unfolding *collision-point-0-def collision-point-pos-def*
by (*simp add*: *vec-eq-iff vector-def forall-3*)
ultimately show *False*
by *contradiction*
qed

theorem *alpoge-map-is-strong-keller-counterexample*:
strong-keller-counterexample
(*alpoge-map* :: ((*3*, *'a*::*field-char-0*) *poly-expr*)^*3*)
unfolding *strong-keller-counterexample-def keller-map-def*
using *alpoge-jacobian-det alpoge-map-not-injective*
by (*intro conjI exI*[*of* - −*2*]) *auto*

corollary *alpoge-map-is-keller-counterexample*:
keller-counterexample
(*alpoge-map* :: ((*3*, *'a*::*field-char-0*) *poly-expr*)^*3*)

using *alpoge-map-is-strong-keller-counterexample*
by (*rule strong-keller-counterexample-imp-keller-counterexample*)

corollary *alpoge-map-not-polynomially-invertible*:
¬ *polynomially-invertible*
(*alpoge-map* :: ((*3*, *'a*::*field-char-0*) *poly-expr*)^*3*)
using *alpoge-map-is-keller-counterexample*
unfolding *keller-counterexample-def* **by** *blast*

4 A determinant-one normalization

definition *normalized-alpoge-map* ::
((*3*, *'a*::*field-char-0*) *poly-expr*)^*3* **where**
normalized-alpoge-map =
vector [
PMul (*PConst* (−*1* / *2*)) *component-a-poly*,
component-b-poly,
component-c-poly]

lemma *eval-normalized-alpoge-map*:
eval-poly-map normalized-alpoge-map (*vector* [*x*, *y*, *z*]) =
vector [
(−*1* / *2*) *
((*1* + *x* * *y*)^*3* * *z* + *y*^*2* * (*1* + *x* * *y*) * (*4* + *3* * *x* * *y*)),
y + *3* * *x* * (*1* + *x* * *y*)^*2* * *z* + *3* * *x* * *y*^*2* * (*4* + *3* * *x* * *y*),
2 * *x* − *3* * *x*^*2* * *y* − *x*^*3* * *z*]
unfolding *normalized-alpoge-map-def eval-poly-map-def component-a-poly-def*
component-b-poly-def component-c-poly-def one-plus-xy-poly-def
four-plus-three-xy-poly-def x-poly-def y-poly-def z-poly-def
by (*simp add*: *vec-eq-iff vector-def forall-3 algebra-simps*)

lemma *jacobian-normalized-alpoge-map*:
jacobian-matrix normalized-alpoge-map p =
(χ *i j*.
if i = *1 then*
(−*1* / *2*) * (*jacobian-matrix alpoge-map p* $ *i* $ *j*)
else jacobian-matrix alpoge-map p $ *i* $ *j*)
unfolding *normalized-alpoge-map-def alpoge-map-def jacobian-matrix-def*
by (*simp add*: *vec-eq-iff vector-def forall-3*)

theorem *normalized-alpoge-jacobian-det*:
det (*jacobian-matrix normalized-alpoge-map p*) = *1*
proof −
let *?J* = *jacobian-matrix alpoge-map p*
have *scaled*:
jacobian-matrix normalized-alpoge-map p =
(χ *i*. *if i* = *1 then* (−*1* / *2*) *s *row i ?J else row i ?J*)
unfolding *jacobian-normalized-alpoge-map row-def*
by (*simp add*: *vec-eq-iff*)

```
  have det-scaled:
    det (χ i. if i = 1 then (−1 / 2) ∗s row i ?J else row i ?J) =
     (−1 / 2) ∗ det (χ i. if i = 1 then row i ?J else row i ?J)
    by (rule det-row-mul)
  have rows: (χ i. row i ?J) = ?J
    unfolding row-def
    by (simp add: vec-eq-iff)
  show ?thesis
    apply (subst scaled)
    apply (subst det-scaled)
    using alpoge-jacobian-det[of p]
    apply (simp add: rows field-simps)
    done
qed

lemma normalized-alpoge-collision:
  eval-poly-map normalized-alpoge-map collision-point-0 =
     vector [1 / 8, 0, 0]
  eval-poly-map normalized-alpoge-map collision-point-pos =
     vector [1 / 8, 0, 0]
  eval-poly-map normalized-alpoge-map collision-point-neg =
     vector [1 / 8, 0, 0]
  unfolding collision-point-0-def collision-point-pos-def collision-point-neg-def
    eval-normalized-alpoge-map
  by (simp-all add: vec-eq-iff vector-def forall-3 field-simps algebra-simps)

lemma normalized-alpoge-map-not-injective:
  ¬ inj
    (eval-poly-map
      (normalized-alpoge-map :: ((3, 'a::field-char-0) poly-expr)^3))
proof
  assume inj: inj
    (eval-poly-map (normalized-alpoge-map :: ((3, 'a) poly-expr)^3))
  have same:
    eval-poly-map (normalized-alpoge-map :: ((3, 'a) poly-expr)^3)
       (collision-point-0 :: 'a^3) =
     eval-poly-map normalized-alpoge-map (collision-point-pos :: 'a^3)
    by (simp only: normalized-alpoge-collision)
  have (collision-point-0 :: 'a^3) = collision-point-pos
    using inj same by (rule injD)
  moreover have (collision-point-0 :: 'a^3) ≠ collision-point-pos
    unfolding collision-point-0-def collision-point-pos-def
    by (simp add: vec-eq-iff vector-def forall-3)
  ultimately show False
    by contradiction
qed

theorem normalized-alpoge-map-is-strong-counterexample:
  strong-normalized-keller-counterexample
```

```
  (normalized-alpoge-map :: ((3, 'a::field-char-0) poly-expr)^3)
 unfolding strong-normalized-keller-counterexample-def normalized-keller-map-def
 using normalized-alpoge-jacobian-det normalized-alpoge-map-not-injective
 by blast

theorem normalized-alpoge-map-is-counterexample:
 normalized-keller-counterexample
  (normalized-alpoge-map :: ((3, 'a::field-char-0) poly-expr)^3)
 using normalized-alpoge-map-is-strong-counterexample
 by (rule strong-normalized-keller-counterexample-imp-counterexample)

corollary normalized-alpoge-map-not-polynomially-invertible:
 ¬ polynomially-invertible
  (normalized-alpoge-map :: ((3, 'a::field-char-0) poly-expr)^3)
 using normalized-alpoge-map-is-counterexample
 unfolding normalized-keller-counterexample-def by blast
```

This is the determinant-one form of the counterexample. Since every coordinate is represented by a polynomial expression, the theorem directly refutes the normalized dimension-three Jacobian conjecture over any field of characteristic zero.

```
corollary complex-jacobian-conjecture-counterexample:
 normalized-keller-counterexample
  (normalized-alpoge-map :: ((3, complex) poly-expr)^3)
 by (rule normalized-alpoge-map-is-counterexample)
```

5 Identity padding in higher dimensions

```
lemma prod-UNIV-sum:
 fixes f :: ('n::finite + 'm::finite) ⇒ 'a::comm-monoid-mult
 shows (∏ x∈UNIV. f x) =
  (∏ i∈UNIV. f (Inl i)) * (∏ j∈UNIV. f (Inr j))
proof −
 have disj: range (Inl :: 'n ⇒ 'n + 'm) ∩ range Inr = {}
  by auto
 have split:
  prod f (range (Inl :: 'n ⇒ 'n + 'm) ∪ range Inr) =
   prod f (range Inl) * prod f (range Inr)
  using disj by (simp add: prod.union-disjoint)
 have left:
  prod f (range (Inl :: 'n ⇒ 'n + 'm)) =
   (∏ i∈UNIV. f (Inl i))
  by (simp add: prod.reindex)
 have right:
  prod f (range (Inr :: 'm ⇒ 'n + 'm)) =
   (∏ j∈UNIV. f (Inr j))
  by (simp add: prod.reindex)
 show ?thesis
```

```
    using split left right by (simp add: UNIV-sum)
qed

definition block-identity-matrix ::
    'a::semiring-1^'n^'n ⇒ 'a^('n + 'm::finite)^('n + 'm) where
  block-identity-matrix A =
    (χ i j.
      case i of
        Inl r ⇒ (case j of Inl c ⇒ A $ r $ c | Inr - ⇒ 0)
      | Inr r ⇒ (case j of Inl - ⇒ 0 | Inr c ⇒ if r = c then 1 else 0))

lemma det-block-identity-matrix:
  fixes A :: 'a::comm-ring-1^'n::finite^'n
  shows det (block-identity-matrix A :: 'a^('n + 'm::finite)^('n + 'm)) = det A
proof −
  let ?U = UNIV :: 'n set
  let ?B = range (Inl :: 'n ⇒ 'n + 'm)
  let ?PU = {p. p permutes ?U}
  let ?PB = {p. p permutes ?B}
  let ?PAll = {p. p permutes (UNIV :: ('n + 'm) set)}
  let ?lift = λp x.
    if x ∈ ?B then Inl (p (inv-into ?U Inl x)) else x
  let ?term = λp. of-int (sign p) *
    (∏ i∈UNIV. block-identity-matrix A $ i $ p i)
  have finite-all: finite (UNIV :: ('n + 'm) set)
    by simp
  have PB-sub: ?PB ⊆ ?PAll
    by (auto intro: permutes-subset)
  have zero-out: ∀ p∈?PAll − ?PB. ?term p = 0
  proof (intro ballI)
    fix p
    assume p: p ∈ ?PAll − ?PB
    have moves-right: ∃ j. p (Inr j) ≠ Inr j
    proof (rule ccontr)
      assume ∄ j. p (Inr j) ≠ Inr j
      then have fixed-outside: ⋀x. x ∉ ?B ⟹ p x = x
        by (metis range-eqI sum.exhaust-sel)
      from p have bij: ⋀y. ∃!x. p x = y
        by (auto simp: permutes-def)
      have p permutes ?B
        unfolding permutes-def using fixed-outside bij by blast
      with p show False
        by blast
    qed
    then obtain j where moved: p (Inr j) ≠ Inr j
      by blast
    have entry-zero:
      block-identity-matrix A $ Inr j $ p (Inr j) = 0
      using moved by (cases p (Inr j)) (auto simp: block-identity-matrix-def)
```

```
  have product-zero:
    (∏ i∈UNIV. block-identity-matrix A $ i $ p i) = 0
    by (rule prod-zero) (use entry-zero in auto)
  show ?term p = 0
    by (simp add: product-zero)
qed
have sum-restrict: sum ?term ?PB = sum ?term ?PAll
  by (rule sum.mono-neutral-cong-left[
    OF finite-permutations[OF finite-all] PB-sub zero-out]) auto
have det-restrict:
  det (block-identity-matrix A :: 'a^('n + 'm)^('n + 'm)) = sum ?term ?PB
  unfolding det-def using sum-restrict by (rule sym)
have bij-inl: bij-betw (Inl :: 'n ⇒ 'n + 'm) ?U ?B
  by (rule bij-betw-imageI) auto
have bij-lift: bij-betw ?lift ?PU ?PB
  by (rule bij-betw-permutations[OF bij-inl])
have lift-eq-map:
  ?lift p = map-permutation ?U (Inl :: 'n ⇒ 'n + 'm) p for p
  unfolding map-permutation-def
  by (auto simp: fun-eq-iff restrict-id-def o-def)
have sign-lift: sign (?lift p) = sign p if p ∈ ?PU for p
  using sign-map-permutation[of Inl :: 'n ⇒ 'n + 'm ?U p]
    that by (simp add: lift-eq-map)
have product-lift:
  (∏ i∈UNIV. block-identity-matrix A $ i $ ?lift p i) =
   (∏ i∈UNIV. A $ i $ p i)
  if p ∈ ?PU for p
proof −
  note split = prod-UNIV-sum[
    of λi. block-identity-matrix A $ i $ ?lift p i]
  have in-left: Inl i ∈ ?B for i
    by auto
  have not-left: Inr j ∉ ?B for j
    by auto
  have inv-left: inv-into ?U Inl (Inl i) = i for i
    by (rule inv-into-f-f) auto
  show ?thesis
    using split that
    by (simp add: block-identity-matrix-def in-left not-left inv-left)
qed
have term-lift:
  ?term (?lift p) =
   of-int (sign p) * (∏ i∈UNIV. A $ i $ p i)
  if p ∈ ?PU for p
  using sign-lift[OF that] product-lift[OF that] by simp
have reindex:
  sum (λp. ?term (?lift p)) ?PU = sum ?term ?PB
  by (rule sum.reindex-bij-betw[OF bij-lift])
have sum ?term ?PB = sum (λp. ?term (?lift p)) ?PU
```

```
    using reindex by (rule sym)
  also have ... =
      sum (λp. of-int (sign p) * (∏ i∈UNIV. A $ i $ p i)) ?PU
    by (rule sum.cong) (auto intro: term-lift)
  also have ... = det A
    by (simp only: det-def)
  finally show ?thesis
    using det-restrict by simp
qed

fun rename-poly-expr ::
    ('v ⇒ 'w) ⇒ ('v, 'a) poly-expr ⇒ ('w, 'a) poly-expr where
  rename-poly-expr f (PConst c) = PConst c
| rename-poly-expr f (PVar v) = PVar (f v)
| rename-poly-expr f (PAdd p q) =
     PAdd (rename-poly-expr f p) (rename-poly-expr f q)
| rename-poly-expr f (PMul p q) =
     PMul (rename-poly-expr f p) (rename-poly-expr f q)
| rename-poly-expr f (PPow p n) = PPow (rename-poly-expr f p) n

lemma eval-rename-poly-expr:
  eval-poly-expr α (rename-poly-expr f p) =
   eval-poly-expr (α ∘ f) p
  by (induction p) simp-all

lemma partial-rename-poly-expr-Inl:
  partial-poly-expr (Inl v) (rename-poly-expr Inl p) =
   rename-poly-expr Inl (partial-poly-expr v p)
  by (induction p) simp-all

lemma eval-partial-rename-poly-expr-Inr:
  eval-poly-expr α
    (partial-poly-expr (Inr w) (rename-poly-expr Inl p)) = 0
  by (induction p) simp-all

definition join-vector ::
    'a^'n ⇒ 'a^'m ⇒ 'a^('n + 'm) where
  join-vector x y =
    (χ i. case i of Inl j ⇒ x $ j | Inr j ⇒ y $ j)

lemma join-vector-Inl [simp]: join-vector x y $ Inl i = x $ i
  by (simp add: join-vector-def)

lemma join-vector-Inr [simp]: join-vector x y $ Inr j = y $ j
  by (simp add: join-vector-def)

lemma join-vector-comp-Inl [simp]:
  ($) (join-vector x y) ∘ Inl = ($) x
  by (rule ext) simp
```

lemma *join-vector-comp-Inr* [*simp*]:
(\$) (*join-vector x y*) ∘ *Inr* = (\$) *y*
by (*rule ext*) *simp*

lemma *join-vector-eq-iff* [*simp*]:
join-vector x y = *join-vector x′ y′* ⟷ *x* = *x′* ∧ *y* = *y′*
by (*simp add*: *join-vector-def vec-eq-iff split-sum-all*)

definition *pad-poly-map* ::
(('*n*, '*a*::*semiring-1*) *poly-expr*)^'*n* ⇒
(('*n* + '*m*::*finite*, '*a*) *poly-expr*)^('*n* + '*m*) **where**
pad-poly-map P =
(χ *i*.
case i of
Inl j ⇒ *rename-poly-expr Inl* (*P* \$ *j*)
| *Inr j* ⇒ *PVar* (*Inr j*))

lemma *eval-pad-poly-map*:
eval-poly-map (*pad-poly-map P*) (*join-vector x y*) =
join-vector (*eval-poly-map P x*) *y*
unfolding *eval-poly-map-def pad-poly-map-def*
by (*simp add*: *vec-eq-iff split-sum-all eval-rename-poly-expr*)

lemma *jacobian-pad-poly-map*:
jacobian-matrix (*pad-poly-map P*) (*join-vector x y*) =
block-identity-matrix (*jacobian-matrix P x*)
unfolding *jacobian-matrix-def pad-poly-map-def*
block-identity-matrix-def
by (*simp add*: *vec-eq-iff split-sum-all eval-rename-poly-expr*
partial-rename-poly-expr-Inl eval-partial-rename-poly-expr-Inr)

corollary *jacobian-det-pad-poly-map*:
det (*jacobian-matrix* (*pad-poly-map P*) (*join-vector x y*)) =
det (*jacobian-matrix P x*)
by (*simp add*: *jacobian-pad-poly-map det-block-identity-matrix*)

definition *left-subvector* :: '*a*^('*n*::*finite* + '*m*::*finite*) ⇒ '*a*^'*n* **where**
left-subvector z = (χ *i*. *z* \$ *Inl i*)

definition *right-subvector* :: '*a*^('*n*::*finite* + '*m*::*finite*) ⇒ '*a*^'*m* **where**
right-subvector z = (χ *j*. *z* \$ *Inr j*)

lemma *join-vector-subvectors* [*simp*]:
join-vector (*left-subvector z*) (*right-subvector z*) = *z*
by (*simp add*: *join-vector-def left-subvector-def right-subvector-def*
vec-eq-iff split-sum-all)

lemma *pad-poly-map-not-injective*:

```
  fixes P :: (('n, 'a::semiring-1) poly-expr)^'n
  assumes ¬ inj (eval-poly-map P)
  shows ¬ inj
    (eval-poly-map
      (pad-poly-map P ::
        (('n + 'm::finite, 'a) poly-expr)^('n + 'm)))
proof −
  from assms obtain x x' where
      same: eval-poly-map P x = eval-poly-map P x' and neq: x ≠ x'
    by (auto simp: inj-def)
  let ?z = 0 :: 'a^'m
  have same-pad:
    eval-poly-map (pad-poly-map P) (join-vector x ?z) =
     eval-poly-map (pad-poly-map P) (join-vector x' ?z)
    by (simp add: eval-pad-poly-map same)
  show ?thesis
  proof
    assume pad-inj: inj
      (eval-poly-map
        (pad-poly-map P ::
          (('n + 'm, 'a) poly-expr)^('n + 'm)))
    have join-vector x ?z = join-vector x' ?z
      using pad-inj same-pad by (rule injD)
    then have x = x'
      by simp
    with neq show False
      by contradiction
  qed
qed

lemma strong-normalized-keller-counterexample-pad-poly-map:
  fixes P :: (('n, 'a::comm-ring-1) poly-expr)^'n
  assumes strong-normalized-keller-counterexample P
  shows strong-normalized-keller-counterexample
    (pad-poly-map P ::
      (('n + 'm::finite, 'a) poly-expr)^('n + 'm))
proof −
  from assms have base-det:
      ⋀x. det (jacobian-matrix P x) = 1
    unfolding strong-normalized-keller-counterexample-def
      normalized-keller-map-def by blast
  have padded-det:
      ⋀z. det (jacobian-matrix
        (pad-poly-map P ::
          (('n + 'm, 'a) poly-expr)^('n + 'm)) z) = 1
  proof −
    fix z :: 'a^('n + 'm)
    have det (jacobian-matrix (pad-poly-map P) z) =
        det (jacobian-matrix P (left-subvector z))
```

```
    using jacobian-det-pad-poly-map[
      of P left-subvector z right-subvector z]
    by simp
  also have ... = 1
    by (rule base-det)
  finally show det (jacobian-matrix
    (pad-poly-map P ::
      (('n + 'm, 'a) poly-expr)^('n + 'm)) z) = 1 .
qed
have padded-not-inj:
  ¬ inj (eval-poly-map
    (pad-poly-map P ::
      (('n + 'm, 'a) poly-expr)^('n + 'm)))
  using assms
  unfolding strong-normalized-keller-counterexample-def
  by (intro pad-poly-map-not-injective) blast
show ?thesis
  unfolding strong-normalized-keller-counterexample-def
    normalized-keller-map-def
  using padded-det padded-not-inj by blast
qed

theorem normalized-alpoge-padding-is-strong-counterexample:
  strong-normalized-keller-counterexample
    (pad-poly-map
      (normalized-alpoge-map :: ((3, 'a::field-char-0) poly-expr)^3) ::
      ((3 + 'm::finite, 'a) poly-expr)^(3 + 'm))
  using normalized-alpoge-map-is-strong-counterexample
  by (rule strong-normalized-keller-counterexample-pad-poly-map)

theorem normalized-alpoge-padding-is-counterexample:
  normalized-keller-counterexample
    (pad-poly-map
      (normalized-alpoge-map :: ((3, 'a::field-char-0) poly-expr)^3) ::
      ((3 + 'm::finite, 'a) poly-expr)^(3 + 'm))
  using normalized-alpoge-padding-is-strong-counterexample
  by (rule strong-normalized-keller-counterexample-imp-counterexample)

corollary complex-jacobian-conjecture-counterexample-in-higher-dimension:
  normalized-keller-counterexample
    (pad-poly-map
      (normalized-alpoge-map :: ((3, complex) poly-expr)^3) ::
      ((3 + 'm::finite, complex) poly-expr)^(3 + 'm))
  by (rule normalized-alpoge-padding-is-counterexample)

lemma padded-dimension-card:
  CARD(3 + 'm::finite) = 3 + CARD('m)
  by (simp add: card-UNIV-sum)
```

lemma *padded-dimension-greater-than-three*:
3 < *CARD*(*3* + *'m*::*finite*)
using *zero-less-card-finite*[**where** *'a* = *'m*]
by (*simp add*: *padded-dimension-card*)

The extra finite type *'m* is nonempty, so the padded coordinate type has dimension strictly greater than three. Conversely, every concrete finite dimension above three can be represented by choosing an extra finite type of the required cardinality. Thus the dimension-three counterexample yields counterexamples in all dimensions $n > 3$. Together with the original theorem this covers every $n \geq 3$. Nothing in this development addresses the two-dimensional conjecture.

end

References

[1] L. Alpöge. An Explicit Counterexample to the Jacobian Conjecture. Post on X, July 2026. `https://x.com/__alpoge__/status/2079028340955197566`, posted July 20, 2026.

[2] D. Cureton. Levent Alpöge/Fable 5's Counterexample to the Jacobian Conjecture in Lean 4. GitHub repository, July 2026. Immutable commit: `https://github.com/deancureton/jacobian/tree/0d4a9212d874226ad81ce5a926becddfa94e6a88`.

[3] A. Harrison. Machine-Verified Anatomy of the Alpöge–Mathew–Fable Counterexample to the Jacobian Conjecture. GitHub repository, July 2026. Independent verification and Lean certificate, immutable commit: `https://github.com/DrAlexHarrison/jacobian-anatomy/tree/74808fb2e1c1691b0007576ba0508e5e7cdcb1e3`.

[4] O.-H. Keller. Ganze Cremona-Transformationen. *Monatshefte für Mathematik und Physik*, 47:299–306, 1939. DOI: `https://doi.org/10.1007/BF01695502`.

[5] A. van den Essen. *Polynomial Automorphisms and the Jacobian Conjecture*, volume 190 of *Progress in Mathematics*. Birkhäuser, Basel, 2000. DOI: `https://doi.org/10.1007/978-3-0348-8440-2`.