Fundamentals of Unconstrained Optimization

Dustin Bryant

February 6, 2026

Abstract

As formal methods gain traction in machine learning and numerical analysis, the community needs computer-checked proofs of core optimization results. Existing Isabelle libraries still lack a foundational framework for unconstrained optimization. We close this gap with a comprehensive Isabelle/HOL development that formalizes:

(1) minimizers, strict and isolated local minimizers;

(2) first- and second-order optimality conditions for scalar functions $f : \mathbb{R} \to \mathbb{R}$;

(3) first-order optimality conditions for vector functions $g : \mathbb{R}^n \to \mathbb{R}$; and

(4) a worked example showing that the continuous function

$$h(x) = \begin{cases} x^4\big(\cos(1/x) + 2\big), & x \neq 0, \\ 0, & x = 0 \end{cases}$$

has a *strict* but *non-isolated* local minimizer at $x = 0$.

The new session `Unconstrained_Optimization` provides sound, reusable foundations for future proof-checking tools and mechanized research in optimization, analysis, and algorithmic correctness.

Contents

1 Auxiliary Facts

```
theory Auxilary-Facts
  imports
    Sigmoid-Universal-Approximation.Limits-Higher-Order-Derivatives
begin
```

1.1 Differentiation Lemmas

```
lemma has-derivative-imp:
  fixes f :: real ⇒ real
  assumes (f has-derivative f′) (at x)
  shows f differentiable (at x) ∧ deriv f x = f′ 1
proof safe
  show f differentiable at x
    by (meson assms differentiableI)
  then show deriv f x = f′ 1
    by (metis DERIV-deriv-iff-real-differentiable assms has-derivative-unique
        has-field-derivative-imp-has-derivative mult.comm-neutral)
qed

lemma DERIV-inverse-func:
  assumes x ≠ 0
  shows DERIV (λw. 1 / w) x :> −1 / x^2
proof −
  have inverse = (/) (1::′a)
    using inverse-eq-divide by auto
  then show ?thesis
     by (metis (no-types) DERIV-inverse assms divide-minus-left numeral-2-eq-2
power-one-over)
qed

lemma power-rule:
```

```
  fixes z :: real and n :: nat
  shows deriv (λx. x ^ n) z = (if n = 0 then 0 else real n ∗ z ^ (n − 1))
  by(subst deriv-pow, simp-all)
```

1.1.1 Transfer Lemmas

```
lemma has-derivative-transfer-on-ball:
  fixes f g :: real ⇒ real
  assumes eps-gt0: 0 < ε
  assumes eq-on-ball: ∀ y. y ∈ ball x ε ⟶ f y = g y
  assumes f-has-deriv: (f has-derivative D) (at x)
  shows (g has-derivative D) (at x)
  using centre-in-ball eps-gt0 eq-on-ball f-has-deriv
    has-derivative-transform-within-open by blast

corollary field-differentiable-transfer-on-ball:
  fixes f g :: real ⇒ real
  assumes 0 < ε
  assumes eq-on-ball: ∀ y. y ∈ ball x ε ⟶ f y = g y
  assumes f-diff: f field-differentiable at x
  shows g field-differentiable at x
  by (metis UNIV-I assms dist-commute-lessI
      field-differentiable-transform-within mem-ball)
```

1.2 Trigonometric Contraction

```
lemma cos-contractive:
  fixes x y :: real
  shows |cos x − cos y| ≤ |x − y|
proof −
  have |cos x − cos y| = |−2 ∗ sin ((x + y) / 2) ∗ sin ((x − y) / 2)|
    by (smt (verit) cos-diff-cos mult-minus-left)
  also have ... ≤ |sin ((x + y) / 2)| ∗ (2∗ |sin ((x − y) / 2)|)
    by (subst abs-mult, force)
  also have ... ≤ 2 ∗ |sin ((x − y) / 2)|
  proof −
    have |sin ((x + y) / 2)| ≤ 1
      using abs-sin-le-one by blast
    then have |sin ((x + y) / 2)| ∗ (2∗ |sin ((x − y) / 2)|) ≤ 1 ∗ (2∗ |sin ((x −
y) / 2)|)
      by(rule mult-right-mono, simp)
    then show ?thesis
      by linarith
  qed
  also have ... ≤ 2 ∗ |(x − y) / 2|
    using abs-sin-le-one by (smt (verit, del-insts) abs-sin-x-le-abs-x)
  also have ... = |x − y|
    by simp
  finally show ?thesis.
qed
```

```
lemma sin-contractive:
  fixes x y :: real
  shows |sin x − sin y| ≤ |x − y|
proof −
  have |sin x − sin y| = |2 ∗ cos ((x + y) / 2) ∗ sin ((x − y) / 2)|
    by (metis (no-types) mult.assoc mult.commute sin-diff-sin)
  also have ... ≤ |cos ((x + y) / 2)| ∗ (2 ∗ |sin ((x − y) / 2)|)
    by (subst abs-mult, force)
  also have ... ≤ 2 ∗ |sin ((x − y) / 2)|
  proof −
    have |cos ((x + y) / 2)| ≤ 1
      using abs-cos-le-one by blast
    then have |cos ((x + y) / 2)| ∗ (2 ∗ |sin ((x − y) / 2)|) ≤ 1 ∗ (2 ∗ |sin ((x − y) / 2)|)
      by (rule mult-right-mono, simp)
    then show ?thesis
      by linarith
  qed
  also have ... ≤ 2 ∗ |(x − y) / 2|
    using abs-sin-le-one by (smt (verit, del-insts) abs-sin-x-le-abs-x)
  also have ... = |x − y|
    by simp
  finally show ?thesis.
qed
```

1.3 Algebraic Factorizations

```
lemma biquadrate-diff-biquadrate-factored:
  fixes x y::real
  shows y^4 − x^4 = (y − x) ∗ (y^3 + y^2 ∗ x + y ∗ x^2 + x^3)
proof −
    have y^4 − x^4 = (y^2 − x^2) ∗ (y^2 + x^2)
     by (metis mult.commute numeral-Bit0 power-add square-diff-square-factored)
    also have ... = (y − x) ∗ (y + x) ∗ (y^2 + x^2)
      by (simp add: power2-eq-square square-diff-square-factored)
    also have ... = (y − x) ∗ (y^3 + y^2 ∗ x + y ∗ x^2 + x^3)
      by (simp add: distrib-left mult.commute power2-eq-square power3-eq-cube)
    finally show ?thesis.
qed
```

1.4 Specific Trigonometric Values

```
lemma sin-5pi-div-4: sin (5 ∗ pi / 4) = − (sqrt 2 / 2)
proof −
  have 5 ∗ pi / 4 = pi + pi / 4
    by simp
  moreover have sin (pi + x) = − sin x for x
    by (simp add: sin-add)
  ultimately show ?thesis
```

```
    using sin-45 by presburger
qed

lemma cos-5pi-div-4: cos (5 ∗ pi / 4) = − (sqrt 2 / 2)
proof −
  have 5 ∗ pi / 4 = pi + pi / 4
    by simp
  moreover have cos (pi + x) = − cos x for x
    by (simp add: cos-add)
  moreover have cos (pi / 4) = sqrt 2 / 2
    by (simp add: real-div-sqrt cos-45)
  ultimately show ?thesis
    by presburger
qed
```

1.5 Local Sign Preservation of Continuous Functions

1.5.1 Local Positivity

```
lemma cont-at-pos-imp-loc-pos:
  fixes g :: real ⇒ real and x :: real
  assumes continuous (at x) g and g x > 0
  shows ∃δ > 0. ∀y. |y − x| < δ ⟶ g y > 0
proof −
  from assms obtain δ where δ-pos: δ > 0
    and ∀y. |y − x| < δ ⟶ |g y − g x| < (g x)/2
    using continuous-at-eps-delta half-gt-zero by blast
  then have ∀y. |y − x| < δ ⟶ g y > 0
    by (smt (verit, best) field-sum-of-halves)
  then show ?thesis
    using δ-pos by blast
qed

lemma cont-at-pos-imp-loc-pos':
  fixes g :: real ⇒ real and x :: real
  assumes continuous (at x) g and g x > 0
  shows ∃Δ > 0. ∀δ. 0 < δ ∧ δ ≤ Δ ⟶ (∀y. |y − x| < δ ⟶ g y > 0)
proof −
  from assms obtain δ where δ-pos: δ > 0 and H: ∀y. |y − x| < δ ⟶ g y > 0
    using cont-at-pos-imp-loc-pos by blast
  have ∀δ' ≤ δ. ∀y. |y − x| < δ' ⟶ g y > 0
  proof clarify
    fix δ' y :: real
    assume δ' ≤ δ and |y − x| < δ'
    thus g y > 0 by (auto simp: H)
  qed
  then show ?thesis
    using δ-pos by blast
qed
```

1.5.2 Local Negativity

```
lemma cont-at-neg-imp-loc-neg:
  fixes g :: real ⇒ real and x :: real
  assumes continuous (at x) g and g x < 0
  shows ∃δ > 0. ∀y. |y − x| < δ ⟶ g y < 0
proof −
  from assms obtain δ where δ-pos: δ > 0
    and ∀y. |y − x| < δ ⟶ |g y − g x| < −(g x)/2
    by (metis continuous-at-eps-delta half-gt-zero neg-0-less-iff-less)
  then have ∀y. |y − x| < δ ⟶ − g y > 0
    by (smt (verit, best) field-sum-of-halves)
  then show ?thesis
    using δ-pos neg-0-less-iff-less by blast
qed

lemma cont-at-neg-imp-loc-neg':
  fixes g :: real ⇒ real and x :: real
  assumes continuous (at x) g and g x < 0
  shows ∃Δ > 0. ∀δ. 0 < δ ∧ δ ≤ Δ ⟶ (∀y. |y − x| < δ ⟶ g y < 0)
proof −
  from assms obtain δ where δ-pos: δ > 0
    and H: ∀y. |y − x| < δ ⟶ −(g y) > 0
    by (smt (verit) cont-at-neg-imp-loc-neg)
  have ∀δ' ≤ δ. ∀y. |y − x| < δ' ⟶ −(g y) > 0
  proof clarify
    fix δ' y :: real
    assume δ' ≤ δ and |y − x| < δ'
    then show −(g y) > 0
      using H by auto
  qed
  then show ?thesis
    using δ-pos neg-0-less-iff-less by blast
qed

end
```

2 Minimizers in Topological and Metric Spaces

```
theory Minimizers-Definition
  imports Auxilary-Facts
begin
```

2.1 Abstract Topological Definitions

```
definition global-minimizer :: ('a::topological-space ⇒ real) ⇒ 'a ⇒ bool where
  global-minimizer f x-star ⟷ (∀x. f x-star ≤ f x)

definition local-minimizer-on :: ('a::topological-space ⇒ real) ⇒ 'a ⇒ 'a set ⇒
```

```
bool where
  local-minimizer-on f x-star U ⟷ (open U ∧ x-star ∈ U ∧ (∀ x ∈ U. f x-star ≤
f x))

definition local-minimizer :: ('a::topological-space ⇒ real) ⇒ 'a ⇒ bool where
  local-minimizer f x-star ⟷ (∃ U. open U ∧ x-star ∈ U ∧ (∀ x ∈ U. f x-star ≤
f x))

definition isolated-local-minimizer-on :: ('a::topological-space ⇒ real) ⇒ 'a ⇒ 'a
set ⇒ bool where
  isolated-local-minimizer-on f x-star U ⟷
    (local-minimizer-on f x-star U ∧ ({x ∈ U. local-minimizer f x} = {x-star}))

definition isolated-local-minimizer :: ('a::topological-space ⇒ real) ⇒ 'a ⇒ bool
where
  isolated-local-minimizer f x-star ⟷
    (∃ U. local-minimizer-on f x-star U ∧ ({x ∈ U. local-minimizer f x} = {x-star}))

definition strict-local-minimizer-on :: ('a::topological-space ⇒ real) ⇒ 'a ⇒ 'a set
⇒ bool where
  strict-local-minimizer-on f x-star U ⟷
    (open U ∧ x-star ∈ U ∧ (∀ x ∈ U − {x-star}. f x-star < f x))

definition strict-local-minimizer :: ('a::topological-space ⇒ real) ⇒ 'a ⇒ bool
where
  strict-local-minimizer f x-star ⟷ (∃ U. strict-local-minimizer-on f x-star U)
```

2.2 Metric Space Reformulations

```
lemma local-minimizer-on-def2:
  fixes f :: 'a::metric-space ⇒ real
  assumes local-minimizer f x-star
  shows ∃ N > 0. ∀ x ∈ ball x-star N. f x-star ≤ f x
proof −
  from assms obtain U where
    open U x-star ∈ U and local-min: ∀ x ∈ U. f x-star ≤ f x
    unfolding local-minimizer-def by auto
  then obtain N where N-pos: N > 0 and ball-in-U: ball x-star N ⊆ U
    using open-contains-ball by blast
  hence ∀ x ∈ ball x-star N. f x-star ≤ f x
    using ball-in-U local-min by auto
  thus ?thesis
    using N-pos by auto
qed

lemma local-minimizer-def2:
  fixes f :: 'a::metric-space ⇒ real
  assumes local-minimizer f x-star
  shows ∃ N > 0. ∀ x. dist x x-star < N ⟶ f x-star ≤ f x
```

```
proof −
  from assms obtain U where
    open U x-star ∈ U and local-min: ∀ x ∈ U. f x-star ≤ f x
    unfolding local-minimizer-def by auto
  then obtain N where N-pos: N > 0 and ball-in-U: ball x-star N ⊆ U
    using open-contains-ball by blast
  hence ∀ x. dist x x-star < N ⟶ x ∈ ball x-star N
    by (subst mem-ball, simp add: dist-commute)
  hence ∀ x. dist x x-star < N ⟶ f x-star ≤ f x
    using ball-in-U local-min by blast
  thus ?thesis
    using N-pos by auto
qed

lemma isolated-local-minimizer-on-def2:
  fixes f :: 'a::metric-space ⇒ real
  assumes isolated-local-minimizer-on f x-star U
  shows ∃ N > 0. ∀ x ∈ ball x-star N. (local-minimizer f x ⟶ x = x-star)
proof −
  from assms have
    local-minimizer-on f x-star U
    and unique-min: {x ∈ U. local-minimizer f x} = {x-star}
    unfolding isolated-local-minimizer-on-def by auto
  then obtain N where N-pos: N > 0 and ball-in-U: ball x-star N ⊆ U
    using open-contains-ball by (metis local-minimizer-on-def)
  have ∀ x ∈ ball x-star N. local-minimizer f x ⟶ x = x-star
  proof(clarify)
    fix x
    assume x ∈ ball x-star N
    then have x ∈ U using ball-in-U by auto
    moreover assume local-minimizer f x
    hence x ∈ {x ∈ U. local-minimizer f x} using ‹x ∈ U› by auto
    hence x ∈ {x-star} using unique-min by auto
    ultimately show x = x-star
      by simp
  qed
  thus ?thesis using N-pos by auto
qed

lemma isolated-local-minimizer-def2:
  fixes f :: 'a::metric-space ⇒ real
  assumes isolated-local-minimizer f x-star
  shows ∃ N > 0. ∀ x ∈ ball x-star N. (local-minimizer f x ⟶ x = x-star)
proof −
  from assms obtain U where
    local-minimizer-on f x-star U
    and unique-min: {x ∈ U. local-minimizer f x} = {x-star}
    unfolding isolated-local-minimizer-def by auto
  then obtain N where N-pos: N > 0 and ball-in-U: ball x-star N ⊆ U
```

```
    using open-contains-ball by (metis local-minimizer-on-def)
  have ∀ x ∈ ball x-star N. local-minimizer f x ⟶ x = x-star
  proof(clarify)
    fix x
    assume x ∈ ball x-star N
    then have x ∈ U using ball-in-U by auto
    moreover assume local-minimizer f x
    hence x ∈ {x ∈ U. local-minimizer f x} using ‹x ∈ U› by auto
    hence x ∈ {x-star} using unique-min by auto
    ultimately show x = x-star by simp
  qed
  thus ?thesis using N-pos by auto
qed

lemma strict-local-minimizer-on-def2:
  fixes f :: 'a::metric-space ⇒ real
  assumes strict-local-minimizer-on f x-star U
  shows ∃ N > 0. ∀ x ∈ ball x-star N − {x-star}. f x-star < f x
proof −
  from assms have
    open U x-star ∈ U and strict-min: ∀ x ∈ U − {x-star}. f x-star < f x
    unfolding strict-local-minimizer-on-def by auto
  then obtain N where N-pos: N > 0 and ball-in-U: ball x-star N ⊆ U
    using open-contains-ball by metis
  have ∀ x ∈ ball x-star N − {x-star}. f x-star < f x
  proof
    fix x
    assume x ∈ ball x-star N − {x-star}
    hence x ∈ U − {x-star} using ball-in-U by auto
    thus f x-star < f x
      using strict-min by auto
  qed
  thus ?thesis using N-pos by auto
qed

lemma strict-local-minimizer-def2:
  fixes f :: 'a::metric-space ⇒ real
  assumes strict-local-minimizer f x-star
  shows ∃ N > 0. ∀ x ∈ ball x-star N − {x-star}. f x-star < f x
proof −
  from assms obtain U where
    strict-local-minimizer-on f x-star U
    unfolding strict-local-minimizer-def by auto
  then have
    open U x-star ∈ U and strict-min: ∀ x ∈ U − {x-star}. f x-star < f x
    unfolding strict-local-minimizer-on-def by auto
  then obtain N where N-pos: N > 0 and ball-in-U: ball x-star N ⊆ U
    using open-contains-ball by metis
  have ∀ x ∈ ball x-star N − {x-star}. f x-star < f x
```

```
  proof
    fix x
    assume x ∈ ball x-star N − {x-star}
    hence x ∈ U − {x-star} using ball-in-U by auto
    thus f x-star < f x
      using strict-min by auto
  qed
  thus ?thesis using N-pos by auto
qed

lemma local-minimizer-neighborhood:
  fixes f :: real ⇒ real
  assumes loc-min: local-minimizer f x-min
  shows ∃δ > 0. ∀h. |h| < δ ⟶ f (x-min + h) ≥ f x-min
proof −
  obtain N where N-pos: N > 0 and N-prop: ∀x. dist x x-min < N ⟶ f x-min
≤ f x
    using local-minimizer-def2[OF loc-min] by auto
  then have ∀h. abs h < N ⟶ f (x-min + h) ≥ f x-min
    by (simp add: dist-real-def)
  then show ?thesis
    using N-pos by blast
qed

lemma local-minimizer-from-neighborhood:
  fixes f :: real ⇒ real and x-min :: real
  assumes ∃δ > 0. ∀x. |x − x-min| < δ ⟶ f x-min ≤ f x
  shows local-minimizer f x-min
proof −
  from assms obtain δ where δ-pos: δ > 0 and H: ∀x. |x − x-min| < δ ⟶ f
x-min ≤ f x
    by auto
  obtain U where U-def: U = {x. |x − x-min| < δ}
    by simp
  then have open U
   by (smt (verit) dist-commute dist-real-def mem-Collect-eq metric-space-class.open-ball
subsetI topological-space-class.openI)
  moreover have x-min ∈ U
    using U-def δ-pos by force
  moreover have ∀x ∈ U. f x-min ≤ f x
    using H U-def by blast
  ultimately show ?thesis
    unfolding local-minimizer-def by auto
qed

end
```

3 Minimizer Implications

theory *First-Order-Conditions*
  imports *Minimizers-Definition*
begin

notation *norm* (‖-‖)

3.1 Implications for a Given Minimizer Type

lemma *strict-local-minimizer-imp-local-minimizer*:
  assumes *strict-local-minimizer f x-star*
  shows *local-minimizer f x-star*
  by (*smt* (*verit*) *Diff-iff assms local-minimizer-def singletonD strict-local-minimizer-def strict-local-minimizer-on-def*)

lemma *isolated-local-minimizer-imp-strict*:
  assumes *isolated-local-minimizer f x-star*
  shows *strict-local-minimizer f x-star*
proof −
  — From isolated_local_minimizer we obtain an open set U such that $x^\star$ is the *only* local minimizer.
  from *assms* **obtain** *U* **where** *iso-props*:
    isolated-local-minimizer-on f x-star U
    unfolding *isolated-local-minimizer-def*
    using *isolated-local-minimizer-on-def* **by** *blast*

  — Unpack `isolated_local_minimizer_on`: $x^\star$ is a `local_minimizer_on` U, and $x^\star$ is unique.

  from *iso-props* **have** *lm-on*: *local-minimizer-on f x-star U*
      unfolding *isolated-local-minimizer-on-def* **using** *local-minimizer-on-def* **by** *presburger*
  moreover from *iso-props* **have** *unique-min*: $\{x \in U.$ *local-minimizer f x*$\} = \{$*x-star*$\}$
    unfolding *isolated-local-minimizer-on-def* **by** *auto*

  — From `local_minimizer_on`, we have: U open, $x^\star \in U$, and $\forall x \in U.\ f(x^\star) \leq f(x)$.

  from *lm-on* **have** *open-U*: *open U* **and** *x-in-U*: *x-star* $\in$ *U* **and** *le-prop*: $\forall x \in U.$ *f x-star* $\leq$ *f x*
    unfolding *local-minimizer-on-def* **by** *auto*

  — Assume, for contradiction, that $x^\star$ is not a strict local minimizer. Then there exists $y \in U \setminus \{x^\star\}$ with $f(y) \leq f(x^\star)$.

  show *strict-local-minimizer f x-star*
  proof (*rule ccontr*)
    assume ¬ *strict-local-minimizer f x-star*

```
    then obtain y where y-props:
      y ∈ U − {x-star} and f y ≤ f x-star
      unfolding strict-local-minimizer-def strict-local-minimizer-on-def
      by (smt (verit, ccfv-SIG) open-U x-in-U)

    from y-props have y ∈ U and y ≠ x-star
      by auto
```

— We already have $f(x^\star) \leq f(y)$ from $\forall x \in U.\ f\ x\text{-}star \leq f\ x$ and $y \in U$. Together with $f(y) \leq f(x^\star)$, this yields $f(x^\star) = f(y)$.

```
    from le-prop ‹y ∈ U› have f x-star ≤ f y
      by auto
    with ‹f y ≤ f x-star› have f x-star = f y
      by auto
```

— Now we show that y is also a local minimizer, contradicting the uniqueness of $x^\star$. To prove this, we must exhibit an open set V around y such that $f(y) \leq f(x)$ for all $x \in V$.

```
    have local-minimizer f y
    proof −
```

— Since U is open and $y \in U$, there exists an open set $V \subseteq U$ containing y.

```
      obtain V where open V and y ∈ V and V ⊆ U
        using ‹open U› ‹y ∈ U› open-subset by auto
```

— On this subset, $f(y) = f(x^\star) \leq f(x)$ for all $x \in V$ (since $V \subseteq U$).

```
      moreover from le-prop and ‹f x-star = f y› have ∀ x ∈ V. f y ≤ f x
        using calculation(3) by auto

      ultimately show local-minimizer f y
        unfolding local-minimizer-def local-minimizer-on-def by auto
    qed
```

— Since y is a local minimizer and $y \in U$, we have $y \in \{x \in U.$ `local_minimizer` $f\ x\}$. By uniqueness, $\{x \in U.$ `local_minimizer` $f\ x\} = \{x^\star\}$, hence $y = x^\star$, contradicting $y \neq x^\star$.

```
    hence y ∈ {x ∈ U. local-minimizer f x}
      by (simp add: ‹y ∈ U›)
    with unique-min have y = x-star by auto
    thus False using ‹y ≠ x-star› by contradiction
  qed
```

— Having reached a contradiction under the assumption that $x^\star$ is not a strict local minimizer, it follows that $x^\star$ must indeed be a strict local minimizer.

```
qed
```

3.2 Characterization of Non-Isolated Minimizers

lemma *not-isolated-minimizer-def*:
  assumes *local-minimizer f x-star*
  shows (∃ *x-seq*::*nat* ⇒ *real*. (∀ *n*. *local-minimizer f* (*x-seq n*) ∧ *x-seq n* ≠ *x-star*) ∧ ((*x-seq* ⟶ *x-star*) *at-top*)) = (¬ *isolated-local-minimizer f x-star*)
proof(*safe*)
  show ⋀*x-seq*. *isolated-local-minimizer f x-star* ⟹ ∀ *n*. *local-minimizer f* (*x-seq n*) ∧ *x-seq n* ≠ *x-star* ⟹ *x-seq* ⟶ *x-star* ⟹ *False*
  proof −
    fix *x-seq* :: *nat* ⇒ *real*
    assume *x-star-isolated-minimizer*: *isolated-local-minimizer f x-star*
    assume *with-sequence-of-local-miniziers*: ∀ *n*. *local-minimizer f* (*x-seq n*) ∧ *x-seq n* ≠ *x-star*
    assume *converging-to-x-star*: *x-seq* ⟶ *x-star*
    have *open-ball-with-unique-min*: ∃ *N* > *0*. ∀ *x* ∈ *ball x-star N*. (*local-minimizer f x* ⟶ *x* = *x-star*)
      by (*simp add*: *isolated-local-minimizer-def2 x-star-isolated-minimizer*)
      then obtain *N* **where** *N-pos*: *N* > *0* **and** *N-prop*: ∀ *x* ∈ *ball x-star N*. (*local-minimizer f x* ⟶ *x* = *x-star*)
      by *blast*
    — Use convergence to show x_{seq} eventually lies in ball$(x^\star, N)$.
    from *converging-to-x-star* **have** ∃ *M*. ∀ *n* ≥ *M*. *x-seq n* ∈ *ball x-star N*
      by (*metis LIMSEQ-iff-nz N-pos dist-commute mem-ball*)
    then obtain *M* **where** *M-def*: ∀ *n* ≥ *M*. *x-seq n* ∈ *ball x-star N*
      by *auto*
    then show *False*
     by (*meson N-prop linorder-not-le order-less-irrefl with-sequence-of-local-miniziers*)
  qed
next
  show ¬ *isolated-local-minimizer f x-star* ⟹ ∃ *x-seq*. (∀ *n*. *local-minimizer f* (*x-seq n*) ∧ *x-seq n* ≠ *x-star*) ∧ *x-seq* ⟶ *x-star*
  proof(*rule ccontr*)
    assume *not-isolated-minimizer*: ¬*isolated-local-minimizer f x-star*
    assume *BWOC*: ∄ *x-seq*. (∀ *n*. *local-minimizer f* (*x-seq n*) ∧ *x-seq n* ≠ *x-star*) ∧ *x-seq* ⟶ *x-star*

    have ∃ *N* > *0*. ∀ *x*. *dist x x-star* < *N* ⟶ *f x-star* ≤ *f x*
      by (*simp add*: *assms local-minimizer-def2*)
     then obtain *N* **where** *N-pos*: (*N*::*nat*) > *0* **and** *x-star-min-on-N-ball*: ∀ *x*. *dist x x-star* < *1*/ *real N* ⟶ *f x-star* ≤ *f x*
      by (*metis dual-order.strict-trans ex-inverse-of-nat-less inverse-eq-divide*)

    obtain *S-n* :: *nat* ⇒ *real set* **where** *S-n-def*: *S-n* = (λ*n*. {*x*. *dist x x-star* < *1* / (*real n* + *N*) ∧ *x* ≠ *x-star* ∧ *local-minimizer f x*})
      by *blast*

    from *not-isolated-minimizer*
    have *non-isolated*: ∀ *U*. *local-minimizer-on f x-star U* ⟶ (∃ *y* ∈ *U*. *y* ≠ *x-star* ∧ *local-minimizer f y*)

```
      by (smt (verit, best) Collect-cong assms isolated-local-minimizer-def local-minimizer-on-def singleton-conv2)

  have ∀ n::nat. ∃ x. x ∈ S-n n
  proof (intro allI)
    fix n::nat
    have pos-radius: 1 / (real n + N) > 0
      using N-pos by simp

     obtain U where U-def: U = ball x-star (1 / (real n + N)) and open-U: open U and U-contains-x-star: x-star ∈ U
      using pos-radius by auto

    have U-contained-in-Inverse-N-Ball: ∀ x ∈ U. dist x x-star < 1 / N
    proof(safe)
      fix x:: real
      assume x-in-U: x ∈ U
      then have dist x x-star < (1 / (real n + N))
        by (simp add: U-def dist-commute)
      also have ... ≤ 1 / real N
        by (simp add: N-pos frac-le)
      finally show dist x x-star < 1 / real N.
    qed

    have ball-non-empty: ∃ y ∈ U. y ≠ x-star ∧ local-minimizer f y
    proof −
      have local-minimizer-on f x-star U
     by (simp add: U-contains-x-star U-contained-in-Inverse-N-Ball local-minimizer-on-def open-U x-star-min-on-N-ball)
      then show ∃ y∈U. y ≠ x-star ∧ local-minimizer f y
        by (simp add: non-isolated)
    qed
    then obtain y where y-in-ball: y ∈ U and y ≠ x-star and local-minimizer f y
      by blast
    then show ∃ x. x ∈ S-n n
      by (smt (verit, best) S-n-def U-def dist-commute mem-Collect-eq mem-ball)
  qed
  then obtain x-seq where x-seq-def: ∀ n. x-seq n ∈ S-n n
    by metis
  have x-seq-converges-to-x-star: x-seq ⟶ x-star
  proof (rule LIMSEQ-I)
    fix r :: real
    assume r-pos: 0 < r
    obtain n-min where n-min-def: 1 / (real n-min + N) < r
      using real-arch-inverse N-pos r-pos
     by (smt (verit, ccfv-SIG) frac-le inverse-eq-divide inverse-positive-iff-positive)
    show ∃ no. ∀ n≥no. norm (x-seq n − x-star) < r
    proof (intro exI allI impI)
```

```
        fix n
        assume n ≥ n-min
        then have n-large-enough: 1 / (real n + N) ≤ 1 / (real n-min + N)
          using N-pos by (subst frac-le, simp-all)
        have dist (x-seq n) x-star < 1 / (real n + N)
          using x-seq-def S-n-def by auto
        also have ... ≤ 1 / (real n-min + N)
          using n-large-enough by auto
        also have ... < r
          using n-min-def by auto
        finally show norm (x-seq n − x-star) < r
          by (simp add: dist-real-def)
      qed
    qed
     have ∃x-seq. (∀n. local-minimizer f (x-seq n) ∧ x-seq n ≠ x-star) ∧ x-seq
⟶ x-star
      using S-n-def x-seq-converges-to-x-star x-seq-def by blast
    then show False
      using BWOC by auto
  qed
qed
```

3.3 First-Order Condition

```
theorem Fermat's-theorem-on-stationary-points:
  fixes f :: real ⇒ real
  assumes (f has-derivative f') (at x-min)
  assumes local-minimizer f x-min
  shows (deriv f) x-min = 0
  by (metis assms has-derivative-imp differential-zero-maxmin local-minimizer-def)

definition stand-basis-vector :: 'n::finite ⇒ real^'n — the i-th standard basis
vector
  where stand-basis-vector i = (χ j. if j = i then 1 else 0)

lemma stand-basis-vector-index[simp]: (stand-basis-vector i) $ j = (if j = i then
(1::real) else 0)
  by (simp add: stand-basis-vector-def)

lemma stand-basis-vector-nonzero[simp]: stand-basis-vector i ≠ 0
  by (smt (verit, del-insts) stand-basis-vector-index zero-index)

lemma norm-stand-basis-vector[simp]: norm (stand-basis-vector i) = 1
  by (smt (verit, best) axis-nth component-le-norm-cart norm-axis-1 norm-le-componentwise-cart
real-norm-def stand-basis-vector-index)

lemma inner-stand-basis-vector[simp]: inner (stand-basis-vector i) (stand-basis-vector
j) = (if i = j then 1 else 0)
  by (metis axis-nth cart-eq-inner-axis norm-eq-1 norm-stand-basis-vector stand-basis-vector-index
```

```
vector-eq)

lemma Basis-characterisation:
  stand-basis-vector i ∈ (Basis :: (real^'n) set)   and
  ∀ b∈(Basis::(real^'n)set). ∃ i. b = stand-basis-vector i
  by (metis (no-types, lifting) Basis-real-def axis-in-Basis-iff cart-eq-inner-axis
    inner-stand-basis-vector insert-iff norm-axis-1 norm-eq-1 stand-basis-vector-index
vector-eq,
    metis axis-index axis-nth cart-eq-inner-axis inner-stand-basis-vector stand-basis-vector-index
vector-eq)

lemma stand-basis-expansion:
  fixes x :: real^'n
  shows x = (∑ j∈UNIV. (x $ j) *R stand-basis-vector j)
proof −
  have (∑ j∈UNIV. (x $ j) *R stand-basis-vector j) $ k = x $ k for k
  proof −
    have (∑ j∈UNIV. (x $ j) *R stand-basis-vector j) $ k
        = (∑ j∈UNIV. (x $ j) ∗ (stand-basis-vector j $ k))
      by simp
    also have ... = (∑ j∈UNIV. (x $ j) ∗ (if j = k then 1 else 0))
      by (smt (verit, best) stand-basis-vector-index sum.cong)
    also have ... = (∑ j∈UNIV. (if j = k then x $ j else 0))
      by (smt (verit, best) mult-cancel-left1 mult-cancel-right1 sum.cong)
    also have ... = x $ k
      by (subst sum.delta, simp-all)
    finally show ?thesis.
  qed
  thus ?thesis
    by (simp add: vec-eq-iff)
qed

lemma has-derivative-affine:
  fixes a v :: 'a::real-normed-vector
  shows ((λt. a + t *R v) has-derivative (λh. h *R v)) (at x)
  unfolding has-derivative-def
proof safe
  have a + y *R v − (a + netlimit (at x) *R v) − (y − netlimit (at x)) *R v = 0
if y ≠ netlimit (at x) for y
    by (simp add: cross3-simps(32))
  then show (λy. (a + y *R v − (a + netlimit (at x) *R v) − (y − netlimit (at
x)) *R v) /R ‖y − netlimit (at x)‖) −x→ 0
    by (simp add: scaleR-left-diff-distrib)
  show bounded-linear (λh. h *R v)
    by (simp add: bounded-linearI' vector-space-assms(2))
qed

theorem Fermat's-theorem-on-stationary-points-mult:
  fixes f :: real ^ 'n ⇒ real
```

```
  assumes der-f: (f has-derivative f′) (at x-min)
  assumes min-f: local-minimizer f x-min
  shows GDERIV f x-min :> 0
proof −
 — Show that f′ kills every standard-basis vector.

  {
    fix i :: ′n
    — Define the 1D slice g_i(t) = f(x_min + t · e_i).
    let ?g = λt::real. f (x-min + t *R stand-basis-vector i)

    — Chain rule gives g_i′(0) = f′(e_i).
    from has-derivative-affine have g-der:
      ((λt. f (x-min + t *R stand-basis-vector i))
      has-derivative (λh. f′ (h *R stand-basis-vector i))) (at 0)
     by (metis (no-types) arithmetic-simps(50) der-f has-derivative-compose scaleR-simps(1))

    — 0 is a local minimizer of g_i because x_min is one for f.
    have g-min: local-minimizer ?g 0
    proof(rule local-minimizer-from-neighborhood)
      obtain δ where δ-pos: δ > 0
        and mono: ⋀x. dist x-min x < δ ⟹ f x ≥ f x-min
        by (metis assms(2) dist-commute local-minimizer-def2)

      have ∀x. |x − 0| < δ ⟶ f (x-min + 0 *R stand-basis-vector i) ≤ f (x-min
+ x *R stand-basis-vector i)
        using mono by (simp add: dist-norm)
      then show ∃δ>0. ∀x. |x − 0| < δ ⟶ f (x-min + 0 *R stand-basis-vector
i) ≤ f (x-min + x *R stand-basis-vector i)
        using δ-pos by blast
    qed

    — Apply the 1-D Fermat lemma to g_i.
    from Fermat′s-theorem-on-stationary-points
    have f′ (stand-basis-vector i) = 0
      using g-der g-min  by (metis has-derivative-imp scale-one)
  }

    — Collecting the result for every i:
  hence zero-on-basis: ⋀i. f′ (stand-basis-vector i) = 0.

  — Use linearity and the coordinate expansion to show f′ = 0 everywhere.
  {
    fix v :: real^′n
    — Expand v = ∑_j v_j · e_j and push f′ through the finite sum.
    have f′ v = 0
    proof −
      have f′ v = f′ (∑j∈UNIV. (v $ j) *R stand-basis-vector j)
        by (metis stand-basis-expansion)
```

```
      also have ... = (∑j∈UNIV. (v $ j) *R f' (stand-basis-vector j))
      by (smt (verit) assms differential-zero-maxmin local-minimizer-def scale-eq-0-iff
sum.neutral)
      also have ... = 0
        using zero-on-basis by simp
      finally show ?thesis.
    qed
  }
  hence f'-zero: f' = (λ-. 0)
    by (simp add: fun-eq-iff)

  — Translate f′ = 0 into the gradient statement.
  have (f has-derivative (λh. 0)) (at x-min)
    using der-f f'-zero by simp
  hence GDERIV f x-min :> (0::real^'n)
    by (simp add: gderiv-def)
  thus ?thesis.
qed

end
```

4 Second-Order Conditions

```
theory Second-Derivative-Test
  imports First-Order-Conditions
begin
```

4.1 Necessary Condition

```
lemma snd-derivative-nonneg-at-local-min-necessary:
  fixes f :: real ⇒ real
  assumes C2-cont-diff-at-xmin: C-k-on 2 f (U :: real set)
  assumes min-in-U: (x-min :: real) ∈ U
  assumes loc-min:   local-minimizer f x-min
  shows deriv (deriv f) x-min ≥ 0
proof −
  have (∃ ε. 0 < ε ∧ {x-min − ε .. x-min + ε} ⊂ U)
  proof −
    have (∃ ε. 0 < ε ∧ ball x-min ε ⊂ U)
     by (smt C2-cont-diff-at-xmin C-k-on-def assms(2) ball-subset-cball cball-eq-ball-iff

          open-contains-cball-eq order-le-less-trans psubsetI)
    then show ?thesis
        by (metis Elementary-Metric-Spaces.open-ball cball-eq-atLeastAtMost centre-in-ball
          open-contains-cball order-trans-rules(21))
  qed
  then obtain ε where ε-pos: 0 < ε and ε-def: {x-min − ε .. x-min + ε} ⊂ U
    by blast
```

```
  have f-diff:    (∀ y ∈ U. (f has-real-derivative (deriv f) y) (at y))
    using C2-cont-diff C2-cont-diff-at-xmin by blast
  have f'-diff: (∀ y ∈ U. (deriv f has-real-derivative (deriv (deriv f)) y) (at y))
    using C2-cont-diff C2-cont-diff-at-xmin by blast
  have f''-contin: continuous-on U (deriv (deriv f))
    using C2-cont-diff assms(1) by blast

  have f'-0: (deriv f) x-min = 0
    using Fermat's-theorem-on-stationary-points
    by (meson assms(2,3) f-diff has-field-derivative-imp-has-derivative)

 — By local minimality at x_min, there is a δ > 0 such that for all h with |h| < δ,
we have f(x_min + h) ≥ f(x_min).
  obtain δ where δ-pos: δ > 0 and δ-prop: ∀ h. |h| < δ ⟶ f (x-min + h) ≥ f
x-min
    by (meson assms(3) local-minimizer-neighborhood)

  from f'-0 have second-deriv-limit-at-x-min:
    ((λh. (deriv f (x-min + h)) / h) ⟶ deriv (deriv f) x-min) (at 0)
    by (smt (verit, best) DERIV-def Lim-cong-within assms(2) f'-diff)

  show ?thesis
  proof(rule ccontr)
    assume ¬ 0 ≤ deriv (deriv f) x-min
    then have BWOC: 0 > deriv (deriv f) x-min
      by auto
    then obtain Δ where Δ-pos: Δ > 0 and
      Δ-def: ∀ δ. 0 < δ ∧ δ ≤ Δ ⟶ (∀ y. |y − x-min| < δ ⟶ deriv (deriv f) y
< 0)
     by (metis C2-cont-diff-at-xmin C-k-on-def min-in-U at-within-open cont-at-neg-imp-loc-neg'
           continuous-on-eq-continuous-within f''-contin)

    — Choose h with 0 < h < min{δ, Δ} so that x_min + h ∈ U.
    obtain h where h-def: h = min ε (min (δ/2) Δ) and h-pos: 0 < h
      using ε-pos δ-pos Δ-pos by fastforce
    have h-lt: h ≤ ε ∧ h < δ ∧ h ≤ Δ
      using δ-pos h-def by linarith
    have neigh-in-U: x-min + h ∈ {x-min − ε .. x-min + ε}
      using h-def h-pos by fastforce

    have f (x-min + h) < f x-min
    proof(rule DERIV-neg-imp-decreasing-open[where a = x-min and f = f and
b = x-min + h])
      show x-min < x-min + h
        using h-pos by simp
    next
      have {x-min..x-min + h} ⊂ U
        using ε-def dual-order.strict-trans2 neigh-in-U by auto
      then show continuous-on {x-min..x-min + h} f
```

```
        by (meson C2-cont-diff C2-cont-diff-at-xmin continuous-on-subset
            differentiable-imp-continuous-on le-less)
    next
      show ⋀x. ⟦x-min < x; x < x-min + h⟧ ⟹ ∃ y. (f has-real-derivative y) (at
x) ∧ y < 0
      proof −
        fix x :: real
        assume x-min-lt-x: x-min < x
        assume x-lt-xmin-pls-h: x < x-min + h

        have xmin-x-subset: {x-min .. x} ⊆ {x-min − ε .. x-min + ε}
            using neigh-in-U x-lt-xmin-pls-h by auto

        — By the Mean Value Theorem applied to f′ on [x_min, x], there exists some
c with x_min < c < x such that:
         have ∃ z > x-min. z < x ∧ deriv f (x) −  deriv f x-min = (x − x-min) ∗
deriv(deriv f) z
        proof(rule MVT2)
          show x-min < x
            using x-min-lt-x by auto
        next
          fix y :: real
          assume x-min-leq-y: x-min ≤ y
          assume y-leq-x: y ≤ x

          from xmin-x-subset have y ∈ U
            using ε-def atLeastAtMost-iff x-min-leq-y y-leq-x by blast
          then show (deriv f has-real-derivative deriv (deriv f) y) (at y)
            using f′-diff by blast
        qed
        then obtain z where z-gt-x-min: z > x-min and
                            z-lt-x: z < x and
                             z-def: deriv f (x) −  deriv f x-min = (x − x-min) ∗ deriv
(deriv f) z
          by blast
        then have mvt-f′: deriv f (x)  = (x − x-min) ∗ deriv (deriv f) z
          by (simp add: f′-0)

        then have x-diff-xmin-pos: x − x-min > 0
           using ‹x-min < x› by simp
        then have left-bound-satisfied: |z − x-min| < x − x-min
           using ‹x-min < z› ‹z < x› by auto
        then have x − x-min < h
           using ‹x < x-min + h› by simp
        then  have  |z − x-min| < h
          using left-bound-satisfied by fastforce
        then have deriv (deriv f) z < 0
           using Δ-def h-lt h-pos by blast
        then  have deriv f x < 0
```

```
          by (metis x-diff-xmin-pos mvt-f′ mult-pos-neg)
         moreover have x ∈ U
           using xmin-x-subset
           by (meson ε-def atLeastAtMost-iff dual-order.strict-iff-not
               subset-eq verit-comp-simplify(2) x-min-lt-x)
         ultimately show ∃ y. (f has-real-derivative y) (at x) ∧ y < 0
           using f-diff by blast
       qed
     qed
    then show False
      by (smt (verit, best) δ-prop h-lt h-pos)
  qed
qed
```

4.2 Sufficient Condition

```
lemma second-derivative-test:
  fixes f :: real ⇒ real and a :: real and b :: real and x-min :: real
  assumes valid-interval: a < b
  assumes twice-continuously-differentiable: C-k-on 2 f {a <..< b}
  assumes min-exists: x-min ∈ {a <..< b}
  assumes fst-deriv-req: (deriv f) x-min = 0
  assumes snd-deriv-req: deriv (deriv f) x-min > 0
  shows loc-min: local-minimizer f x-min
proof −
  from twice-continuously-differentiable
  have f′′-cont: continuous-on {a <..< b} (deriv (deriv f))
    by (metis C-k-on-def Suc-1 lessI nat.simps(2) second-derivative-alt-def)
  then obtain Δ where Δ-pos: Δ > 0
    and Δ-prop: ∀ δ. 0 < δ ∧ δ ≤ Δ ⟶ (∀ y. |y − x-min| < δ ⟶ deriv (deriv f)
y > 0)
   by (metis assms(3,5) at-within-open cont-at-pos-imp-loc-pos′ continuous-on-eq-continuous-within

        open-real-greaterThanLessThan)

  obtain δ where δ-min: δ = min Δ (min ((x-min − a) / 2) ((b − x-min) / 2))
    by blast

  have δ-pos: δ > 0
  proof (cases δ = Δ)
    show δ = Δ ⟹ 0 < δ
      by (simp add: Δ-pos)
  next
    assume δ ≠ Δ
    then have δ = min ((x-min − a) / 2) ((b − x-min) / 2)
      using δ-min by linarith
    then show 0 < δ
      using min-exists by force
  qed
```

```
have neigh-of-x-min-contained-in-ab: a < x-min − δ ∧ x-min + δ < b
  by (smt (z3) δ-min δ-pos field-sum-of-halves)

have local-min: ∀ x. |x − x-min| < δ ⟶ f x ≥ f x-min
proof clarify
  fix x
  assume A: |x − x-min| < δ
  consider (eq) x = x-min | (lt) x < x-min | (gt) x > x-min
    by linarith
  then show f x ≥ f x-min
  proof cases
    case eq
    then show ?thesis
      by simp
  next
    case lt
    have a-lt-x-and-xmin-lt-b: a < x ∧ x-min < b
      using A neigh-of-x-min-contained-in-ab by linarith
    have f x > f x-min
    proof (rule DERIV-neg-imp-decreasing-open[where a = x])
      show x < x-min
        by (simp add: lt)
    next
      fix y :: real
      assume x-lt-y: x < y
      assume y-lt-x-min: y < x-min
      — For x < x_min, apply the Mean Value Theorem to f on [x, x_min].
      have ∃ z > y. z < x-min ∧ deriv f x-min − deriv f y = (x-min − y) ∗ deriv
(deriv f) z
        proof (rule MVT2[where a = y and b = x-min and f = deriv f and f′
= deriv (deriv f)])
        show y < x-min
          by (simp add: y-lt-x-min)
      next
        fix z :: real
        assume y-lt-z: y ≤ z
        assume z-lt-x-min: z ≤ x-min
        show (deriv f has-real-derivative (deriv (deriv f)) z) (at z)
        proof (subst C2-cont-diff[where f = f, where U = {a <..< b}])
          show C-k-on 2 f {a<..<b}
            by (simp add: assms(2))
          show z ∈ {a<..<b} and True
            using a-lt-x-and-xmin-lt-b x-lt-y y-lt-z z-lt-x-min by auto
        qed
      qed
      then obtain z where
        z-props: y < z z < x-min and
        eq: deriv f x-min − deriv f y = (x-min − y) ∗ deriv (deriv f) z
```

```
          by blast
        have deriv f x-min = 0
          using fst-deriv-req by simp
        hence deriv f y = − (x-min − y) ∗ deriv (deriv f) z
          using eq by linarith
        moreover have x-min − x > 0
          using lt by simp
        have deriv (deriv f) z > 0
          by (smt (verit) A Δ-prop δ-min x-lt-y z-props)
        ultimately have deriv f y < 0
          by (simp add: mult-less-0-iff y-lt-x-min)
        then show ∃ z. (f has-real-derivative z) (at y) ∧ z < 0
        by (meson C2-cont-diff a-lt-x-and-xmin-lt-b assms(2) dual-order.strict-trans
                    greaterThanLessThan-iff x-lt-y y-lt-x-min)
      next
        have continuous-on {a <..< b} f
          by (simp add: C2-cont-diff assms(2) differentiable-imp-continuous-on)
        then show continuous-on {x..x-min} f
          by (smt (verit, del-insts) a-lt-x-and-xmin-lt-b atLeastAtMost-iff
                    continuous-on-subset greaterThanLessThan-iff subsetI)
      qed
      then show f x-min ≤ f x
        by simp
    next
      case gt
      have a-lt-xmin-and-x-lt-b: a < x-min ∧ x < b
        using A ‹a < x-min − δ ∧ x-min + δ < b› by linarith
      have f x > f x-min
      proof (rule DERIV-pos-imp-increasing-open[where a = x-min])
        show x-min < x
          by (simp add: gt)
      next
        fix y :: real
        assume y-gt-xmin: x-min < y
        assume y-lt-x: y < x
        — For x_min < y, apply the Mean Value Theorem to f′ on [x_min, y].
        have ∃ z > x-min. z < y ∧ deriv f y − deriv f x-min = (y − x-min) ∗ deriv
(deriv f) z
        proof (rule MVT2[where a = x-min and b = y and f = deriv f and f′
= deriv (deriv f)])
          show x-min < y
            by (simp add: y-gt-xmin)
        next
          fix z :: real
          assume z-ge-xmin: x-min ≤ z
          assume z-le-y: z ≤ y
          show (deriv f has-real-derivative (deriv (deriv f)) z) (at z)
          proof (subst C2-cont-diff[where f = f and U = {a<..<b}])
            show C-k-on 2 f {a<..<b}
```

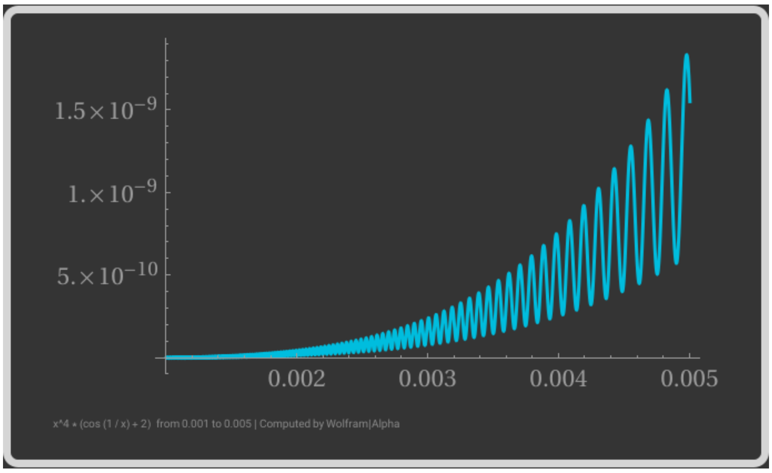
```
       by (simp add: assms(2))
      show z ∈ {a<..<b} and True
       using a-lt-xmin-and-x-lt-b y-lt-x z-ge-xmin z-le-y by auto
     qed
    qed
    then obtain z where
     z-props: x-min < z z < y
     and eq: deriv f y − deriv f x-min = (y − x-min) ∗ deriv (deriv f) z
     by blast
    have deriv f x-min = 0
     using fst-deriv-req by simp
    hence deriv f y = (y − x-min) ∗ deriv (deriv f) z
     using eq by simp
    moreover have y − x-min > 0
     using y-gt-xmin by simp
    moreover have deriv (deriv f) z > 0
     by (smt (verit, best) A Δ-prop δ-min y-lt-x z-props(1,2))
    ultimately have deriv f y > 0
     by auto
    then show ∃ d. (f has-real-derivative d) (at y) ∧ d > 0
    by (meson C2-cont-diff a-lt-xmin-and-x-lt-b assms(2) dual-order.strict-trans
          greaterThanLessThan-iff y-lt-x y-gt-xmin)
   next
    have continuous-on {a <..< b} f
     by (simp add: C2-cont-diff assms(2) differentiable-imp-continuous-on)
    then show continuous-on {x-min..x} f
     by (smt (verit, del-insts) a-lt-xmin-and-x-lt-b atLeastAtMost-iff
          continuous-on-subset greaterThanLessThan-iff subsetI)
   qed
   then show ?thesis
    by simp
  qed
 qed
 show ?thesis
  by (rule local-minimizer-from-neighborhood, smt δ-pos local-min)
qed

end
```

5 Pathological Example: Non-Isolated Strict Local Minima

```
theory Cont-Nonisolated-Strict-Local-Minimizer-Exists
 imports Second-Derivative-Test HOL−Library.Quadratic-Discriminant
begin
```

Idea of the example. We construct a continuous function

$$f(x) = \begin{cases} x^4\bigl(\cos(1/x) + 2\bigr), & x \neq 0, \\ 0, & x = 0 \end{cases}$$



x^4 * (cos (1 / x) + 2) from 0.001 to 0.005 | Computed by Wolfram|Alpha

whose oscillations *speed up* as $x \to 0$ because of the $\cos(1/x)$ term. Multiplying by x^4 makes the function and its first derivative vanish at the origin, ensuring that $x = 0$ is a strict local minimizer, while the shifted cosine creates infinitely many additional strict local minimizers that accumulate at 0. Hence the minimizer at 0 is *strict* but *not isolated*.

```
theorem Exists-Continuous-Func-with-non-isolated-strict-local-minimizer:
  ∃ f::real ⇒ real. continuous-on ℝ f ∧
    (∃ x-star. strict-local-minimizer f x-star ∧ ¬ isolated-local-minimizer f x-star)
proof –
  obtain f where f-def: f = (λ(x::real). if x ≠ 0 then x^4 * (cos (1 / x) + 2)
else 0)
    by simp

  have deriv-f: ⋀x::real. deriv f x = (if x = 0 then 0 else x² * sin (1 / x) +
                                    4 * x^3 * cos (1 / x) + 8 * x^3)
                     ∧ (λx. f x) differentiable-on UNIV
                     ∧ deriv (deriv f) x = (if x = 0 then 0 else 6*x * sin (1 / x) +
                                    (12*x² − 1)* cos (1 / x) + 24*x²)
                     ∧ (deriv f) differentiable-on UNIV
  proof (safe)
    — First we compute the derivative away from 0, then we compute it at 0.
    have deriv-f-at-nonzero:
     ⋀x. x ≠ 0 ⟶ deriv f x = (x² * sin (1 / x) + 4*x^3 * cos (1 / x) + 8*x^3)
          ∧ f field-differentiable at x
    proof (safe)
      fix x :: real
```

```
assume x-type: x ≠ 0

have cos-inverse-diff: (λw. cos (1 / w)) field-differentiable at x
proof -
  have f1: (λw. 1 / w) field-differentiable at x
    by (simp add: field-differentiable-divide x-type)
  have (λz. cos z) field-differentiable at (1 / x)
    by (simp add: field-differentiable-within-cos)
  then show ?thesis
    by (metis DERIV-chain2 f1 field-differentiable-def)
qed
then have (λx. cos (1 / x) + 2) field-differentiable at x
  by (simp add: Derivative.field-differentiable-add)
then have f2: (λx. x^4 * (cos (1 / x) + 2)) field-differentiable at x
     by (subst field-differentiable-mult, simp add: field-differentiable-power, simp-all)

have deriv-2nd-part: deriv (λw. (λx. cos (1 / x) + 2) w) x = (sin (1 / x)) / x^2
proof -
  have deriv (λw. (λx. cos (1 / x) + 2) w) x =
        (deriv (λw. (λx. cos (1 / x)) w) x + deriv (λw. (λx. 2) w) x)
    by (rule deriv-add, simp add: cos-inverse-diff, simp)
  also have ... = (sin (1 / x)) / x^2
  proof -
    have f1: DERIV (λz. cos z) (1 / x) :> −sin (1 / x)
      by simp
    have f2: DERIV (λw. 1 / w) x :> −1 / x^2
      using DERIV-inverse-func x-type by blast
    from f1 f2 have DERIV ((λz. cos z) ∘ (λw. 1 / w)) x :> (−sin (1 / x)) * (−1 / x^2)
        by (rule DERIV-chain)
    then show ?thesis
      by (simp add: DERIV-imp-deriv o-def)
  qed
  finally show ?thesis.
qed

show deriv f x = x^2 * sin (1 / x) + 4*x^3 * cos (1 / x) + 8*x^3
proof -
  have deriv f x = deriv (λx. x^4 * (cos (1 / x) + 2)) x
    by (metis (no-types, lifting) f-def mult-eq-0-iff power-zero-numeral)
  also have ... = x^4 * deriv (λx. cos (1 / x) + 2) x +
                  deriv (λx. x^4) x * (cos (1 / x) + 2)
    by (rule deriv-mult, simp add: field-differentiable-power,
        simp add: Derivative.field-differentiable-add cos-inverse-diff)
  also have ... = x^4 * (sin (1 / x)) / x^2 +
                  deriv (λx. x^4) x * (cos (1 / x) + 2)
    by (simp add: deriv-2nd-part)
```

```
        also have ... = x^4 * (sin (1 / x)) / x² + (4*x^3) * (cos (1 / x) + 2)
          by (subst power-rule, simp)
        also have ... = x² * (sin (1 / x)) + (4*x^3) * (cos (1 / x) + 2)
          by (simp add: power2-eq-square power4-eq-xxxx)
        also have ... = x² * sin (1 / x) + 4*x^3 * cos (1 / x) + 8*x^3
          by (simp add: Rings.ring-distribs(2) mult.commute)
        finally show ?thesis.
      qed
      from x-type f-def f2 show f field-differentiable at x
        by (subst field-differentiable-transfer-on-ball[where f = λ x. (x^4 * (cos (1
/ x) + 2))
              and ε = |x|], simp-all)
    qed

    have deriv-f-at-0: deriv f 0 = 0 ∧ f field-differentiable at 0
    proof –
    — By the definition of deriv, we need to show the limit of the difference quotient
is 0.
      have dq-limit: ((λh. (f (0 + h) − f 0) / h) ⟶ 0) (at 0)
      proof
        fix ε :: real
        assume ε-pos: 0 < ε
        — Choose δ > 0 to make |difference quotient| < ε.
        obtain δ where δ-def: δ = (ε / 3) powr (1 / 3)
          by simp
        — A reasonable δ based on the growth of |h³|.
        have δ-pos: δ > 0
          using ε-pos by (simp add: δ-def)
        have ∃δ>0. ∀h. 0 < |h| ∧ |h| < δ ⟶ |(f (0 + h) − f 0) / h − 0| < ε
        proof (intro exI[where x=δ], intro conjI insert δ-pos, clarify)
          fix h :: real
          assume h-pos: 0 < |h|
          assume h-lt-δ: |h| < δ

          have |(f (0 + h) − f 0) / h − 0| = |f h / h|
            by (simp add: f-def)
          also have ... = |h^4 * (cos (1 / h) + 2) / h|
            using f-def by presburger
          also have ... = |h^3 * (cos (1 / h) + 2)|
        by (simp add: power3-eq-cube power4-eq-xxxx vector-space-over-itself.scale-scale)
          also have ... ≤ |h^3| * |cos (1 / h) + 2|
            by (metis abs-mult order.refl)
          also have ... ≤ |h^3| * (|cos (1 / h)| + |2|)
            by (simp add: mult-left-mono)
          also have ... ≤ |h^3| * (1 + 2)
            by (simp add: mult-left-mono)
          also have ... = 3 * |h^3|
            by simp
          also have ... < 3 * δ^3
```

```
          using power-strict-mono[of |h| δ 3] by (simp add: h-lt-δ power-abs)
        also have ... = 3 ∗ (ε / 3)
          by (metis δ-def ε-pos div-self less-le more-arith-simps(5)
                 mult-eq-0-iff pos-le-divide-eq powr-numeral powr-one-gt-zero-iff
                      powr-powr times-divide-eq-left verit-comp-simplify(19)
zero-neq-numeral)
        also have ... = ε
          by simp
        finally show |(f (0 + h) − f 0) / h − 0| < ε.
      qed
      then show ∃ d>0.∀ x∈UNIV. 0 < dist x 0 ∧ dist x 0 < d ⟶ dist ((f (0
+ x) − f 0) / x) 0 ≤ ε
        by (metis arithmetic-simps(57) dist-real-def less-le)
    qed
    then show ?thesis
      using DERIV-def DERIV-imp-deriv field-differentiable-def by blast
  qed

  show deriv-f: ⋀x. deriv f x =
    (if x = 0 then 0 else x^2 ∗ sin (1 / x) + 4 ∗ x^3 ∗ cos (1 / x) + 8 ∗ x^3)
    using deriv-f-at-0 deriv-f-at-nonzero by presburger

  show f-is-differentiable: (λx. f x) differentiable-on UNIV
    by (metis deriv-f-at-0 deriv-f-at-nonzero differentiable-on-def
        field-differentiable-imp-differentiable)

  have snd-deriv-f-at-nonzero:
    ⋀x. x ≠ 0 ⟶ deriv (deriv f) x = (6∗x ∗ sin (1 / x) + (12∗x^2 − 1)∗ cos
(1 / x) + 24∗x^2)
         ∧ (deriv f) field-differentiable at x
  proof (safe)
    fix x :: real
    assume x-type: x ≠ 0

    have fst-term-diff: (λw. w^2 ∗ sin (1 / w)) field-differentiable at x
    proof −
      have f1: (λw. w^2) field-differentiable at x
        by (simp add: field-differentiable-power)
      have (λw. sin (1 / w)) field-differentiable at x
        by (metis DERIV-chain2 DERIV-inverse-func field-differentiable-at-sin
              field-differentiable-def x-type)
      then show ?thesis
        by (simp add: f1 field-differentiable-mult)
    qed

    have fst-term-deriv: deriv (λw. w^2 ∗ sin (1 / w)) x = 2 ∗ x ∗ sin (1 / x)
− cos (1 / x)
    proof −
      have deriv (λx. x^2 ∗ sin (1 / x)) x =
```

```
          x^2 ∗ deriv (λx. sin (1 / x)) x + deriv (λx. x^2) x ∗ sin (1 / x)
        by (rule deriv-mult, simp add: field-differentiable-power,
          metis DERIV-chain2 DERIV-inverse-func field-differentiable-at-sin
             field-differentiable-def x-type)
      moreover have deriv (λx. x^2) x = 2 ∗ x
        using power-rule by auto
      moreover have deriv (λx. sin (1 / x)) x = −cos (1 / x) / x^2
      proof −
        have f1: DERIV (λz. sin z) (1 / x) :> cos (1 / x)
          by simp
        have f2: DERIV (λx. 1 / x) x :> −1 / x^2
          using DERIV-inverse-func x-type by blast
         from f1 f2 have DERIV ((λz. sin z) ∘ (λx. 1 / x)) x :> cos (1 / x) ∗
(−1 / x^2)
          by (rule DERIV-chain)
        then show ?thesis
          by (simp add: DERIV-imp-deriv o-def)
      qed
      ultimately show ?thesis
        by (simp add: x-type)
    qed

    have snd-term-diff: (λx. 4 ∗ x^3 ∗ cos (1 / x)) field-differentiable at x
    proof −
      have t1: (λx. 4 ∗ x^3) field-differentiable at x
        by (simp add: field-differentiable-power field-differentiable-mult)
      have t2: (λx. cos (1 / x)) field-differentiable at x
        by (metis DERIV-chain2 DERIV-inverse-func field-differentiable-at-cos
             field-differentiable-def x-type)
      show ?thesis
        by (simp add: t1 t2 field-differentiable-mult)
    qed
   have snd-term-diff': (λw. 4 ∗ w ^ 3 ∗ cos (1 / w) + 8 ∗ w ^ 3) field-differentiable
at x
    proof −
      have t3: (λx. 8 ∗ x^3) field-differentiable at x
        by (simp add: field-differentiable-mult field-differentiable-power)
      show ?thesis
        by (simp add: Derivative.field-differentiable-add t3 snd-term-diff)
    qed

    have snd-term-deriv:
      deriv (λx. 4 ∗ x^3 ∗ cos (1 / x) + 8 ∗ x^3) x =
       12 ∗ x^2 ∗ cos (1 / x) + 4 ∗ x ∗ sin (1 / x) + 24 ∗ x^2
    proof −
      have deriv (λx. 4 ∗ x^3 ∗ cos (1 / x) + 8 ∗ x^3) x =
          deriv (λx. 4 ∗ x^3 ∗ cos (1 / x)) x + deriv (λx. 8 ∗ x^3) x
        by (rule deriv-add, simp add: snd-term-diff,
          simp add: field-differentiable-mult field-differentiable-power)
```

```
    also have ... = (4∗x^3) ∗ (deriv (λx. cos (1 / x)) x) +
                 ((12 ∗ x^2) ∗ (cos (1 / x))) + deriv (λx. 8 ∗ x^3) x
    proof −
      have deriv (λx. 4 ∗ x^3 ∗ cos (1 / x)) x =
            (4∗x^3) ∗ (deriv (λx. cos (1 / x)) x) +
            (deriv (λx. 4 ∗ x^3) x) ∗ (cos (1 / x))
      by (rule deriv-mult, simp add: field-differentiable-mult field-differentiable-power,
                metis DERIV-fun-cos DERIV-inverse-func field-differentiable-def
x-type)
      then have deriv (λx. 4 ∗ x^3) x = 12 ∗ x^2
      proof −
        have deriv (λx. 4 ∗ x^3) x = 4 ∗ deriv (λx. x^3) x
          by (rule deriv-cmult, simp add: field-differentiable-power)
        then show ?thesis
          by (simp add: power-rule)
      qed
      then show ?thesis
        using ‹deriv (λx. 4 ∗ x^3 ∗ cos (1 / x)) x = (4∗x^3) ∗ (deriv (λx. cos
(1 / x)) x) +
                                       (deriv (λx. 4 ∗ x^3) x) ∗ (cos (1 / x))›
        by auto
    qed
    also have ... = (4∗x^3) ∗ (deriv (λx. cos (1 / x)) x) +
                 ((12 ∗ x^2) ∗ (cos (1 / x))) + 24 ∗ x^2
    proof −
      have deriv (λx. 8 ∗ x^3) x = 24 ∗ x^2
      proof −
        have deriv (λx. 8 ∗ x^3) x = 8 ∗ deriv (λx. x^3) x
          by (rule deriv-cmult, simp add: field-differentiable-power)
        then show ?thesis
          by (simp add: power-rule)
      qed
      then show ?thesis
        by auto
    qed
    also have ... = (4∗x^3) ∗ sin (1 / x) / x^2 + ((12 ∗ x^2) ∗ (cos (1 / x)))
+ 24 ∗ x^2
    proof −
      have deriv (λx. cos (1 / x)) x = sin (1 / x) / x^2
      proof −
        have f1: DERIV (λz. cos z) (1 / x) :> −sin (1 / x)
          by simp
        have f2: DERIV (λx. 1 / x) x :> −1 / x^2
          using DERIV-inverse-func x-type by blast
       from f1 f2 have DERIV ((λz. cos z) ∘ (λx. 1 / x)) x :> (−sin (1 / x))
∗ (−1 / x^2)
          by (rule DERIV-chain)
        then show ?thesis
          by (simp add: DERIV-imp-deriv o-def)
```

```
      qed
      then show ?thesis
        by auto
     qed
     also have ... = ((12 * x^2) * (cos (1 / x))) + (4*x^3) * sin (1 / x) / x^2
+ 24 * x^2
      by linarith
     also have ... = (12 * x^2) * (cos (1 / x)) + 4*x * sin (1 / x) + 24 * x^2
     proof -
      have (4*x^3) * sin (1 / x) / x^2 = 4*x * sin (1 / x)
        by (simp add: power2-eq-square power3-eq-cube)
      then show ?thesis
        by presburger
     qed
     finally show ?thesis.
    qed

    show deriv (deriv f) x = (6*x * sin (1 / x) + (12*x² − 1)* cos (1 / x) +
24*x²)
    proof -
     have deriv (deriv f) x = deriv (λx. x² * sin (1 / x) + 4 * x^3 * cos (1 /
x) + 8 * x^3) x
     by (metis (no-types, opaque-lifting) deriv-f mult-cancel-left2 mult-cancel-right2

          power-zero-numeral pth-7(2))
     also have ... = deriv (λx. x² * sin (1 / x) + (4 * x^3 * cos (1 / x) + 8 *
x^3)) x
      by (meson Groups.add-ac(1))
     also have ... = deriv (λx. x^2 * sin (1 / x)) x +
                     deriv (λx. 4 * x^3 * cos (1 / x) + 8 * x^3) x
      by (rule deriv-add, simp add: fst-term-diff, simp add: snd-term-diff')
     also have ... = 2 * x * sin (1 / x) − cos (1 / x) +
                      deriv (λx. 4 * x^3 * cos (1 / x) + 8 * x^3) x
      by (simp add: fst-term-deriv)
     also have ... = 2 * x * sin (1 / x) − cos (1 / x) +
                    12 * x^2 * cos (1 / x) + 4 * x * sin (1 / x) + 24 * x^2
      by (simp add: snd-term-deriv)
     also have ... = 2 * x * sin (1 / x) + 4 * x * sin (1 / x) +
                      12 * x^2 * cos (1 / x) − cos (1 / x) + 24 * x^2
      by simp
     also have ... = (6*x * sin (1 / x) + (12*x² − 1)* cos (1 / x) + 24*x²)
      by (smt (verit, best) cos-add cos-zero mult-diff-mult sin-zero)
     finally show ?thesis.
    qed

    show (deriv f) field-differentiable at x
    proof (rule field-differentiable-transfer-on-ball
        [where f = λ x. (x² * sin (1 / x) + 4 * x^3 * cos (1 / x) + 8 * x^3)
and ε = |x|])
```

```
        show  0 < |x|
          by (simp add: x-type)
        show ∀ y. y ∈ ball x |x| ⟶ y² ∗ sin (1 / y) + 4 ∗ y ^ 3 ∗ cos (1 / y) +
8 ∗ y ^ 3 =
          deriv f y
          by (simp add: deriv-f)
           show (λx. x² ∗ sin (1 / x) + 4 ∗ x ^ 3 ∗ cos (1 / x) + 8 ∗ x ^
3)field-differentiable at x
       by (simp add: Derivative.field-differentiable-add fst-term-diff is-num-normalize(1)

              snd-term-diff ')
      qed
    qed

    have deriv2-f-at-0:
      deriv (deriv f) 0 = 0 ∧ (deriv f) field-differentiable at 0
    proof −
        — By the definition of deriv, we need to show the limit of the difference
quotient of f′ is 0.
      have dq-limit: ((λh. (deriv f (0 + h) − deriv f 0) / h) ⟶ 0) (at 0)
      proof
        fix ε :: real
        assume ε-pos: 0 < ε
         have ∃δ>0. ∀ h. 0 < |h| ∧ |h| < δ ⟶ |(deriv f (0 + h) − deriv f 0) / h
− 0| < ε
        proof (cases ε < 1/6)
          assume eps-lt-inv6: ε < 1/6
          — Choose δ > 0 to ensure |difference quotient| < ε.
          obtain δ where δ-def: δ = ε / 2
            by blast
          have δ-pos: δ > 0
            using ε-pos by (simp add: δ-def)
          show ∃δ>0. ∀ h. 0 < |h| ∧ |h| < δ ⟶ |(deriv f (0 + h) − deriv f 0) /
h − 0| < ε
          proof (intro exI[where x=δ], intro conjI insert δ-pos, clarify)
            fix h :: real
            assume h-pos: 0 < |h|
            assume h-lt-δ: |h| < δ

            have h-bound1: |h| < ε / 2
              using h-lt-δ by (simp add: δ-def)
            have h-bound2: 12 ∗ |h^2| < ε / 2
            proof −
              have |h| < ε / 2 using h-bound1 by blast
              then have |h^2| < (ε / 2)^2
                     by (metis abs-ge-zero abs-power2 power2-abs power-strict-mono
zero-less-numeral)
              then have 12 ∗ |h^2| < 12 ∗ (ε / 2)^2
                by (simp add: mult-strict-left-mono)
```

```
              also have ... = 12 ∗ (ε^2 / 4)
                by (simp add: power2-eq-square)
              also have ... = 3 ∗ ε^2
                by simp
              also have ... < ε/2
              proof −
                have ε ∗ 6 < 1
                  by (meson eps-lt-inv6 less-divide-eq-numeral1(1))
                then show ?thesis
                  by (simp add: ε-pos power2-eq-square)
              qed
              finally show ?thesis.
            qed
            have |(deriv f (0 + h) − deriv f 0) / h − 0| = |deriv f h / h|
              by (simp add: deriv-f-at-0)
            also have ... = |(h² ∗ sin (1 / h) + 4∗h^3 ∗ cos (1 / h) + 8∗h^3) / h|
              using deriv-f by presburger
            also have ... = |(h² ∗ sin (1 / h) / h) + (4∗h^3 ∗ cos (1 / h)) / h +
(8∗h^3) / h|
              by (simp add: add-divide-distrib)
            also have ... = |h ∗ sin (1 / h) + (4∗h^2 ∗ cos (1 / h)) + 8 ∗ h^2|
              by (simp add: more-arith-simps(11) power2-eq-square power3-eq-cube)
            also have ... ≤ |h ∗ sin (1 / h)| + |4∗h^2 ∗ cos (1 / h)| + |8 ∗ h^2|
              by linarith
            also have ... ≤ |h| ∗ |sin (1 / h)| + 4 ∗ |h^2| ∗ |cos (1 / h)| + 8 ∗ |h^2|
              by (simp add: abs-mult)
            also have ... ≤ |h| + 4 ∗ |h^2| + 8 ∗ |h^2|
            proof −
              have i1: |h| ∗ |sin (1 / h)| ≤ |h|
                using h-pos by fastforce
              have |h| ∗ |cos (1 / h)| ≤ |h|
                by (simp add: mult-left-le)
              then show ?thesis
                by (smt (verit) cos-ge-minus-one cos-le-one i1 mult-left-le)
            qed
            also have ... = |h| + 12 ∗ |h^2|
              by simp
            also have ... < ε
              using h-bound1 h-bound2 by auto
            finally show |(deriv f (0 + h) − deriv f 0) / h − 0| < ε.
          qed
        next
          assume ¬ ε < 1/6
          then have ε ≥ 1/6 by linarith
          then have eps-half: ε / 2 ≥ 1/12 by linarith
          obtain δ where δ-def: δ = (1::real)/12 by blast
          have δ-pos: δ > 0 using ε-pos by (simp add: δ-def)
          show ∃δ>0. ∀h. 0 < |h| ∧ |h| < δ ⟶ |(deriv f (0 + h) − deriv f 0) /
h − 0| < ε
```

```
proof (intro exI[where x=δ], intro conjI insert δ-pos, clarify)
  fix h :: real
  assume h-pos: 0 < |h|
  assume h-lt-δ: |h| < δ
  have h-bound1: |h| < ε / 2
  proof −
    have |h| < δ using h-lt-δ by blast
    also have ... = (1::real)/12 by (simp add: δ-def)
    also have ... ≤ ε / 2 using eps-half by blast
    finally show ?thesis.
  qed
  have h-bound2: 12 ∗ |h|^2 < ε / 2
  proof −
    from h-bound1 have |h|^2 < (1/12)^2
     by (metis δ-def abs-ge-zero h-lt-δ power-strict-mono zero-less-numeral)
    hence 12 ∗ |h|^2 < 12 ∗ (1/12)^2
      by (rule mult-strict-left-mono, simp-all)
    also have ... = 1/12 by (simp add: power-one-over)
    also have ... ≤ ε / 2 using eps-half by blast
    finally show ?thesis.
  qed
  have |(deriv f (0 + h) − deriv f 0) / h − 0| = |deriv f h / h|
    by (simp add: deriv-f-at-0)
 also have ... = |(h^2 ∗ sin (1 / h) + 4∗h^3 ∗ cos (1 / h) + 8∗h^3) / h|
    using deriv-f by presburger
  also have ... = |(h^2 ∗ sin (1 / h) / h) + (4∗h^3 ∗ cos (1 / h)) / h +
(8∗h^3) / h|
    by (simp add: add-divide-distrib)
  also have ... = |h ∗ sin (1 / h) + (4∗h^2 ∗ cos (1 / h)) + 8 ∗ h^2|
    by (simp add: more-arith-simps(11) power2-eq-square power3-eq-cube)
  also have ... ≤ |h ∗ sin (1 / h)| + |4∗h^2 ∗ cos (1 / h)| + |8 ∗ h^2|
    by linarith
 also have ... ≤ |h| ∗ |sin (1 / h)| + 4 ∗ |h^2| ∗ |cos (1 / h)| + 8 ∗ |h^2|
    by (simp add: abs-mult)
  also have ... ≤ |h| + 4 ∗ |h^2| + 8 ∗ |h^2|
  proof −
    have i1: |h| ∗ |sin (1 / h)| ≤ |h|
      using h-pos by fastforce
    have |h| ∗ |cos (1 / h)| ≤ |h|
      by (simp add: mult-left-le)
    then show ?thesis
      by (smt (verit) cos-ge-minus-one cos-le-one i1 mult-left-le)
  qed
  also have ... = |h| + 12 ∗ |h^2|
    by simp
  also have ... < ε
    using h-bound1 h-bound2 by auto
  finally show |(deriv f (0 + h) − deriv f 0) / h − 0| < ε.
qed
```

```
      qed
      then show ∃ d>0. ∀ x∈UNIV. 0 < dist x 0 ∧ dist x 0 < d ⟶
                        dist ((deriv f (0 + x) − deriv f 0) / x) 0 ≤ ε
        by (metis cancel-comm-monoid-add-class.diff-zero dist-real-def le-less)
    qed
    then show ?thesis
      using DERIV-def DERIV-imp-deriv field-differentiable-def by blast
  qed

  show ⋀x. deriv (deriv f) x = (if x = 0 then 0 else 6 ∗ x ∗ sin (1 / x)
                                   + (12 ∗ x^2 − 1) ∗ cos (1 / x)
                                   + 24 ∗ x^2)
    using snd-deriv-f-at-nonzero deriv2-f-at-0 by presburger

  show (deriv f) differentiable-on UNIV
    by (metis deriv2-f-at-0 differentiable-on-def
            field-differentiable-imp-differentiable snd-deriv-f-at-nonzero)
qed
then have f-cont: continuous-on ℝ f
 by (meson continuous-on-subset differentiable-imp-continuous-on top.extremum)
have f′-cont: continuous-on ℝ (deriv f)
 by (meson continuous-on-subset deriv-f differentiable-imp-continuous-on top.extremum)

obtain U where U-def: U = {x :: real. −1 < x ∧ x < 1}
  by blast
then have open-neighborhood-of-zero: open U ∧ 0 ∈ U
  using lemma-interval-lt by (subst open-dist, subst dist-real-def,fastforce)

have strict-local-minimizer-at-0: strict-local-minimizer f 0
  unfolding strict-local-minimizer-def  strict-local-minimizer-on-def
proof (intro exI[where x=U],(subst sym[OF conj-assoc],rule conjI), rule open-neighborhood-of-zero)
  show ∀ x ∈ U − {0}. f 0 < f x
  proof
    fix x
    assume x-type: x ∈ U − {0}
    then have x-nonzero: x ≠ 0
      by blast
    have cos(1/x) + 2 ≥ 1
      by (smt (verit) cos-ge-minus-one)
    then have x^4 ∗ (cos(1/x) + 2) ≥ x^4 ∗ 1
      by (rule mult-left-mono, force)
    then have f x ≥ x^4
      by (simp add: f-def x-nonzero)
    then have f x > 0
    by (smt (verit, del-insts) mult-le-0-iff power4-eq-xxxx x-nonzero zero-le-mult-iff)
    then show f 0 < f x
      using f-def by force
  qed
qed
```

```
then have zero-min: local-minimizer f 0
  by (simp add: strict-local-minimizer-imp-local-minimizer)
have (∃ x-seq::nat ⇒ real. (∀ n. local-minimizer f (x-seq n) ∧ x-seq n ≠ 0) ∧
                                        ((x-seq ⟶ 0) at-top))
proof −
  obtain left-seq :: nat ⇒ real where left-seq-def: ∀ n ∈ ℕ. n ≠ 0 ⟶
        left-seq n = inverse ((5 ∗ pi / 4) + 2 ∗ real n ∗ pi)
    by force
  obtain right-seq :: nat ⇒ real where right-seq-def: ∀ n ∈ ℕ. n ≠ 0 ⟶
        right-seq n = inverse  (pi + 2 ∗ real n ∗ pi)
    by force

  have zero-lt-left-seq-lt-right-seq-both-pos: ∀ n ∈ ℕ. n ≠ 0 ⟶
                                        0 < left-seq n ∧ left-seq n < right-seq n
  proof clarify
    fix n::nat
    assume n-pos: 0 < n
    then have inv-left: inverse (left-seq n) = (5 ∗ pi / 4) + 2 ∗ real n ∗ pi
      by (metis bot-nat-0.not-eq-extremum id-apply inverse-inverse-eq left-seq-def
of-nat-eq-id
          of-nat-in-Nats)

    have inv-right: inverse (right-seq n) = pi + 2 ∗ real n ∗ pi
         by (metis bot-nat-0.not-eq-extremum id-apply inverse-inverse-eq n-pos
of-nat-eq-id
          of-nat-in-Nats right-seq-def)

    have denom-ineq: (pi + 2 ∗ real n ∗ pi) < ((5 ∗ pi / 4) + 2 ∗ real n ∗ pi)
    proof −
      have (5 ∗ pi / 4) + 2 ∗ real n ∗ pi = 2 ∗ real n ∗ pi + (5 ∗ pi / 4)
        by simp
      have ((5 ∗ pi / 4) + 2 ∗ real n ∗ pi) − (pi + 2 ∗ real n ∗ pi) =
              (5 ∗ pi / 4) + 2 ∗ real n ∗ pi − pi − 2 ∗ real n ∗ pi
        by simp
      also have ... = (5 ∗ pi / 4) − pi
        by simp
      also have ... = (5 ∗ pi / 4) − (4 ∗ pi / 4)
        by simp
      also have ... = (5 − 4) ∗ pi / 4
        by simp
      also have ... = pi / 4
        by simp
      then show ?thesis
        by simp
    qed
    then have left-seq n < right-seq n
     by (smt (verit) inv-left inv-right inverse-positive-iff-positive le-imp-inverse-le

          mult-nonneg-nonneg of-nat-less-0-iff pi-gt3)
```

```
      then show 0 < left-seq n ∧ left-seq n < right-seq n
      by (smt (verit, best) denom-ineq inv-left inverse-positive-iff-positive mult-nonneg-nonneg

         of-nat-less-0-iff pi-gt3)
    qed
    have first-and-second-order-conditions: ∀ n. n ≠ 0 ⟶
  (∃ y ∈ {left-seq n .. right-seq n}. (y^2 ∗ sin (1 / y) + 4 ∗ y^3 ∗ cos (1 / y) +
8 ∗ y^3) = 0 ∧
  (6∗y ∗ sin (1 / y) + (12∗y² − 1)∗ cos (1 / y) + 24∗y²) > 0) ∧
  ((left-seq n)^2 ∗ sin (1 /(left-seq n)) + 4 ∗ (left-seq n)^3 ∗ cos (1 / (left-seq n))
+
                              8 ∗ (left-seq n)^3) < 0 ∧
  ((right-seq n)^2 ∗ sin (1 /(right-seq n)) + 4 ∗ (right-seq n)^3 ∗ cos (1 / (right-seq
n))
                             + 8 ∗ (right-seq n)^3) > 0
    proof(clarify)
      fix n:: nat
      assume n-pos: 0 < n
      then have n-ge-1: 1 ≤ n
        by simp
      show (∃ y∈{left-seq n..right-seq n}. y² ∗ sin (1 / y) + 4 ∗ y ^ 3 ∗ cos (1 /
y) + 8 ∗ y ^ 3 = 0 ∧ 0 < 6 ∗ y ∗ sin (1 / y) + (12 ∗ y² − 1) ∗ cos (1 / y) +
24 ∗ y²) ∧
           (left-seq n)² ∗ sin (1 / left-seq n) + 4 ∗ left-seq n ^ 3 ∗ cos (1 / left-seq
n) + 8 ∗ left-seq n ^ 3 < 0 ∧
             0 < (right-seq n)² ∗ sin (1 / right-seq n) + 4 ∗ right-seq n ^ 3 ∗ cos
(1 / right-seq n) + 8 ∗ right-seq n ^ 3
      proof safe
        show left-seq-less-zero: (λx. x^2 ∗ sin (1 / x) + 4 ∗ x^3 ∗ cos (1 / x) + 8
∗ x^3) (left-seq n) < 0
        proof −
          obtain x where x-def: x = left-seq n
            by blast
          — Rewrite 1/x in terms of 5π/4 + 2nπ.

          then have inv-x-eqs: inverse x = inverse (inverse ((5 ∗ pi / 4) + 2 ∗ real
n ∗ pi))
          by (metis bot-nat-0.not-eq-extremum id-apply left-seq-def n-pos of-nat-eq-id
of-nat-in-Nats)
          then have x-inv: 1/x = (5 ∗ pi / 4) + 2 ∗ real n ∗ pi
            by (simp add: inverse-eq-divide)

          — Evaluate sin(1/x) and cos(1/x).
          have sin-inv-x: sin (1 / x) = − (sqrt 2 / 2)
          proof −
            have sin (1 / x) = sin ((5 ∗ pi / 4) + 2 ∗ real n ∗ pi)
              using x-inv by presburger
            also have ... = sin (5 ∗ pi / 4)
              by (simp add: sin-add)
```

```
  also have ... = - (sqrt 2 / 2)
    using sin-5pi-div-4 by blast
  finally show sin (1 / x) = - (sqrt 2 / 2).
qed

have cos-inv-x: cos (1 / x) = - (sqrt 2 / 2)
proof -
  have cos-val: cos (1 / x) = cos ((5 * pi / 4) + 2 * real n * pi)
    using x-inv by presburger
  also have ... = cos (5 * pi / 4)
    by (simp add: cos-add)
  also have ... = - (sqrt 2 / 2)
    using cos-5pi-div-4 by linarith
  finally show cos (1 / x) = - (sqrt 2 / 2).
qed
```

— Substitute these into the expression.

```
have expr: x^2 * sin (1 / x) + 4 * x^3 * cos (1 / x) + 8 * x^3
              = - (sqrt 2 / 2) * x^2 + (8 - 2 * sqrt 2) * x^3
proof -
  have x^2 * sin (1 / x) + 4 * x^3 * cos (1 / x) + 8 * x^3
          = (x^2 * - (sqrt 2 / 2)) + 4 * x^3 * (-(sqrt 2 / 2)) + 8 * x^3
    by (simp add: cos-inv-x sin-inv-x)

  also have ... = x^2 * - (sqrt 2 / 2) + (-2 * sqrt 2) * x^3 + 8 * x^3
    by simp
  also have ... =  - (sqrt 2 / 2) * x^2 + (8 - 2 * sqrt 2) * x^3
  proof -
    have - (sqrt 2 / 2) + (x ^ 3 * (sqrt 2 * - 2) + x ^ 3 * 8) =
          - (sqrt 2 / 2) + x ^ 3 * (sqrt 2 * - 2 + 8)
      by (metis (no-types) nat-distrib(2))
    then show ?thesis
      by (simp add: Groups.mult-ac(2))
  qed
  finally show rewrite-expr:
    x^2 * sin (1 / x) + 4 * x^3 * cos (1 / x) + 8 * x^3
     = - (sqrt 2 / 2) * x^2 + (8 - 2 * sqrt 2) * x^3.
qed
```

— Factor out x^3, and rewrite x^3 as $\left(\frac{5\pi}{4} + 2n\pi\right)^{-1}$.

```
have deriv-right-seq-eval: sin (1 / x) * x^2 + 4 * x^3 * cos (1 / x) + 8
* x^3 =
      (- (sqrt 2 / 2)*((5 * pi / 4) + 2 * real n * pi) + (8 - 2 * sqrt 2))
* x^3
proof -
  have sin (1 / x) * x^2 + 4 * x^3 * cos (1 / x) + 8 * x^3 =
    - (sqrt 2 / 2)*inverse x * x^3 + (8 - 2 * sqrt 2) * x^3
      by (smt (verit, del-insts) Groups.mult-ac(2) cos-inv-x cos-zero di-
```

```
vide-eq-0-iff expr
                    left-inverse more-arith-simps(11) one-power2 power2-eq-square
power3-eq-cube
                power-minus sin-inv-x sin-zero)
      also have ... = (− (sqrt 2 / 2)∗inverse x + (8 − 2 ∗ sqrt 2)) ∗ x^3
        by (metis (no-types) distrib-right)
      also have ... = (− (sqrt 2 / 2)∗((5 ∗ pi / 4) + 2 ∗ real n ∗ pi) +
                                        (8 − 2 ∗ sqrt 2)) ∗ x^3
        by (simp add: inv-x-eqs)
      finally show ?thesis.
    qed

    — Combine into a single fraction and show negativity.
    have first-term-eval: x^3 > 0
   by (smt (verit) mult-nonneg-nonneg of-nat-0-le-iff pi-gt3 x-inv zero-compare-simps(7)

              zero-less-power)
    have neg-term: (−(sqrt 2 / 2)∗((5 ∗ pi / 4) + 2 ∗ real n ∗ pi) + (8 −
2 ∗ sqrt 2)) < 0
      proof −
        have n-ge1: n ≥ 1
          using n-ge-1 by auto
        have lower-bound: 2 ∗ real n ∗ pi ≥ 2 ∗ pi
          using n-ge1 by (simp add: mult-left-mono)
        then have mult-bound: − (sqrt 2 / 2) ∗ ((5 ∗ pi / 4) + 2 ∗ real n ∗
pi)
                              ≤ − (sqrt 2 / 2) ∗ (5 ∗ pi / 4 + 2 ∗ pi)
          by (simp add: mult-left-mono)
        moreover have (− (sqrt 2 / 2) ∗ (5 ∗ pi / 4 + 2 ∗ pi) + (8 − 2 ∗
sqrt 2)) < 0
        proof −
          have 5 ∗ pi / 4 + 2 ∗ pi = 13 ∗ pi / 4
            by simp
          then have simpification: (− (sqrt 2 / 2) ∗ (5 ∗ pi / 4 + 2 ∗ pi) +
(8 − 2 ∗ sqrt 2))
                  =    (64 − 16 ∗ sqrt 2 − 13 ∗ pi ∗ sqrt 2) / 8
            by (simp add: field-simps)
           have sufficies-to-show-numerator-neg:((64 − 16 ∗ sqrt 2 − 13 ∗ pi
∗ sqrt 2) / 8 < 0)
              = (64 − 16 ∗ sqrt 2 − 13 ∗ pi ∗ sqrt 2 < 0)
            by simp
          have sqrt 2 ∗ (16 + 13 ∗ pi) > 64
          proof −
            have pi-gt-3: pi > 3
              by (simp add: pi-gt3)
            hence 16 + 13 ∗ pi > 16 + 13 ∗ 3
              by (simp add: mult-strict-left-mono)
            hence 16 + 13 ∗ pi > 55
              by simp
```

```
                  then have sqrt 2 * (16 + 13 * pi) > sqrt 2 * 55
                    by (simp add: mult-strict-left-mono)
                  moreover have sqrt 2 * 55 > 64
                  proof -
                    have (sqrt 2 * 55)^2 = 2 * 55^2
                      by (simp add: power-mult-distrib)
                    also have ... = 2 * (55*55)
                      by auto
                    also have ... = 6050
                      by simp
                    also have ... > 64*64
                      by eval
                    moreover have sqrt 2 * 55 > 0
                      by simp
                    ultimately show sqrt 2 * 55 > 64
                      using power-mono-iff
                      by (metis less-le power2-eq-square zero-less-numeral)
                  qed
                  ultimately show ?thesis
                    by linarith
                qed
                then have 64 - 16 * sqrt 2 - 13 * pi * sqrt 2 < 0
                  by (simp add: Groups.mult-ac(2) distrib-left)
                then show ?thesis
                  using simpification sufficies-to-show-numerator-neg by presburger
              qed
              then show ?thesis
                using mult-bound by linarith
            qed
            then show (left-seq n)^2 * sin (1 / left-seq n) +
                  4 * left-seq n ^ 3 * cos (1 / left-seq n) + 8 * left-seq n ^ 3 < 0
              by (metis deriv-right-seq-eval first-term-eval mult.commute x-def
                  zero-compare-simps(10))
          qed
          show right-seq-greater-zero:(λx. x^2 * sin (1 / x) + 4 * x^3 * cos (1 /
x) + 8 * x^3)
                                                            (right-seq n) > 0
        proof -
          obtain x where x-def: x = right-seq n
            by blast
          then have inv-x-eqs: inverse x = inverse (inverse (pi + 2 * real n * pi))
               by (metis id-apply n-pos of-nat-eq-id of-nat-in-Nats of-nat-less-0-iff
right-seq-def)
          have x-inv: 1 / x = pi + 2 * real n * pi
                  unfolding right-seq-def by (metis inv-x-eqs inverse-eq-divide in-
verse-inverse-eq)

          have sin-inv-x: sin (1 / x) = 0
            by (metis add.inverse-neutral sin-2npi sin-periodic-pi2 x-inv)
```

```
        have cos-inv-x: cos (1 / x) = −1
          using cos-2npi cos-periodic-pi2 x-inv by presburger

        have f-x: x^2 ∗ sin (1 / x) + 4 ∗ x^3 ∗ cos (1 / x) + 8 ∗ x^3 = 4 ∗ x^3
          by (simp add: cos-inv-x sin-inv-x)

        have x-pos: x > 0
          unfolding right-seq-def
             by (smt (verit) mult-nonneg-nonneg of-nat-less-0-iff pi-gt-zero x-inv
zero-less-divide-iff)
        then show 0 < (right-seq n)² ∗ sin (1 / right-seq n) + 4 ∗ right-seq n ^
3 ∗ cos (1 / right-seq n) + 8 ∗ right-seq n ^ 3
          using cos-inv-x sin-inv-x x-def by fastforce
      qed

      show ∃y∈{left-seq n..right-seq n}. y² ∗ sin (1 / y) + 4 ∗ y ^ 3 ∗ cos (1 /
y) + 8 ∗ y ^ 3 =
                              0 ∧ 0 < 6 ∗ y ∗ sin (1 / y) + (12 ∗ y² − 1) ∗ cos (1 /
y) + 24 ∗ y²
      proof −
        have existence-of-minimizing-sequence: ∃y∈{left-seq n..right-seq n}. y² ∗
sin (1 / y) + 4 ∗ y ^ 3 ∗ cos (1 / y) + 8 ∗ y ^ 3 = 0
        proof −
          have ∃x≥left-seq n. x ≤ right-seq n ∧ (λx. x^2 ∗ sin (1 / x) + 4 ∗ x^3
∗ cos (1 / x) + 8 ∗ x^3) x = 0
          proof(rule IVT')
            show (left-seq n)² ∗ sin (1 / left-seq n) + 4 ∗ left-seq n ^ 3 ∗ cos (1 /
left-seq n) + 8 ∗ left-seq n ^ 3 ≤ 0
              using left-seq-less-zero by auto
            show 0 ≤ (right-seq n)² ∗ sin (1 / right-seq n) + 4 ∗ right-seq n ^ 3
∗ cos (1 / right-seq n) + 8 ∗ right-seq n ^ 3
              using right-seq-greater-zero by linarith
            show left-seq n ≤ right-seq n
             by (metis id-apply leD linorder-linear n-pos of-nat-eq-id of-nat-in-Nats
zero-lt-left-seq-lt-right-seq-both-pos)
            show continuous-on {left-seq n..right-seq n} (λx. x² ∗ sin (1 / x) + 4
∗ x ^ 3 ∗ cos (1 / x) + 8 ∗ x ^ 3)
            proof − — We prove continuity by establishing it is differentiable.
            — First, note that left_seq_n is positive, so the interval does not contain
0.
              have left-seq-pos: left-seq n > 0
                  by (metis bot-nat-0.extremum-strict id-apply n-pos of-nat-eq-id
of-nat-in-Nats zero-lt-left-seq-lt-right-seq-both-pos)

              — Transfer global differentiability to local differentiability of deriv f.

              have ⋀x. x ∈ {left-seq n..right-seq n} ⟶ (λx. x² ∗ sin (1 / x) + 4
∗ x ^ 3 ∗ cos (1 / x) + 8 ∗ x ^ 3) field-differentiable at x
```

```
               proof clarify
                 fix x::real
                 assume x-type: x ∈ {left-seq n..right-seq n}
                show (λx. x^2 ∗ sin (1 / x) + 4 ∗ x ^ 3 ∗ cos (1 / x) + 8 ∗ x ^ 3)
field-differentiable at x
                   proof(rule field-differentiable-transfer-on-ball[where f = deriv f
and ε = x])
                   show 0 < x
                     using left-seq-pos x-type by auto
                  show ∀ y. y ∈ ball x x ⟶ deriv f y = y^2 ∗ sin (1 / y) + 4 ∗ y ^
3 ∗ cos (1 / y) + 8 ∗ y ^ 3
                     by (simp add: deriv-f)
                   show deriv f field-differentiable at x
                by (meson UNIV-I deriv-f differentiable-on-def field-differentiable-def
real-differentiableE)
                 qed
               qed
               then have (λx. x^2 ∗ sin (1 / x) + 4 ∗ x ^ 3 ∗ cos (1 / x) + 8 ∗ x
^ 3) differentiable-on {left-seq n..right-seq n}
            by (meson differentiable-at-imp-differentiable-on field-differentiable-imp-differentiable)
               then show ?thesis
                 using differentiable-imp-continuous-on by blast
             qed
           qed
           then show ∃ y∈{left-seq n..right-seq n}. y^2 ∗ sin (1 / y) + 4 ∗ y ^ 3 ∗
cos (1 / y) + 8 ∗ y ^ 3 = 0
             by presburger
         qed
        then obtain min-n where min-n-def: min-n ∈{left-seq n..right-seq n} ∧
min-n^2 ∗ sin (1 / min-n) + 4 ∗ min-n ^ 3 ∗ cos (1 / min-n) + 8 ∗ min-n ^ 3 =
0
           by blast
          have ⋀y. y ∈ {left-seq n .. right-seq n} ⟶ 0 < 6 ∗ y ∗ sin (1 / y) +
(12 ∗ y^2 − 1) ∗ cos (1 / y) + 24 ∗ y^2
         proof (clarify)
           fix y :: real
           assume y-int: y ∈ {left-seq n .. right-seq n}
           — Since left_seq_n > 0, every y in the interval is positive.
           then have y-pos: y > 0
          by (metis atLeastAtMost-iff bot-nat-0.extremum id-apply linorder-not-less
n-pos
            of-nat-eq-id of-nat-in-Nats order-less-le-trans zero-lt-left-seq-lt-right-seq-both-pos)

            have ∃ x-nc :: real ⇒ real. ∀ c ∈ {0..pi/4}. x-nc c = inverse (pi + c
+ 2∗pi∗real n)
             by auto
           then obtain x-nc :: real ⇒ real where x-nc-def: ∀ c ∈ {0..pi/4}. x-nc
c = inverse (pi + c + 2∗pi∗real n)
             by auto
```

```
          have ∃ x-nc :: real ⇒ real. ∀ c ∈ {0..pi/4}. x-nc c = inverse (pi + c
+ 2∗pi∗real n)
           by auto
         then obtain x-nc :: real ⇒ real where x-nc-def: ∀ c ∈ {0..pi/4}. x-nc
c = inverse (pi + c + 2∗pi∗real n)
           by auto
        have continuous-on-subinterval: continuous-on {0..pi/4} x-nc
        proof −
          have cont-denom: continuous-on {0..pi/4} (λc. pi + c + 2∗pi∗real n)
          proof −
            have continuous-on {0..pi/4} (λc. c)
              using continuous-on-id by blast
            moreover have continuous-on {0..pi/4} (λc. pi + 2∗pi∗real n)
              using continuous-on-const by blast
            ultimately show ?thesis
              by (simp add: continuous-on-add)
          qed
              then have continuous-on {0..pi/4} (λx. inverse ((λc. pi + c +
2∗pi∗real n) x))
            by(rule continuous-on-inverse,
               smt (verit) add-mono-thms-linordered-field(4) atLeastAtMost-iff
               of-nat-less-0-iff pi-neq-zero pi-not-less-zero zero-compare-simps(4))

          then show ?thesis
            using continuous-on-cong x-nc-def by fastforce
        qed

        have minimizer-dom: ∃x. 0 ≤ x ∧ x ≤ pi/4 ∧ x-nc x = y
        proof(rule IVT2')
          show x-nc (pi / 4) ≤ y
          proof −
            have x-nc (pi / 4) = inverse ( pi + pi / 4 + 2 ∗ real n ∗ pi)
             by (metis (no-types, opaque-lifting) atLeastAtMost-iff divide-eq-imp
              divide-real-def linorder-not-less mult.left-commute mult.right-neutral

               mult-le-0-iff nle-le of-nat-0-le-iff of-nat-numeral pi-gt-zero x-nc-def

                 zero-neq-numeral)
            also have ... = inverse ((5 ∗ pi / 4) + 2 ∗ real n ∗ pi)
              by simp
            also have ... = left-seq n
                 by (metis bot-nat-0.not-eq-extremum id-apply left-seq-def n-pos
of-nat-eq-id of-nat-in-Nats)
            also have ... ≤ y
              using y-int by presburger
            finally show ?thesis.
          qed
          show y ≤ x-nc 0
          proof −
```

```
      have y ≤ right-seq n
        using y-int by presburger
      also have ... = inverse  (pi + 2 ∗ real n ∗ pi)
          by (metis bot-nat-0.not-eq-extremum id-apply n-pos of-nat-eq-id of-nat-in-Nats right-seq-def)
      also have ... = x-nc 0
        using x-nc-def by auto
      finally show ?thesis.
    qed
    show 0 ≤ pi / 4
      by simp
    show continuous-on {0..pi / 4} x-nc
      using continuous-on-subinterval by simp
  qed
  then have minimizer-dom': ∃ c ∈ {0..pi/4}. y = x-nc c
    using atLeastAtMost-iff by blast
```

— We will show that $f''(x_{nc}(c)) > 0$ for all $c \in [0, 1]$, then use the fact that $left\_seq_n \leq x_{nc}(c) \leq right\_seq_n$ together with the IVT to establish the existence of $c \in [0, \frac{\pi}{4}]$ such that $x_{nc}(c) = y$, and then conclude that $f''(y) > 0$.

```
    have snd-deriv-positive-in-neighborhood: ∀ c ∈ {0..pi/4}. left-seq n ≤ x-nc c ∧ x-nc c ≤ right-seq n ∧ deriv (deriv f) (x-nc c) > 0
  proof (safe)
    fix c :: real
    assume c-type: c ∈ {0..pi/4}
    then have c-bounds: 0 ≤ c ∧ c ≤ pi/4
      by simp

    have x-nc-eqs: x-nc c = inverse (pi + c + 2∗pi∗real n)
      using c-bounds inverse-eq-divide pi-half-le-two x-nc-def by auto
    show left-seq n ≤ x-nc c
    proof −
      have f1: left-seq n = inverse ((5 ∗ pi / 4) + 2 ∗ real n ∗ pi)
           by (metis bot-nat-0.not-eq-extremum id-apply left-seq-def n-pos of-nat-eq-id of-nat-in-Nats)
      from c-bounds have 1/ ((5 ∗ pi / 4) + 2 ∗ real n ∗ pi) ≤ 1/ (pi + c + 2∗pi∗real n)
        by(subst frac-le, simp-all, simp add: add-sign-intros(1))
      then show ?thesis
        by (simp add: f1 x-nc-eqs inverse-eq-divide)
    qed

    then have x-nc-pos: x-nc c > 0
     by (metis id-apply n-pos of-nat-eq-id of-nat-in-Nats order-less-le-trans zero-lt-left-seq-lt-right-seq-both-pos zero-order(5))

    show x-nc c ≤ right-seq n
    proof −
```

```
  have f1: right-seq n = inverse (pi + 2 * real n * pi)
       by (metis bot-nat-0.not-eq-extremum id-apply n-pos of-nat-eq-id of-nat-in-Nats right-seq-def)
  from c-bounds have 1/ (pi + c + 2*pi*real n) ≤ 1 /(pi + 2 * real n * pi)
          by(subst frac-le, simp-all, smt (verit, del-insts) m2pi-less-pi mult-sign-intros(1) of-nat-less-0-iff)
  then show ?thesis
    by (simp add: f1 x-nc-eqs inverse-eq-divide)
qed

— Bounds on sin(c) and cos(c).
have pi + c + 2*pi*real n ≥ 3*pi
proof −
  have pi + c + 2*pi*real n ≥ pi + 0 + 2*pi*real 1
    by (smt (verit, best) Num.of-nat-simps(2) c-bounds mult-left-mono n-ge-1
        pi-not-less-zero real-of-nat-ge-one-iff)
  then show ?thesis
    by linarith
qed
then have x-nc-bound: x-nc c ≤ inverse(3*pi)
  by (smt (verit) le-imp-inverse-le pi-gt-zero x-nc-eqs)
      then have cos-coef-bound: (1− 12 * (x-nc c)^2) ≥ (1− 12 * (inverse(3*pi))^2)
  using x-nc-pos by force

have sin-bound: 0 ≤ sin c ∧ sin c ≤ sqrt(2)/2
proof safe
  show 0 ≤ sin c
    using c-bounds sin-ge-zero by auto
  show sin c ≤ sqrt(2)/2
          by (smt (verit, best) c-bounds frac-le pi-not-less-zero sin-45 sin-mono-less-eq)
qed
have cos-bound: sqrt(2)/2 ≤ cos c ∧ cos c ≤ 1
proof safe
  show sqrt 2 / 2 ≤ cos c
         by (smt (verit) c-bounds cos-45 cos-monotone-0-pi-le machin pi-machin)
  show cos c ≤ 1
    by simp
qed

show 0 < deriv (deriv f) (x-nc c)
proof −
  — Lower bound of f''(x_nc).
    have snd-deriv-at-x-nc: deriv (deriv f) (x-nc c) = (1− 12 * (x-nc c)^2) * cos c − 6 * (x-nc c) * sin c + 24 * (x-nc c)^2
```

```
proof−
  have f1: sin (1 / (x-nc c)) = −sin c
  proof −
    have sin (1 / (x-nc c)) = sin (pi + c + 2∗pi∗real n)
      by (simp add: inverse-eq-divide x-nc-eqs)
    also have ... = sin (pi + c)
   by (metis Groups.mult-ac(2) id-apply of-real-eq-id sin.plus-of-nat)
    also have ... = −sin c
      by simp
    finally show ?thesis.
  qed
  have f2: cos (1 / (x-nc c)) = −cos c
  proof −
    have cos (1 / (x-nc c)) = cos (pi + c + 2∗pi∗real n)
      by (simp add: inverse-eq-divide x-nc-eqs)
    also have ... = cos (pi + c)
   by (metis Groups.mult-ac(2) id-apply of-real-eq-id cos.plus-of-nat)
    also have ... = −cos c
      by simp
    finally show ?thesis.
  qed

 have deriv (deriv f) (x-nc c) = (12∗(x-nc c)^2 − 1)∗ cos (1 / (x-nc c)) + 6∗(x-nc c) ∗ sin (1 / (x-nc c)) + 24∗(x-nc c)^2
    using deriv-f x-nc-pos by auto
  also have ... = (1− 12 ∗ (x-nc c)^2) ∗ cos c − 6 ∗ (x-nc c) ∗ sin c + 24 ∗ (x-nc c)^2
    by (smt (verit) f1 f2 minus-mult-commute more-arith-simps(8))
  finally show ?thesis.
qed
have snd-deriv-bound: deriv (deriv f) (x-nc c) ≥ (1 − 12 ∗ (x-nc c)^2 − 6 ∗ (x-nc c)) ∗ (sqrt 2 / 2)
proof −
  have deriv (deriv f) (x-nc c) ≥ (1− 12 ∗ (x-nc c)^2) ∗ cos c − 6 ∗ (x-nc c) ∗ (sqrt(2)/2) + 24 ∗ (x-nc c)^2
    using snd-deriv-at-x-nc sin-bound x-nc-pos by auto
  also have ... ≥ (1 − 12 ∗ (x-nc c)^2 − 6 ∗ (x-nc c)) ∗ (sqrt 2 / 2)
 by (smt (verit, best) calculation cos-bound divide-pos-pos one-power2 real-le-rsqrt right-diff-distrib' sum-le-prod1 vector-space-over-itself.scale-left-diff-distrib zero-compare-simps(12))
  then show ?thesis.
qed

show 0 < deriv (deriv f) (x-nc c)
proof −
 obtain h :: real ⇒ real where h-def: h = (λx. − 12 ∗ x^2 − 6 ∗ x + 1)
    by auto
  have diff-h: ∀ x. h field-differentiable at x
```

```
          unfolding h-def
        proof clarify
          fix x::real
          have d1: (λx. − 12 ∗ x^2) field-differentiable at x
      by(rule field-differentiable-mult, simp, simp add: field-differentiable-power)
          have d2: (λx. − 6 ∗ x) field-differentiable at x
      by(rule field-differentiable-mult, simp, simp add: field-differentiable-power)
         from d1 d2 show (λx. − 12 ∗ x^2 − 6 ∗ x + 1) field-differentiable
at x
      by(subst field-differentiable-add, simp add: Derivative.field-differentiable-diff,
simp-all)
        qed

         have h-roots: ∀x. h x = 0 ⟷ x = (−6 + sqrt 84) / 24 ∨ x =
(−6 − sqrt 84) / 24
        proof(clarify)
          fix x ::real
           have roots: (12 ∗ x^2 + 6 ∗ x + (−1) = 0) = (x = (− 6 + sqrt
(6^2 − 4 ∗ 12 ∗ (−1))) / (2 ∗ 12) ∨ x = (− 6 − sqrt (6^2 − 4 ∗ 12 ∗ (−1))) /
(2 ∗ 12))
                    using discrim-def by(subst discriminant-iff, eval, force)

          then show (h x = 0) = (x = (− 6 + sqrt 84) / 24 ∨ x = (− 6
− sqrt 84) / 24)
            using h-def by auto
        qed

        have right-root-positive: (− 6 + sqrt 84) / 24 > 0
        proof −
          have − 6 + sqrt 84 > − 6 + sqrt 64
            by (smt (verit) real-sqrt-less-mono)
          then show (− 6 + sqrt 84) / 24 > 0
            by simp
        qed
        then have left-root-neg: 0 > (− 6 − sqrt 84) / 24
          by fastforce
       have h-pos-on-interval: ∀ x ∈ {0..<(−6 + sqrt 84) / 24}. h x > 0
        proof(rule ccontr)
          assume ¬ (∀x∈{0..<(− 6 + sqrt 84) / 24}. 0 < h x)
           then obtain z where z-def: z ∈ {0..<(− 6 + sqrt 84) / 24} ∧
0 ≥ h z
            by fastforce
           then have z-not-root: z ≠ (− 6 + sqrt 84) / 24 ∧ z ≠ (− 6 −
sqrt 84) / 24
            using z-def by force
          show False
          proof(cases h z = 0)
            show h z = 0 ⟹ False
              using h-roots z-not-root by blast
```

```
  next
    assume h z ≠ 0
    then have hz-neg: h z < 0
      using z-def by auto
    have ∃x. 0 ≤ x ∧ x ≤ z ∧ h x = 0
    proof(rule IVT2')
      show h z ≤ 0
        by (simp add: z-def)
      show 0 ≤ h 0
        by (simp add: h-def)
      show 0 ≤ z
        using z-def by fastforce
      show continuous-on {0..z} h
                   by (meson continuous-at-imp-continuous-on diff-h
field-differentiable-imp-continuous-at)
    qed
    then show False
   by (metis atLeastLessThan-iff h-roots left-root-neg not-less z-def)
  qed
qed

have (−6 + sqrt 84) / 24 > 1 / (3 ∗ pi)
proof −
  have i1: 64 / pi^2 < 8
  proof −
    have pi∗pi > 3∗3
by (meson pi-gt3 mult-strict-mono pi-gt-zero verit-comp-simplify(7))
    then have pi^2 > 9
      by (simp add: power2-eq-square)
    then have 64/pi^2 < 64/8
      by (smt (verit) frac-less2)
    also have ... = 8
      by eval
    finally show ?thesis.
  qed

  have i2: 96/pi < 32
  proof −
    have 96/pi < 96/3
      by (meson frac-less2 order.refl pi-gt3 verit-comp-simplify(19))
    also have ... = 32
      by eval
    finally show ?thesis.
  qed

  have (8/pi + 6)^2 < 84
  proof −
    have ((8::real)/pi + 6)^2 = (8/pi)^2 + 2∗(8/pi)∗6 + 6^2
      by (simp add: power2-sum)
```

```
           also have ... = 8^2/pi^2 + 2∗(8/pi)∗6 + 6^2
            by (simp add: power-divide)
           also have ... = 64/pi^2 +96/pi + 36
            by simp
           also have ... < 84
            using i1 i2 by linarith
           finally show ?thesis.
          qed
          then have lt-sqrt84: 8/pi + 6 < sqrt(84)
           using real-less-rsqrt by presburger
          have lt-3pi-sqrt84: 24 + 18∗pi < 3 ∗ pi  ∗ sqrt (84)
          proof −
           have 24 + 18∗pi = 3∗8 + 3∗6∗pi
            by simp
           also have ... = 3∗pi∗(8/pi) + 3∗pi∗6
            by simp
           also have ... = 3∗pi∗((8/pi)+6)
            by (simp add: distrib-left)
           also have ... < 3 ∗ pi ∗ sqrt(84)
            by (simp add: lt-sqrt84)
           finally show ?thesis.
          qed
          have (−6+sqrt(84))∗(3∗pi) > 24
          proof −
           have (−6+sqrt(84))∗(3∗pi) = −6∗(3∗pi) + sqrt(84)∗(3∗pi)
            by (meson ring-class.ring-distribs(2))
           also have ... =  −18∗pi + 3∗pi ∗ sqrt(84)
            by simp
           also have ... > 24
            using lt-3pi-sqrt84 by auto
           finally show ?thesis.
          qed
          then have (−6+sqrt(84))∗(3∗pi) / 24 > 1
           by simp
          then show (−6+sqrt(84)) / 24 > 1 / (3∗pi)
              by (metis pi-gt-zero pos-divide-less-eq times-divide-eq-left
zero-compare-simps(6) zero-less-numeral)
         qed
         then have x-nc c < (−6+sqrt(84)) / 24
          by (metis dual-order.strict-trans2 inverse-eq-divide x-nc-bound)
         then have h-x-nc-pos: h (x-nc c) > 0
          by (simp add: h-pos-on-interval less-eq-real-def x-nc-pos)

         have deriv (deriv f) (x-nc c) ≥ (sqrt(2)/2) ∗ h (x-nc c)
            by (metis Groups.mult-ac(2) snd-deriv-bound diff-add-eq h-def
mult-minus-left uminus-add-conv-diff)
         then show ?thesis
       by (smt (verit) h-x-nc-pos half-gt-zero-iff mult-pos-pos real-sqrt-gt-0-iff)
        qed
```

```
       qed
      qed

      then show 0 < 6 ∗ y ∗ sin (1 / y) + (12 ∗ y^2 − 1) ∗ cos (1 / y) + 24 ∗ y^2
       by (smt (verit, best) deriv-f minimizer-dom')
     qed
      then show ∃y∈{left-seq n..right-seq n}. y^2 ∗ sin (1 / y) + 4 ∗ y ^ 3 ∗ cos (1 / y) + 8 ∗ y ^ 3 = 0 ∧ 0 < 6 ∗ y ∗ sin (1 / y) + (12 ∗ y^2 − 1) ∗ cos (1 / y) + 24 ∗ y^2
       using min-n-def by blast
    qed
   qed
  qed

  have optimality-conditions: ∀n. n ≠ 0 ⟶ (∃ y ∈ {left-seq n .. right-seq n}. (deriv f) y = 0 ∧ deriv (deriv f) y > 0)
  proof clarify
   fix n::nat
   assume 0 < n
   then obtain min-n where min-n-def: min-n ∈{left-seq n..right-seq n}
                                  ∧ min-n^2 ∗ sin (1 / min-n) + 4 ∗ min-n ^ 3 ∗ cos (1 / min-n) + 8 ∗ min-n ^ 3 = 0
                                   ∧ 0 < 6 ∗ min-n ∗ sin (1 / min-n) + (12 ∗ min-n^2 − 1) ∗ cos (1 / min-n) + 24 ∗ min-n^2
    using first-and-second-order-conditions bot-nat-0.not-eq-extremum by presburger
   have fst-order-condition: deriv f min-n = 0
    using deriv-f min-n-def by presburger
   have snd-order-condition: deriv (deriv f) min-n > 0
    using deriv-f min-n-def by fastforce
   show ∃y∈{left-seq n..right-seq n}. deriv f y = 0 ∧ 0 < deriv (deriv f) y
    using fst-order-condition min-n-def snd-order-condition by blast
  qed

  have seq-of-local-minizers-exists: ∀n. n ≠ 0 ⟶ (∃ y ∈ {left-seq n .. right-seq n}. local-minimizer f y)
  proof(clarify)
   fix n::nat
   assume n-pos: 0 < n
   then obtain y where y-def: (y ∈ {left-seq n .. right-seq n} ∧ (deriv f) y = 0 ∧ deriv (deriv f) y > 0)
    using gr-implies-not0 optimality-conditions by presburger
   have right-seq-def2: right-seq n = inverse (pi + 2 ∗ real n ∗ pi)
   by (metis id-apply less-not-refl n-pos of-nat-eq-id of-nat-in-Nats right-seq-def)

   have y ∈ {left-seq n..right-seq n} ∧ local-minimizer f y
   proof(subst second-derivative-test[where a = left-seq n, where b = right-seq n])
```

```
    show proper-interval: left-seq n < right-seq n
    by (metis (no-types) id-apply n-pos of-nat-eq-id of-nat-in-Nats rel-simps(70)
zero-lt-left-seq-lt-right-seq-both-pos)
    show C-k-on 2 f {left-seq n <..<right-seq n}
    proof(rule C-k-on-subset[where U = {0<..<(1::real)}])
      show f-contin-diff-on-right: C-k-on 2 f {0<..<(1::real)}
      proof(rule C2-on-open-U-def2)
        show open {0<..<(1::real)}
         using lemma-interval by(subst open-dist, subst dist-real-def, simp add:
abs-minus-commute lemma-interval-lt)
        show f differentiable-on {0<..<(1::real)}
          by (meson deriv-f differentiable-on-subset top.extremum)
        show deriv f differentiable-on {0<..<(1::real)}
          by (meson deriv-f differentiable-on-subset top.extremum)
        show continuous-on {0<..<(1::real)} (deriv (deriv f))
        proof -
          have ∀ x ∈ {0<..<1}. deriv (deriv f) x = 6*x * sin(1/x) + (12*x^2
− 1)*cos(1/x) + 24*x^2
            by (simp add: deriv-f)
          moreover have continuous-on {0<..<(1::real)} (λx. 6*x * sin(1/x)
+ (12*x^2 − 1)*cos(1/x) + 24*x^2)
          proof -
            have {0<..<(1::real)} ⊆ {x :: real. x>0}
              by fastforce
            moreover have continuous-on {x :: real. x>0} (λx. 6*x * sin(1/x)
+ (12*x^2 − 1)*cos(1/x) + 24*x^2)
              by (auto intro!: continuous-intros)
            ultimately show ?thesis
              using  continuous-on-subset by blast
          qed
          ultimately show continuous-on {0<..<1} (deriv (deriv f))
            using continuous-on-cong by fastforce
        qed
      qed

        show open {left-seq n<..<right-seq n} ∧ {left-seq n<..<right-seq n} ⊂
{0<..<1}
      proof -
        have 0 < left-seq n
               by (metis id-apply n-pos of-nat-eq-id of-nat-in-Nats order.asym
zero-lt-left-seq-lt-right-seq-both-pos)
        moreover have right-seq n < 1
          using right-seq-def2
         by (smt (verit, ccfv-SIG) inverse-1 inverse-le-imp-le mult-sign-intros(5)
n-pos of-nat-0-less-iff pi-gt3)
        ultimately show ?thesis
          using proper-interval by fastforce
      qed
    qed
```

```
        show y ∈ {left-seq n<..<right-seq n}
        proof −
          have y ∈ {left-seq n..right-seq n}
            using y-def by blast
          moreover have y ≠ left-seq n
          proof(rule ccontr)
            assume ¬ y ≠ left-seq n
            then have deriv f y ≠ 0
              using deriv-f first-and-second-order-conditions
              by (metis n-pos rel-simps(70) y-def)
            then show False
              by (simp add: y-def)
          qed
          moreover have y ≠ right-seq n
          proof(rule ccontr)
            assume ¬ y ≠ right-seq n
            then have deriv f y ≠ 0
              using deriv-f first-and-second-order-conditions
              by (metis n-pos rel-simps(70) y-def)
            then show False
              by (simp add: y-def)
          qed
          ultimately show y ∈ {left-seq n<..<right-seq n}
            by auto
        qed
        show deriv f y = 0 and 0 < deriv (deriv f) y
          using y-def by auto
        show y ∈ {left-seq n..right-seq n} ∧ True
          using y-def by blast
      qed
      then show ∃y∈{left-seq n..right-seq n}. local-minimizer f y
        by blast
    qed
    show ∃x-seq. (∀n. local-minimizer f (x-seq n) ∧ x-seq n ≠ 0) ∧ x-seq ⟶ 0
    proof −
      define x-seq where
       x-seq n = (SOME y. y ∈ {left-seq (n+1)..right-seq (n+1)} ∧ local-minimizer f y) for n
        have x-seq-prop: ∀n. x-seq n ∈ {left-seq (n+1)..right-seq (n+1)} ∧ local-minimizer f (x-seq n)
       by (metis (mono-tags, lifting) seq-of-local-minizers-exists someI-ex verit-eq-simplify(7) x-seq-def zero-eq-add-iff-both-eq-0)

        from x-seq-prop have bounds: ∀n. left-seq (n+1) ≤ x-seq n ∧ x-seq n ≤ right-seq (n+1)
        by auto

      have nonzero: ∀n. x-seq n ≠ 0
```

```
      by (metis Suc-eq-plus1 bounds id-apply nat.simps(3) not-less of-nat-eq-id
of-nat-in-Nats zero-lt-left-seq-lt-right-seq-both-pos)

    have left-seq-converges: left-seq ⟶ 0
    proof (rule LIMSEQ-I)
      fix ε :: real
      assume ε-pos: 0 < ε
      then obtain N where N-def: (N::nat) = ⌈1 / (2 ∗ pi ∗ ε)⌉ + 1
          by (metis add-mono-thms-linordered-field(5) arithmetic-simps(50) divide-pos-pos
            mult-sign-intros(5) pi-gt-zero pos-int-cases semiring-norm(172)
            zero-less-ceiling zero-less-numeral)
      then have N-gt-0: N > 0
     by (smt (verit) ε-pos divide-pos-pos gr0I int-ops(1) m2pi-less-pi mult-sign-intros(5)
zero-less-ceiling)

      have ∀n ≥ N. |left-seq n| < ε
      proof clarify
        fix n :: nat
        assume n-ge: n ≥ N
        have left-seq-eqs: left-seq n = inverse ((5 ∗ pi / 4) + 2 ∗ pi ∗ real n)
          unfolding left-seq-def
         by (metis N-gt-0 id-apply left-seq-def linorder-not-less mult.commute n-ge
of-nat-eq-id of-nat-in-Nats vector-space-over-itself.scale-scale)
        show |left-seq n| < ε
        proof −
          have |left-seq n| = 1 / ((5 ∗ pi / 4) + 2 ∗ pi ∗ real n)
            by (simp add: left-seq-eqs inverse-eq-divide)
          also have ... ≤  1 / (2 ∗ pi ∗ real N)
             by (smt (verit, best) N-gt-0 divide-nonneg-nonneg frac-le m2pi-less-pi
mult-left-mono mult-sign-intros(5) n-ge of-nat-0-less-iff of-nat-mono)
          also have ... <  1 / (2 ∗ pi ∗ (⌈1 / (2 ∗ pi ∗ ε)⌉))
           by (smt (verit, best) N-def ε-pos ceiling-correct divide-pos-pos frac-less2
m2pi-less-pi mult-less-cancel-left-pos mult-sign-intros(5) of-int-1 of-int-add of-int-of-nat-eq)
          also have ... ≤ 1 / (2 ∗ pi ∗ (1 / (2 ∗ pi ∗ ε)))
              by (smt (verit, ccfv-SIG) ε-pos ceiling-correct frac-le mult-left-mono
mult-sign-intros(5) pi-gt-zero zero-less-divide-iff)
          also have ... = ε
            by simp
          finally show ?thesis.
        qed
      qed
      then show ∃N. ∀n≥N. ‖left-seq n − 0‖ < ε
        by (metis cancel-comm-monoid-add-class.diff-zero real-norm-def)
    qed
    have right-seq-converges: right-seq ⟶ 0
    proof (rule LIMSEQ-I)
      fix ε::real
      assume eps-pos: 0 < ε
```

```
    then obtain N where N-def: (N::nat) = ⌈1 / (2 * pi * ε)⌉ + 1
        by (metis add-mono-thms-linordered-field(5) arithmetic-simps(50) divide-pos-pos
              mult-sign-intros(5) pi-gt-zero pos-int-cases semiring-norm(172)
              zero-less-ceiling zero-less-numeral)
    hence N-gt-0: N > 0
         by (smt (verit) eps-pos divide-pos-pos gr0I int-ops(1) m2pi-less-pi
mult-sign-intros(5)
              zero-less-ceiling)

    have ∀ n≥N. |right-seq n| < ε
    proof clarify
      fix n :: nat
      assume n-ge: n ≥ N
      have right-seq-eqs: right-seq n = inverse (pi + 2 * pi * real n)
        unfolding right-seq-def
     by (metis N-gt-0 id-apply linorder-not-less mult.commute mult.left-commute
n-ge of-nat-eq-id of-nat-in-Nats right-seq-def)
      show |right-seq n| < ε
      proof -
        have |right-seq n| = 1 / (pi + 2 * pi * real n)
          by (simp add: right-seq-eqs inverse-eq-divide)
        also have ... ≤ 1 / (2 * pi * real N)
          by (smt (verit, best) N-gt-0 divide-nonneg-nonneg frac-le m2pi-less-pi
                    mult-left-mono mult-sign-intros(5) n-ge of-nat-0-less-iff
of-nat-mono)
        also have ... < 1 / (2 * pi * (⌈1 / (2 * pi * ε)⌉))
       by (smt (verit, best) N-def eps-pos ceiling-correct divide-pos-pos frac-less2
m2pi-less-pi
                 mult-less-cancel-left-pos mult-sign-intros(5) of-int-1 of-int-add
of-int-of-nat-eq)
        also have ... ≤ 1 / (2 * pi * (1 / (2 * pi * ε)))
         by (smt (verit, ccfv-SIG) eps-pos ceiling-correct frac-le mult-left-mono
                mult-sign-intros(5) pi-gt-zero zero-less-divide-iff)
        also have ... = ε
          by simp
        finally show ?thesis
          by blast
      qed
    qed
    then show ∃ no. ∀ n≥no. ‖right-seq n − 0‖ < ε
      by (metis cancel-comm-monoid-add-class.diff-zero real-norm-def)
  qed
  have x-seq-converges: x-seq ⟶ 0
  proof (rule LIMSEQ-I)
    fix ε :: real
    assume ε-pos: 0 < ε

    obtain N₀ where N₀: ∀ n≥N₀. ‖left-seq (n+1) − 0‖ < ε
```

```
          using left-seq-converges
          by (meson LIMSEQ-iff ε-pos le-diff-conv)

        obtain N_1 where N_1: ∀n≥N_1. ‖right-seq (n+1) − 0‖ < ε
          using right-seq-converges
          by (meson LIMSEQ-iff ε-pos le-diff-conv)

        obtain N where N = max N_0 N_1
          by simp
        hence N-def: N ≥ N_0 ∧ N ≥ N_1
          by simp

        show ∃N. ∀n≥N. ‖x-seq n − 0‖ < ε
        proof (intro exI[where x=N] exI allI impI)
          fix n :: nat
          assume N-leq-n: N ≤ n

          from bounds have left-seq (n+1) ≤ x-seq n ∧ x-seq n ≤ right-seq (n+1)
            by auto
          hence ‖x-seq n‖ ≤ ‖left-seq (n+1)‖ ∨ ‖x-seq n‖ ≤ ‖right-seq (n+1)‖
            by force
          moreover have ‖left-seq (n+1)‖ < ε ∧ ‖right-seq (n+1)‖ < ε
            using N_0 N_1 N-leq-n N-def by auto
          ultimately show ‖x-seq n − 0‖ < ε
            by auto
        qed
      qed
      then show ?thesis
        using nonzero x-seq-prop by blast
    qed
  qed
  then show ?thesis
    using zero-min f-cont not-isolated-minimizer-def strict-local-minimizer-at-0 by auto
qed

end
theory Unconstrained-Optimization
  imports Auxilary-Facts
          Minimizers-Definition
          First-Order-Conditions
          Second-Derivative-Test
          Cont-Nonisolated-Strict-Local-Minimizer-Exists
begin

end
```

References

[1] T. Nipkow, M. Wenzel, and L. C. Paulson. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg, 2002.

[2] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer Series in Operations Research and Financial Engineering. Springer, New York, second edition, 2006.