Cauchy's Functional Equation

Michikazu Hirata

November 4, 2026

Abstract

Cauchy's functional equation is the equation

$$h(x + y) = h(x) + h(y).$$

This entry formalizes properties of the solutions $h : \mathbb{R} \to \mathbb{R}$ of Cauchy's functional equation, including:

- A function h satisfying Cauchy's functional equation has the form $h(x) = cx$ for some c if one of the following conditions holds:
  - h is continuous at some point.
  - h is continuous.
  - h is bounded on some interval.
  - h is (anti)monotone on some interval.
  - h is measurable.
- There are non-linear solutions to Cauchy's functional equation.
- The set of non-linear solutions has cardinality $|\mathbb{R}^{\mathbb{R}}|$.
- The graph of a non-linear solution $\{(x, h(x)) \mid x \in \mathbb{R}\}$ is dense in $\mathbb{R} \times \mathbb{R}$.

Contents

1 Cauchy's Functional Equation **2**
1.1 Case: h is Continuous 2
1.2 Case: h is Monotone 4
1.3 Case: h is Bounded 7
1.4 Case: h is Measurable 8
1.4.1 Lemmas for Regularity of Measures (to be moved) . . 8
1.4.2 Steinhaus' Theorem 10
1.4.3 Conclusion 15
1.5 Corollaries 15

2 Non-Linear Solutions **16**
2.1 Existence 16
2.2 Cardinality 21
2.3 Density of the Graph of a Non-Linear Solution 21

theory *Cauchy-Functional-Equation*
imports
HOL−Probability.Probability
Riesz-Representation.Regular-Measure
begin

1 Cauchy's Functional Equation

1.1 Case: h is Continuous

The proof of the following lemma is based on https://en.wikipedia.org/wiki/Cauchy%27s_functional_equation, accessed 17 January 2026.

lemma *Cauchy-equation-Q-linear*:
assumes *h-eq*:$\bigwedge x\ y.\ h\ (x + y) = h\ x + h\ y$
and *q*:$q \in \mathbb{Q}$
shows $h\ (q * v) = q * h\ v$
proof −
have *h0*:$h\ 0 = 0$
using *h-eq*[*of 0 0*] **by** *simp*
have *h-neq*:$h\ (-\ x) = -\ h\ x$ **for** x
using *h-eq*[*of x − x*] **by** (*simp add*: *add-eq-0-iff2 h0*)
have *N*:$h\ (of\text{-}nat\ n * r) = of\text{-}nat\ n * h\ r$ **for** $n\ r$
by(*induction n*) (*auto simp add*: *h0 mult.commute h-eq distrib-left*)
have *Z*:$h\ (of\text{-}int\ i * r) = of\text{-}int\ i * h\ r$ **for** $i\ r$
by(*rule int-cases2*[*of i*])
(*use N h-neq mult-minus-right*[*of h r of-nat -*]
of-int-minus[*of int -*] *of-int-of-nat-eq* **in** *auto*)
have *N-inverse*:$h\ (r\ /\ of\text{-}nat\ n) = h\ r\ /\ of\text{-}nat\ n$ **for** $r\ n$
by (*metis eq-divide-eq h0 mult.commute N*)
obtain $n\ m$ **where** *nm*: $q = of\text{-}int\ m\ /\ of\text{-}nat\ n$
by (*metis Rats-cases' q less-le of-nat-nat*)
show $h\ (q * v) = q * h\ v$
by(*simp add*: *nm N-inverse Z*)
qed

lemma *Cauchy-equation-isCont-continuous*:
fixes $h :: 'a :: real\text{-}normed\text{-}vector \Rightarrow 'b :: real\text{-}normed\text{-}vector$
assumes *h-eq*:$\bigwedge x\ y.\ h\ (x + y) = h\ x + h\ y$
and *h-Cont*:*isCont h x*
shows *continuous-on UNIV h*
proof(*safe intro*!: *continuous-at-imp-continuous-on*)
have *h0*:$h\ 0 = 0$

```
    using h-eq[of 0 0] by simp
  have h-neg:h (− x) = − h x for x
    using h-eq[of x − x] by (simp add: add-eq-0-iff2 h0)
  have eq:h (y + e) − h y = h e for y e
    using h-eq[of y + e − y] by(auto simp:  h-neg[symmetric])
  fix y :: 'a
  show isCont h y
    using h-Cont
    by(simp add: eq Lim-null[where l=h y] Lim-null[where l=h x] isCont-iff)
qed

lemma Cauchy-equation-continuous':
  fixes h :: - :: field-char-0 ⇒ real
  assumes h-eq:⋀x y. h (x + y) = h x + h y
    and h-Cont:continuous-on UNIV h
  shows h = (λx. x * h 1)
proof
  fix x :: real
  have q-eq:q ∈ ℚ ⟹ h q = q * h 1 for q
    using Cauchy-equation-Q-linear[of h q 1] h-eq by simp
  have ∃ q∈ℚ. x < q ∧ q < x + 1 / (1 + real n) for n
    using Rats-dense-in-real[of x x + 1 / (1 + real n)] by fastforce
  then obtain qn where
    qn:⋀n. qn n ∈ ℚ ⋀n. x < qn n ⋀n. qn n < x + 1 / (1 + real n)
    by metis
  have lim-qn:qn ⟶ x
    unfolding LIMSEQ-iff
  proof safe
    fix r :: real
    assume 0 < r
    then obtain N where 1 / (1 + real N) < r
      using nat-approx-posE by auto
    hence 1 / (1 + real n) < r if n:N ≤ n for n
      using divide-mono[of 1 + real N 1 + real n 1 1] of-nat-mono[OF n] by simp
    thus ∃ no. ∀ n≥no. norm (qn n − x) < r
      by(intro exI[where x=N]) (smt (verit, ccfv-SIG) qn real-norm-def)
  qed
  hence (λn. h (qn n)) ⟶ h x
    using h-Cont by(auto simp: continuous-on-sequentially comp-def)
  moreover have (λn. qn n * h 1) ⟶ x * h 1
    using lim-qn by(auto intro!: tendsto-mult-right)
  ultimately show h x = x * h 1
    using qn by(auto simp: q-eq[symmetric] intro!: tendsto-unique)
qed

corollary Cauchy-equation-continuous:
  fixes h :: - :: field-char-0 ⇒ real
  assumes ⋀x y. h (x + y) = h x + h y
    and continuous-on UNIV h
```

shows $\exists\, c.\ h = (\lambda x.\ x * c)$
using *assms Cauchy-equation-continuous'* **by** *blast*

corollary
fixes $h :: - :: \textit{field-char-0} \Rightarrow \textit{real}$
assumes *h-eq*:$\bigwedge x\ y.\ h\ (x + y) = h\ x + h\ y$
and *isCont h a*
shows *Cauchy-equation-continuous-at'*: $h = (\lambda x.\ x * h\ 1)$
and *Cauchy-equation-continuous-at*: $\exists\, c.\ h = (\lambda x.\ x * c)$
using *Cauchy-equation-continuous Cauchy-equation-continuous' assms Cauchy-equation-isCont-continuous* **by** *blast+*

1.2 Case: h is Monotone

lemma *Cauchy-bounded-zero-imp-continuous-at-zero*:
fixes $h :: 'a :: \{\textit{linordered-field}, \textit{linorder-topology}\} \Rightarrow \textit{real}$
assumes *h-eq*:$\bigwedge x\ y.\ h\ (x + y) = h\ x + h\ y$
and *ex*:$\exists \varepsilon{>}0.\ \exists M.\ \forall x.\ |x| < \varepsilon \longrightarrow |h\ x| \leq M$
shows *isCont h 0*
proof −
obtain $\varepsilon\ M$ **where** *eps*:$0 < \varepsilon$ **and** *bound*:$\bigwedge x.\ |x| < \varepsilon \Longrightarrow |h\ x| \leq M$
using *ex* **by** *blast*
have *h0*:$h\ 0 = 0$
using *h-eq*[*of 0 0*] **by** *simp*
have *Nmul*:$h\ (\textit{of-nat}\ n * r) = \textit{of-nat}\ n * h\ r$ **for** $n :: \textit{nat}$ **and** $r :: 'a$
by (*induction n*) (*auto simp*: *h0 h-eq distrib-right*)
have *M-nonneg*:$0 \leq M$
using *bound*[*of 0*] *eps* **by** *simp*
have *delta-bound*:$|h\ x| \leq M\ /\ \textit{real}\ n$
if *n*:$1 \leq n$ **and** *x*:$|x| < \varepsilon\ /\ \textit{of-nat}\ n$ **for** $x :: 'a$ **and** $n :: \textit{nat}$
proof −
have *n-pos*:$0 < (\textit{of-nat}\ n :: 'a)$ **and** *n-pos'*:$0 < \textit{real}\ n$
using *n* **by** *auto*
have *abs-lt*:$|\textit{of-nat}\ n * x| < \varepsilon$
using *mult-strict-left-mono*[*OF x n-pos*] *n-pos* **by** (*simp add*: *abs-mult*)
have $\textit{real}\ n * |h\ x| \leq M$
using *bound*[*OF abs-lt*] **by** (*simp add*: *Nmul abs-mult*)
thus *?thesis*
using *n-pos'* **by** (*simp add*: *field-simps*)
qed
have *main*:$\exists\, \delta{>}0.\ \forall x.\ |x| < \delta \longrightarrow |h\ x| < e$ **if** *e*:$0 < e$ **for** $e :: \textit{real}$
proof −
obtain *n0* **where** *n0*:$M\ /\ e < \textit{real}\ n0$
using *reals-Archimedean2* **by** *blast*
define n **where** $n = \textit{max}\ n0\ 1$
have *n1*:$1 \leq n$
by (*simp add*: *n-def*)
have *n*:$M\ /\ e < \textit{real}\ n$
using *n0 n1* **by** (*simp add*: *n-def del*: *max.bounded-iff*)

```
    have Me:M / real n < e
      using n n1 e M-nonneg by (simp add: field-simps)
    show ?thesis
    proof (intro exI[of - ε / of-nat n] conjI allI impI)
      show 0 < ε / of-nat n
        using eps n1 by simp
      fix x :: 'a
      assume x:|x| < ε / of-nat n
      show |h x| < e
        using delta-bound[OF n1 x] Me by linarith
    qed
  qed
  have ev-right:eventually (λx. |h x| < e) (at-right 0)
    and ev-left:eventually (λx. |h x| < e) (at-left 0) if e:0 < e for e
  proof −
    obtain δ where d:0 < δ and d':⋀x. |x| < δ ⟹ |h x| < e
      using main[OF e] by blast
    show eventually (λx. |h x| < e) (at-right 0)
      unfolding eventually-at-right-field
      by (intro exI[of - δ] allI impI conjI d) (use d' in force)
    show eventually (λx. |h x| < e) (at-left 0)
      unfolding eventually-at-left-field
      by (intro exI[of - − δ] allI impI conjI) (use d d' in force)+
  qed
  show ?thesis
    unfolding isCont-def tendsto-iff eventually-at-split
    using ev-right ev-left by (simp add: h0 dist-real-def)
qed

corollary
  fixes h :: - :: field-char-0 ⇒ real
  assumes h-eq:⋀x y. h (x + y) = h x + h y
    and ab:a < b
    and h-mono:mono-on {a..b} h
  shows Cauchy-equation-mono-on': h = (λx. x * h 1)
    and Cauchy-equation-mono-on: ∃ c. h = (λx. x * c)
proof −
  have h0:h 0 = 0
    using h-eq[of 0 0] by simp
  have h-neg:h (− x) = − h x for x
    using h-eq[of x − x] by (simp add: add-eq-0-iff2 h0)
  have h-eq':h (x − y) = h x − h y for x y
    using h-eq[of x − y] by(auto simp: h-neg[symmetric])
  define m where m = (a + b) / 2
  have bound:|h y| ≤ h b − h a if y:|y| < b − a for y
  proof −
    define x where x = y / 2
    have mem:m + x ∈ {a..b} m − x ∈ {a..b}
      using y by (auto simp: m-def x-def abs-less-iff, argo+)
```

```
    have hb:h a ≤ h (m + x) h (m + x) ≤ h b h a ≤ h (m − x) h (m − x) ≤ h b
      using h-mono mem ab by (auto simp: mono-on-def)
    have h y = h (m + x) − h (m − x)
      using h-eq'[of m + x m − x] by (simp add: x-def)
    thus ?thesis
      using hb by linarith
  qed
  have ex:∃ε>0. ∃ M. ∀ x. |x| < ε ⟶ |h x| ≤ M
    using ab bound by (intro exI[of - b − a] exI[of - h b − h a]) auto
  note isc = Cauchy-bounded-zero-imp-continuous-at-zero[OF h-eq ex]
  show h = (λx. x ∗ h 1)
    using Cauchy-equation-continuous-at'[OF h-eq isc] .
  thus ∃ c. h = (λx. x ∗ c)
    by blast
qed

corollary
  fixes h :: - :: field-char-0 ⇒ real
  assumes h-eq:⋀x y. h (x + y) = h x + h y
    and h-mono:mono h
  shows Cauchy-equation-mono': h = (λx. x ∗ h 1)
    and Cauchy-equation-mono: ∃ c. h = (λx. x ∗ c)
  using Cauchy-equation-mono-on'[of h,OF h-eq - mono-on-subset[OF h-mono]]
  by auto

corollary
  fixes h :: - :: field-char-0 ⇒ real
  assumes h-eq:⋀x y. h (x + y) = h x + h y
    and ab:a < b
    and h-mono:antimono-on {a..b} h
  shows Cauchy-equation-antimono-on': h = (λx. x ∗ h 1)
    and Cauchy-equation-antimono-on: ∃ c. h = (λx. x ∗ c)
proof −
  have h-eq2:⋀x y. (− h) (x + y) = (− h) x + (− h) y
    using h-eq by simp
  have h-mono2:mono-on {a..b} (− h)
    using h-mono by (auto simp: monotone-on-def)
  show h = (λx. x ∗ h 1)
  proof
    fix x
    show h x = x ∗ h 1
      using fun-cong[OF Cauchy-equation-mono-on'[OF h-eq2 ab h-mono2], of x]
by simp
  qed
  thus ∃ c. h = (λx. x ∗ c)
    by blast
qed
```

1.3 Case: h is Bounded

corollary
fixes $h :: - :: field\text{-}char\text{-}0 \Rightarrow real$
assumes $h\text{-}eq: \bigwedge x\ y.\ h\ (x + y) = h\ x + h\ y$
and $ab: a < b$
and $h\text{-}bdd\text{-}above: \exists\, M.\ \forall\, x \in \{a..b\}.\ h\ x \leq M$
shows $Cauchy\text{-}equation\text{-}bounded\text{-}above\text{-}on'$: $h = (\lambda x.\ x * h\ 1)$
and $Cauchy\text{-}equation\text{-}bounded\text{-}above\text{-}on$: $\exists\, c.\ h = (\lambda x.\ x * c)$
proof −
have $h0: h\ 0 = 0$
using $h\text{-}eq[of\ 0\ 0]$ **by** $simp$
have $h\text{-}neg: h\ (-x) = -\ h\ x$ **for** x
using $h\text{-}eq[of\ x\ -x]$ **by** $(simp\ add: add\text{-}eq\text{-}0\text{-}iff2\ h0)$
have $h\text{-}eq': h\ (x - y) = h\ x - h\ y$ **for** $x\ y$
using $h\text{-}eq[of\ x\ -y]$ **by**$(auto\ simp: h\text{-}neg[symmetric])$
obtain M **where** $M: \bigwedge x.\ x \in \{a..b\} \Longrightarrow h\ x \leq M$
using $h\text{-}bdd\text{-}above$ **by** $blast$
define L **where** $L = b - a$
define M' **where** $M' = M - h\ a$
have $L\text{-}pos: 0 < L$
using ab **by** $(simp\ add: L\text{-}def)$
have $M'\text{-}bound: h\ x \leq M'$ **if** $x: x \in \{0..L\}$ **for** x
using $M[of\ x + a]\ h\text{-}eq[of\ x\ a]\ x$ **by** $(simp\ add: L\text{-}def\ M'\text{-}def)$
have $bound\text{-}abs: |h\ x| \leq M' + |h\ L|$ **if** $x: x \in \{0..L\}$ **for** x
proof −
have $h\ L - h\ x \leq M'$
using $M'\text{-}bound[of\ L - x]\ h\text{-}eq'[of\ L\ x]\ x$ **by** $simp$
thus $?thesis$
using $M'\text{-}bound[OF\ x]$ **by** $linarith$
qed
have $near\text{-}0: |h\ x| \leq M' + |h\ L|$ **if** $xL: |x| < L$ **for** x
using $bound\text{-}abs[of\ x]\ xL\ h\text{-}neg[of\ x]\ bound\text{-}abs[of\ -x]$
by $(cases\ 0 \leq x)\ auto$
have $ex: \exists\, \varepsilon > 0.\ \exists\, M.\ \forall\, x.\ |x| < \varepsilon \longrightarrow |h\ x| \leq M$
using $L\text{-}pos\ near\text{-}0$ **by** $blast$
note $isc = Cauchy\text{-}bounded\text{-}zero\text{-}imp\text{-}continuous\text{-}at\text{-}zero[OF\ h\text{-}eq\ ex]$
show $h = (\lambda x.\ x * h\ 1)$
using $Cauchy\text{-}equation\text{-}continuous\text{-}at'[OF\ h\text{-}eq\ isc]$ **.**
thus $\exists\, c.\ h = (\lambda x.\ x * c)$
by $blast$
qed

corollary
fixes $h :: - :: field\text{-}char\text{-}0 \Rightarrow real$
assumes $h\text{-}eq: \bigwedge x\ y.\ h\ (x + y) = h\ x + h\ y$
and $ab: a < b$
and $h\text{-}bdd\text{-}below: \exists\, M.\ \forall\, x \in \{a..b\}.\ M \leq h\ x$
shows $Cauchy\text{-}equation\text{-}bounded\text{-}below\text{-}on'$: $h = (\lambda x.\ x * h\ 1)$
and $Cauchy\text{-}equation\text{-}bounded\text{-}below\text{-}on$: $\exists\, c.\ h = (\lambda x.\ x * c)$

proof −
have *h-eq2*:⋀*x y*. (− *h*) (*x* + *y*) = (− *h*) *x* + (− *h*) *y*
using *h-eq* **by** *simp*
obtain *M* **where** *M*:⋀*x*. *x* ∈ {*a*..*b*} ⟹ *M* ≤ *h x*
using *h-bdd-below* **by** *blast*
have *h-bdd-above2*:∃ *M*. ∀ *x*∈{*a*..*b*}. (− *h*) *x* ≤ *M*
using *M* **by** (*intro exI*[*of* - − *M*]) *simp*
show *h* = (λ*x*. *x* ∗ *h 1*)
proof
fix *x*
show *h x* = *x* ∗ *h 1*
using *fun-cong*[*OF Cauchy-equation-bounded-above-on'*[*OF h-eq2 ab h-bdd-above2*],
of x]
by *simp*
qed
thus ∃ *c*. *h* = (λ*x*. *x* ∗ *c*)
by *blast*
qed

corollary
fixes *h* :: - :: *field-char-0* ⇒ *real*
assumes *h-eq*:⋀*x y*. *h* (*x* + *y*) = *h x* + *h y*
and *ab*:*a* < *b*
and *h-bdd*:∃ *M*. ∀ *x*∈{*a*..*b*}. |*h x*| ≤ *M*
shows *Cauchy-equation-bounded-on'*: *h* = (λ*x*. *x* ∗ *h 1*)
and *Cauchy-equation-bounded-on*: ∃ *c*. *h* = (λ*x*. *x* ∗ *c*)
proof −
have *h-bdd-above*:∃ *M*. ∀ *x*∈{*a*..*b*}. *h x* ≤ *M*
using *h-bdd abs-le-D1* **by** *blast*
show *h* = (λ*x*. *x* ∗ *h 1*)
using *Cauchy-equation-bounded-above-on'*[*OF h-eq ab h-bdd-above*] .
thus ∃ *c*. *h* = (λ*x*. *x* ∗ *c*)
by *blast*
qed

1.4 Case: h is Measurable

1.4.1 Lemmas for Regularity of Measures (to be moved)

lemma *lborel-inner-regular*:
assumes *A*[*measurable*]:*A* ∈ *sets borel* **and** *A-bdd*:*bounded A*
shows *measure lborel A* = (⨆ *K*∈{*K*. *compact K* ∧ *K* ⊆ *A*}. *measure lborel K*)
proof −
obtain *a* **where** *a*:*A* ⊆ *cball 0 a*
by (*meson A-bdd bounded-subset-cball cball-subset-cball-iff order.trans*
order-refl)
interpret *finite-measure restrict-space lborel* (*cball 0 a*)
using *emeasure-lborel-cball-finite less-top*
by(*fastforce intro*!: *finite-measureI simp*: *emeasure-restrict-space*)
let *?X* = *subtopology euclidean* (*cball 0 a*)

have *measure lborel A = measure (restrict-space lborel (cball 0 a)) A*
using *a* **by**(*simp add*: *measure-restrict-space*)
also have ...
= (⨆ (*measure (restrict-space lborel (cball 0 a)) ‘ {K. compactin ?X K ∧ K ⊆ A}*))
using *a* **by**(*intro inner-regular-Polish Polish-space-closedin*)
(*auto simp*: *borel-of-subtopology borel-of-euclidean sets-restrict-space*)
also have ... = (⨆ *K*∈{*K. compact K ∧ K ⊆ A*}. *measure lborel K*)
using *a* **by**(*auto intro*!: *Sup.SUP-cong simp*: *compactin-subtopology measure-restrict-space*)
finally show *?thesis* **.**
qed

corollary *lborel-inner-regular-eps*:
assumes *A*[*measurable*]:*A ∈ sets borel* **and** *A-bdd*:*bounded A*
and *e*:*0 < e*
obtains *K* **where** *K ⊆ A compact K measure lborel A < measure lborel K + e*
proof −
have ∗:{*K. compact K ∧ K ⊆ A*} ≠ {}
by *blast*
have ∗∗:*bdd-above* (*measure lborel ‘ {K. compact K ∧ K ⊆ A}*)
by(*auto intro*!: *bdd-aboveI*[**where** *M*=*measure lborel A*] *fmeasurableI*
measure-mono-fmeasurable borel-compact emeasure-bounded-finite A-bdd)
have ∗∗∗:⋀*y. y < measure lborel A*
⟹ ∃ *i*∈{*K. compact K ∧ K ⊆ A*}. *y < measure lborel i*
using *lborel-inner-regular*[*OF A A-bdd*] *le-cSUP-iff*[*OF* ∗ ∗∗] **by** *auto*
show *?thesis*
using *that* ∗∗∗[*of measure lborel A − e*] *e* **by** *force*
qed

lemma *lborel-outer-regular*:
assumes *A*[*measurable*]:*A ∈ sets borel* **and** *A-bdd*:*bounded A*
shows *measure lborel A* = (⨅ (*measure lborel ‘ {C. open C ∧ A ⊆ C ∧ bounded C}*))
proof −
obtain *a′* **where** *a′*:*A ⊆ cball 0 a′*
by (*meson A-bdd bounded-subset-cball cball-subset-cball-iff order.trans order-refl*)
then obtain *a* **where** *a*:*a′ < a*
using *gt-ex* **by** *blast*
hence *A-subst-ball*:*A ⊆ ball 0 a*
using *a′* **by** *fastforce*
interpret *finite-measure restrict-space lborel* (*ball 0 a*)
using *emeasure-lborel-ball-finite less-top*
by(*fastforce intro*!: *finite-measureI simp*: *emeasure-restrict-space*)
let *?X = subtopology euclidean* (*ball 0 a*)
have *measure lborel A = measure* (*restrict-space lborel* (*ball 0 a*)) *A*
using *A-subst-ball* **by**(*simp add*: *measure-restrict-space*)
also have ...
= (⨅ (*measure* (*restrict-space lborel* (*ball 0 a*))‘ {*C. openin ?X C ∧ A ⊆ C*}))

```
  using A-subst-ball by(intro outer-regularD outer-regular' metrizable-space-subtopology)
    (auto simp: sets-restrict-space borel-of-subtopology borel-of-euclidean
      metrizable-space-euclidean)
  also have ... = (⨅ (measure lborel ` {C. open C ∧ A ⊆ C ∧ bounded C})) (is
?lhs = ?rhs)
  proof(rule antisym)
    show ?lhs ≥ ?rhs
      using A-subst-ball
      by(intro cInf-superset-mono)
        (auto simp: openin-open-eq[of ball 0 a] measure-restrict-space
          dest: bounded-subset-ballI)
  next
    show ?lhs ≤ ?rhs
    proof(safe intro!: cInf-mono)
      fix U
      assume open U A ⊆ U bounded U
      thus ∃ a∈measure (restrict-space lborel (ball 0 a)) `
                {C. openin (top-of-set (ball 0 a)) C ∧ A ⊆ C}.
              a ≤ Henstock-Kurzweil-Integration.content U
        using A-subst-ball
        by(auto intro!: exI[where x=ball 0 a ∩ U] measure-mono-fmeasurable
            fmeasurableI emeasure-bounded-finite simp: measure-restrict-space)
    qed(use A-subst-ball in auto intro!: bdd-belowI[where m=0])
  qed
  finally show ?thesis .
qed

corollary lborel-outer-regular-eps:
  assumes A[measurable]:A ∈ sets borel and A-bdd:bounded A
    and e:0 < e
  obtains U where A ⊆ U open U bounded U measure lborel U < measure lborel
A + e
proof −
  have ∗:{C. open C ∧ A ⊆ C ∧ bounded C} ≠ {}
    by (metis (no-types, lifting) open-ball bounded-ball A-bdd
        bounded-subset-ballD empty-iff mem-Collect-eq)
  have ∗∗:bdd-below (measure lborel ` {C. open C ∧ A ⊆ C ∧ bounded C})
    by fast
  have ∗∗∗:⋀y. y > measure lborel A
            ⟹ ∃ i∈{C. open C ∧ A ⊆ C ∧ bounded C}. measure lborel i < y
    using lborel-outer-regular[OF A A-bdd] cINF-le-iff[OF ∗ ∗∗] by auto
  show ?thesis
    using that ∗∗∗[of measure lborel A + e] e by auto
qed
```

1.4.2 Steinhaus' Theorem

The proof of the following lemma is based on https://matthewhr.wordpress.com/2012/11/14/steinhauss-theorem/.

```
lemma weak-Steinhaus:
  fixes A :: 'a :: euclidean-space set
  assumes [measurable]:A ∈ sets borel
    and meas:0 < emeasure lborel A
  shows 0 ∈ interior {a − a'|a a'. a ∈ A ∧ a' ∈ A}
proof −
  have exr:∃ r>0. ∀ x∈K. ball x r ⊆ U
    if K:compact K and UK:K ⊆ U and U:open U for U K :: 'a set
    using Met-TC.lebesgue-number[of K {U}] that by auto
  have ∃ C'∈sets borel. bounded C' ∧ C' ⊆ C
                  ∧ 0 < emeasure lborel C' ∧ emeasure lborel C' < ⊤
    if C[measurable]:C ∈ sets borel 0 < emeasure lborel C for C :: 'a set
  proof −
    define Cn where Cn ≡ (λn. C ∩ ball 0 (real n))
    have [measurable]: Cn n ∈ sets borel for n
      by(simp add: Cn-def)
    have inc:incseq Cn
      by(auto simp: Cn-def intro!: monoI)
    have Cn-subst:Cn n ⊆ C for n
      by(simp add: Cn-def)
    have meas-fin:emeasure lborel (Cn n) < ⊤ for n
      by (metis Cn-def bounded-Int bounded-ball emeasure-bounded-finite
          infinity-ennreal-def)
    have UnCn: C = (⋃ (range Cn))
      using reals-Archimedean2[of abs -]
      by(fastforce simp: Cn-def abs-less-iff minus-less-iff)
    have bCn: bounded (Cn n) for n
      by(auto simp: Cn-def)
    have ∃ n. 0 < emeasure lborel (Cn n)
    proof(rule ccontr)
      assume ∄ n. 0 < emeasure lborel (Cn n)
      then have emeasure0:⋀n. emeasure lborel (Cn n) = 0
        by simp
      have (λn. emeasure lborel (Cn n)) ⟶ emeasure lborel C
        by(auto intro!: Lim-emeasure-incseq inc simp: UnCn)
      thus False
        using C by(simp add: emeasure0 LIMSEQ-const-iff)
    qed
    then obtain n where 0 < emeasure lborel (Cn n)
      by blast
    thus ?thesis
      using Cn-subst bCn meas-fin by(auto intro!: bexI[where x=Cn n])
  qed
  then obtain A' where
    A'[measurable]: A' ∈ sets borel bounded A' A' ⊆ A
                    0 < emeasure lborel A' emeasure lborel A' < ⊤
    using assms by meson
  hence 0 < measure lborel A'
    by (simp add: emeasure-eq-ennreal-measure)
```

```
  then obtain e where e:0 < e e < measure lborel A' / 4
    by (metis dense zero-less-divide-iff zero-less-numeral)
  then obtain U K where
    U[measurable]:A' ⊆ U open U bounded U measure lborel U < measure lborel A'
+ e
    and K:K ⊆ A' compact K measure lborel A' < measure lborel K + e
    using lborel-outer-regular-eps lborel-inner-regular-eps A' by meson
  hence meas-le:measure lborel A' ≤ measure lborel U
              measure lborel K ≤ measure lborel A'
    by(auto intro!: measure-mono-fmeasurable fmeasurableI borel-compact
        emeasure-bounded-finite A')
  have less-UK:measure lborel U < 2 * measure lborel K
  proof −
    have measure lborel U / 2 ≤ measure lborel U − measure lborel A' / 2
      using meas-le by simp
    also have ... < measure lborel U − 2 * e
      using e by auto
    also have ... < measure lborel K
      using U K by linarith
    finally show ?thesis
      by simp
  qed
  obtain r where r:⋀x. x ∈ K ⟹ ball x r ⊆ U and r-pos: r > 0
    using exr[of K U] U K by fast
  have ball 0 r ⊆ {a − a'|a a'. a ∈ A ∧ a' ∈ A}
  proof(rule ccontr)
    assume ¬ ball 0 r ⊆ {a − a' |a a'. a ∈ A ∧ a' ∈ A}
    then have ball 0 r ∩ (UNIV − {a − a' |a a'. a ∈ K ∧ a' ∈ K}) ≠ {}
      using K A' by blast
    then obtain x where x:x ∈ ball 0 r (+) x ‘ K ∩ K = {}
      by fastforce
    have Kx-U:(+) x ‘ K ⊆ U
    proof safe
      show k ∈ K ⟹ x + k ∈ U for k
        using r[of k] x
        by (metis Groups.add-ac(2) add-0 image-add-ball rev-image-eqI subset-iff)
    qed
    have 2 * measure lborel K = 2 * measure lebesgue K
      by (simp add: K borel-compact)
    also have ... = measure lebesgue ((+) x ‘ K) + measure lebesgue K
      by(simp add: measure-translation)
    also have ... = measure lborel ((+) x ‘ K) + measure lborel K
      by(simp add: K borel-compact compact-translation)
    also have ... = measure lborel (((+) x ‘ K) ∪ K)
      using emeasure-compact-finite[of K] emeasure-compact-finite[of (+) x ‘ K]
      using x by(intro measure-Union[symmetric])
            (auto simp: K compact-translation borel-compact)
    also have ... ≤ measure lborel U
      using U K r Kx-U
```

```
    by(auto intro!: measure-mono-fmeasurable fmeasurableI emeasure-bounded-finite
        simp: compact-translation borel-compact)
    finally show False
      using less-UK by auto
  qed
  thus ?thesis
    using r-pos mem-interior by blast
qed
```

The proof of the following lemma is based on https://math.stackexchange.com/questions/45861/show-that-fxy-fxfy-implies-f-continuous-leftrightarrow-f-measur.

```
lemma Cauchy-equation-continuous-iff-measurable:
  fixes h :: real ⇒ real
  assumes h-eq:⋀x y. h (x + y) = h x + h y
  shows continuous-on UNIV h ⟷ h ∈ borel-measurable borel
proof
  assume [measurable]: h ∈ borel-measurable borel
  have h0:h 0 = 0
    using h-eq[of 0 0] by simp
  have h-neg:h (− x) = − h x for x
    using h-eq[of x − x] by (simp add: add-eq-0-iff2 h0)
  have h-eq':h (x − y) = h x − h y for x y
    using h-eq[of x − y] by(auto simp: h-neg[symmetric])
  have isCont h 0
    unfolding continuous-at-real-range
  proof safe
    fix e :: real
    assume e:0 < e
    have h0:h 0 = 0
      using h-eq[of 0 0] by simp
    have [measurable]:A ∈ sets borel ⟹ h −‘ A ∈ sets borel for A
      by(auto intro!: measurable-sets-borel)
    have ∃ a b. a < b ∧ b − a = e ∧ 0 < emeasure lborel (h −‘ {a<..<b})
    proof(rule ccontr)
      assume ∄ a b. a < b ∧ b − a = e ∧
          0 < emeasure lborel (h −‘ {a<..<b})
     then have h:⋀a b. a < b ⟹ b − a = e ⟹ emeasure lborel (h −‘ {a<..<b})
= 0
        using not-gr-zero by blast
      have ∞ = emeasure lborel (UNIV :: real set)
        by simp
      also have ... = emeasure lborel
        (h −‘ (⋃ n. {e * real-of-int (from-nat n) / 2 <..<e * real-of-int (from-nat
n) / 2 + e}))
      proof −
        have x ∈ (⋃ n.
              {e * real-of-int (from-nat n) / 2 <..<e * real-of-int (from-nat n) / 2
+ e})
          for x
```

```
      proof −
        obtain i :: int where i ≤ 2 * x / e and i2: 2 * x / e < i + 1
          by (meson floor-exists1)
        then have i1:i − 1 < 2 * x / e
          by linarith
        obtain n where n:from-nat n = i − 1
          using from-nat-to-nat by blast
        thus ?thesis
          using i1 i2 e by(auto intro!: exI[where x=n]
              simp: mult.commute n pos-less-divide-eq pos-divide-less-eq
              right-diff-distrib distrib-left diff-divide-distrib)
      qed
     hence *:(⋃ n. {e * real-of-int (from-nat n) / 2 <..<e * real-of-int (from-nat
n) / 2 + e})
                = UNIV
        by fast
      thus ?thesis
        unfolding * by simp
    qed
    also have ... = emeasure lborel
     ((⋃ n. h −‘ ({e * real-of-int (from-nat n) / 2 <..<e * real-of-int (from-nat
n) / 2 + e})))
      by(rule arg-cong[where f=emeasure lborel]) blast
    also have ... ≤ (∑ n. emeasure lborel
          (h −‘ {e * real-of-int (from-nat n) / 2 <..<e * real-of-int (from-nat n)
/ 2 + e}))
      by(rule emeasure-subadditive-countably) auto
    also have ... = (∑ n. 0)
      by(intro suminf-cong h) (use e in auto)
    finally show False
      by simp
  qed
   then obtain a b where ab:a < b b − a = e 0 < emeasure lborel (h −‘
{a<..<b})
    by blast
  then obtain d where d: d > 0
    ball 0 d ⊆ {α − β|α β. α ∈ h −‘ {a<..<b} ∧ β ∈ h −‘ {a<..<b}}
    using weak-Steinhaus[of h −‘ {a<..<b}]
    by (auto simp: mem-interior)
  have {α − β|α β. α ∈ h −‘ {a<..<b} ∧ β ∈ h −‘ {a<..<b}} ⊆
      h −‘ {c − d|c d. c ∈ {a<..<b} ∧ d ∈ {a<..<b}}
    by(fastforce simp: h-eq′)
  also have ... ⊆ h −‘ ({−e<..<e})
    using ab by auto
  finally show ∃ δ>0. ∀ x′. norm (x′ − 0) < δ ⟶ |h x′ − h 0| < e
    using d
    by(intro exI[where x=d]) (simp add: abs-less-iff h0 minus-less-iff subset-eq)
 qed
 thus continuous-on UNIV h
```

by(*intro Cauchy-equation-isCont-continuous h-eq*)
qed(*rule borel-measurable-continuous-onI*)

1.4.3 Conclusion

corollary
fixes $h :: real \Rightarrow real$
assumes *h-meas*[*measurable*]:$h \in$ *borel-measurable borel*
and *h-eq*:$\bigwedge x\ y.\ h\ (x + y) = h\ x + h\ y$
shows *Cauchy-equation-measurable'*: $h = (\lambda x.\ x * h\ 1)$
and *Cauchy-equation-measurable*: $\exists\, c.\ h = (\lambda x.\ x * c)$
using *Cauchy-equation-continuous-iff-measurable assms*
Cauchy-equation-continuous' Cauchy-equation-continuous
by *blast+*

1.5 Corollaries

corollary *multiplicative-measurable-powr*:
fixes $A :: real \Rightarrow real$
assumes A:$A \in$ *borel-measurable* (*restrict-space borel* $\{0<..\}$)
and *A-pos*:$\bigwedge x.\ 0 < x \Longrightarrow 0 < A\ x$
and *eq*:$\bigwedge x\ y.\ 0 < x \Longrightarrow 0 < y \Longrightarrow A\ (x * y) = A\ x * A\ y$
obtains γ **where** $\bigwedge x.\ 0 < x \Longrightarrow A\ x = x\ powr\ \gamma$
proof −
define h **where** $h = (\lambda s.\ ln\ (A\ (exp\ s)))$
have *h-eq*:$h\ (s + t) = h\ s + h\ t$ **for** $s\ t$
using *A-pos*[*OF exp-gt-zero*[*of s*]] *A-pos*[*OF exp-gt-zero*[*of t*]]
by(*auto simp*: *h-def exp-add eq ln-mult*)
have *h-meas*:$h \in$ *borel-measurable borel*
unfolding *h-def*
by (*intro borel-measurable-ln measurable-compose*[*OF - A*] *measurable-restrict-space2*)
auto
obtain c **where** c:$h = (\lambda s.\ s * c)$
using *Cauchy-equation-measurable h-meas h-eq* **by** *blast*
show *?thesis*
proof (*rule that*[*of c*])
fix $x :: real$
assume x:$0 < x$
have $A\ x = exp\ (h\ (ln\ x))$
unfolding *h-def* **using** x *A-pos*[*OF x*] **by** *simp*
also have $\ldots = x\ powr\ c$
using $c\ x$ **by** (*simp add*: *powr-def*)
finally show $A\ x = x\ powr\ c$ **.**
qed
qed

lemma *cocycle-mono-solution*:
fixes $H :: real \Rightarrow real$ **and** $\gamma :: real$
assumes *mono*:*mono H*
and *eq*:$\bigwedge u\ v.\ H\ (u + v) = H\ u * exp\ (\gamma * v) + H\ v$

```
  obtains c where 0 ≤ c
    and ⋀u. H u = (if γ = 0 then c * u else c * (exp (γ * u) − 1) / γ)
proof −
  have H0:H 0 = 0
    using eq[of 0 0] by simp
  have H1-nonneg:0 ≤ H 1
    using monoD[OF mono, of 0 1] H0 by simp
  show ?thesis
  proof (cases γ = 0)
    assume gamma:γ = 0
    have eq':H (u + v) = H u + H v for u v
      using eq[of u v] gamma by simp
    obtain c where c:H = (λx. x * c)
      using Cauchy-equation-mono eq' mono by blast
    have 0 ≤ c
      using monoD[OF mono, of 0 1] c by simp
    thus ?thesis
      using that[of c] by(simp add: gamma c)
  next
    assume gamma:γ ≠ 0
    have swap:H u * (exp (γ * v) − 1) = H v * (exp (γ * u) − 1) for u v
      using eq[of u v] eq[of v u] by (simp add: algebra-simps)
    define k where k = H 1 / (exp γ − 1)
    have Hform:H u = k * (exp (γ * u) − 1) for u
      using swap[of u 1] gamma by (simp add: k-def field-simps)
    define c where c = γ * k
    have ratio-pos:0 < γ / (exp γ − 1)
      by(cases 0 < γ)
        (use gamma in auto simp add: zero-less-divide-iff)
    have 0 ≤ c
      using mult-nonneg-nonneg[OF H1-nonneg less-imp-le[OF ratio-pos]]
      by(simp add: c-def k-def mult.commute)
    thus ?thesis
      using that[of c] by(simp add: gamma Hform c-def)
  qed
qed
```

2 Non-Linear Solutions

Cf. Hamel [1] and https://en.wikipedia.org/wiki/Cauchy%27s_functional_equation.

2.1 Existence

```
lemma infinite-Hamel-basis:
  fixes B :: real set
  assumes module.span (λq x. of-rat q * x) B = UNIV
  shows infinite B
```

```
proof −
  interpret ratvs: vector-space λq x. of-rat q ∗ (x :: real)
    by unfold-locales (auto simp: of-rat-add of-rat-mult algebra-simps)
  have ∗:countable (range (λu :: real ⇒ rat. ∑ v∈S. of-rat (u v) ∗ v))
    if S:finite S for S
  proof −
    have sub:range (λu :: real ⇒ rat. ∑ v∈S. of-rat (u v) ∗ v)
              ⊆ (λu. ∑ v∈S. of-rat (u v) ∗ v) ‘ (S →E (UNIV :: rat set))
    proof safe
      fix u :: real ⇒ rat
      show (∑ v∈S. of-rat (u v) ∗ v)
            ∈ (λu. ∑ v∈S. of-rat (u v) ∗ v) ‘ (S →E (UNIV :: rat set))
        unfolding image-iff
        by (intro bexI[of - restrict u S] sum.cong) (auto simp: PiE-iff)
    qed
    show ?thesis
      by (intro countable-subset[OF sub] countable-image countable-PiE S) auto
  qed
  show ?thesis
  proof
    assume fin:finite B
    have countable (ratvs.span B)
      using ratvs.span-finite[OF fin] ∗[OF fin] by simp
    thus False
      using assms uncountable-UNIV-real by simp
  qed
qed

lemma rat-Hamel-basisE:
  obtains B :: real set
  where module.independent (λq x. of-rat q ∗ x) B
    and module.span (λq x. of-rat q ∗ x) B = UNIV
    and 0 ∉ B
proof −
  interpret ratvs: vector-space λq x. of-rat q ∗ (x :: real)
    by unfold-locales (auto simp: of-rat-add of-rat-mult algebra-simps)
  obtain B where B:ratvs.independent B UNIV ⊆ ratvs.span B
    using ratvs.maximal-independent-subset[of UNIV] by auto
  hence sp:ratvs.span B = UNIV
    by auto
  have 0 ∉ B
    using B ratvs.dependent-zero by auto
  with B sp
  show ?thesis
    by (intro that) auto
qed

lemma lists-lepoll-infinite:
  assumes inf:infinite A
```

```
  shows lists A ≲ A
proof −
  define L where L ≡ λn. {l ∈ lists A. length l = n}
  have AA:A × A ≈ A
   using card-of-Times-same-infinite[OF inf] by (simp add: eqpoll-iff-card-of-ordIso)
  have L:L n ≲ A for n
  proof (induction n)
    case ih:(Suc n)
    have inj:inj-on (λl. (hd l, tl l)) (L (Suc n))
      by (auto simp: inj-on-def L-def)
        (metis hd-Cons-tl length-greater-0-conv zero-less-Suc)
    have img:(λl. (hd l, tl l)) ‘ L (Suc n) ⊆ A × L n
    proof (rule image-subsetI)
      fix l
      assume l:l ∈ L (Suc n)
      then obtain a l′ where l = a # l′
        by (cases l) (auto simp: L-def)
      with l show (hd l, tl l) ∈ A × L n
        by (auto simp: L-def)
    qed
    have L (Suc n) ≲ A × L n
      unfolding lepoll-def by (intro exI[of - λl. (hd l, tl l)] conjI inj img)
    also have A × L n ≲ A × A
      by (intro times-lepoll-mono lepoll-refl ih)
    also have A × A ≈ A
      by (rule AA)
    finally show ?case .
  qed (auto simp: L-def intro!: finite-lepoll-infinite inf)
  have lists A ≲ Sigma (UNIV :: nat set) L
    unfolding lepoll-def
    by (intro exI[of - λl. (length l, l)] conjI) (auto simp: inj-on-def L-def)
  also have ... ≲ (UNIV :: nat set) × A
    by (intro Sigma-lepoll-mono subset-refl L)
  also have ... ≲ A × A
    by (intro times-lepoll-mono lepoll-refl infinite-le-lepoll[THEN iffD1, OF inf])
  also have A × A ≈ A
    by (rule AA)
  finally show ?thesis .
qed

lemma UNIV-real-lepoll-basis-Diff:
  fixes u v :: real
  assumes sp:module.span (λq x. of-rat q ∗ x) B = UNIV
  shows (UNIV :: real set) ≲ B − {u, v}
proof −
  interpret ratvs: vector-space λq x. of-rat q ∗ (x :: real)
    by unfold-locales (auto simp: of-rat-add of-rat-mult algebra-simps)
  note inf = infinite-Hamel-basis[OF sp]
  have Binf:infinite (B − {u, v})
```

```
    by (intro Diff-infinite-finite inf) simp
  have (UNIV :: real set) ⊆ (λl. ∑(q, y)←l. of-rat q * y) ` lists (UNIV × B)
  proof
    fix x :: real
    have x ∈ ratvs.span B
      by (simp add: sp)
    then obtain t r where t-fin:finite t and tB:t ⊆ B
      and x:x = (∑ a∈t. of-rat (r a) * a)
      unfolding ratvs.span-explicit by blast
    obtain xs where xs:set xs = t and xs-dist:distinct xs
      using finite-distinct-list t-fin by blast
    have x = (∑(q, y)←map (λa. (r a, a)) xs. of-rat q * y)
      by (simp add: x xs xs-dist sum-list-distinct-conv-sum-set)
    moreover have map (λa. (r a, a)) xs ∈ lists (UNIV × B)
      using tB xs by (auto simp: in-lists-conv-set)
    ultimately show x ∈ (λl. ∑(q, y)←l. of-rat q * y) ` lists (UNIV × B)
      by blast
  qed
  hence (UNIV :: real set) ≲ lists ((UNIV :: rat set) × B)
    by (rule subset-image-lepoll[where f=λl. ∑(q, y)←l. of-rat q * y])
  also have lists ((UNIV :: rat set) × B) ≲ (UNIV :: rat set) × B
    using inf finite-cartesian-productD2 by (intro lists-lepoll-infinite) blast
  also have (UNIV :: rat set) × B ≲ B × B
  proof (intro times-lepoll-mono lepoll-refl)
    have (UNIV :: rat set) ≲ (UNIV :: nat set)
      unfolding lepoll-def by (intro exI[of - to-nat] conjI) simp-all
    thus (UNIV :: rat set) ≲ B
      using infinite-le-lepoll[THEN iffD1, OF inf] lepoll-trans by blast
  qed
  also have B × B ≈ B
    using card-of-Times-same-infinite[OF inf]
    by(simp add: eqpoll-iff-card-of-ordIso)
  also have ... ≲ insert u (insert v (B − {u, v}))
    by (intro subset-imp-lepoll) auto
  also have ... ≈ insert v (B − {u, v})
    using Binf by (intro infinite-insert-eqpoll) simp
  also have ... ≈ B − {u, v}
    using Binf by (rule infinite-insert-eqpoll)
  finally show ?thesis .
qed

corollary card-Hamel-basis:
  fixes B :: real set
  assumes sp: module.span (λq x. of-rat q * x) B = UNIV
  shows B ≈ (UNIV :: real set)
proof(rule lepoll-antisym)
  have (UNIV :: real set) ≲ B − {0, 0}
    by(intro UNIV-real-lepoll-basis-Diff sp)
  also have ... ≲ B
```

```
    by (simp add: subset-imp-lepoll)
  finally show (UNIV :: real set) ≲ B .
qed(simp add: subset-imp-lepoll)

lemma UNIV-fun-lepoll-nonlinear:
  (UNIV :: (real ⇒ real) set)
    ≲ {h :: real ⇒ real. (∀ x y. h (x + y) = h x + h y) ∧ (∀ c. h ≠ (λx. x * c))}
proof −
  interpret ratvsp: vector-space-pair
    λq x. of-rat q * (x :: real) λq x. of-rat q * (x :: real)
    by unfold-locales (auto simp: of-rat-add of-rat-mult algebra-simps)
  obtain B :: real set
    where indep:¬ module.dependent (λq x. of-rat q * x) B
      and span:module.span (λq x. of-rat q * x) B = UNIV and 0:0 ∉ B
    by (rule rat-Hamel-basisE)
  note inf = infinite-Hamel-basis[OF span]
  obtain b1 where b1:b1 ∈ B
    using inf all-not-in-conv by fastforce
  have infinite (B − {b1})
    using inf by (simp add: infinite-remove)
  then obtain b2 where b2:b2 ∈ B and b2-ne:b2 ≠ b1
    using infinite-imp-nonempty by blast
  obtain ι :: real ⇒ real where ι-inj:inj ι and ι-range:range ι ⊆ B − {b1, b2}
    using UNIV-real-lepoll-basis-Diff[OF span, of b1 b2]
    by (auto simp: lepoll-def)
  define Φ :: (real ⇒ real) ⇒ real ⇒ real where
    Φ φ = ratvsp.construct B (λb. if b = b1 then 1 else if b = b2 then 0 else φ (inv
ι b))
    for φ
  have add:Φ φ (x + y) = Φ φ x + Φ φ y for φ x y
     using ratvsp.linear-add[OF ratvsp.linear-construct[OF indep]] by (simp add:
Φ-def)
  have vb1:Φ φ b1 = 1 for φ
    using ratvsp.construct-basis[OF indep b1] by (simp add: Φ-def)
  have vb2:Φ φ b2 = 0 for φ
    using ratvsp.construct-basis[OF indep b2] b2-ne by (simp add: Φ-def)
  have vι:Φ φ (ι x) = φ x for φ x
  proof −
    have ιB:ι x ∈ B and ιb1:ι x ≠ b1 and ιb2:ι x ≠ b2
      using ι-range by auto
    show ?thesis
      using ratvsp.construct-basis[OF indep ιB] ιb1 ιb2 inv-f-f[OF ι-inj]
      by (simp add: Φ-def)
  qed
  have nonlin:Φ φ ≠ (λx. x * c) for φ c
  proof
    assume eq:Φ φ = (λx. x * c)
    have c = 0
      using fun-cong[OF eq, of b2] vb2[of φ] b2 0 by auto
```

thus *False*
using *fun-cong*[*OF eq*, *of b1*] *vb1*[*of* φ] **by** *simp*
qed
have *inj* Φ
using $v\iota$ **by** (*metis ext inj-def*)
moreover have *range* $\Phi \subseteq \{h.\ (\forall\, x\ y.\ h\ (x + y) = h\ x + h\ y) \wedge (\forall\, c.\ h \neq (\lambda x.\ x * c))\}$
by (*intro image-subsetI CollectI conjI allI add nonlin*)
ultimately show *?thesis*
unfolding *lepoll-def* **by** (*intro exI*[*of* - Φ] *conjI*)
qed

lemma *Cauchy-equation-nonlinear-solution-ex*:
$\exists h :: real \Rightarrow real.\ (\forall\, x\ y.\ h\ (x + y) = h\ x + h\ y) \wedge (\forall\, c.\ h \neq (\lambda x.\ x * c))$
proof −
obtain $\Phi :: (real \Rightarrow real) \Rightarrow real \Rightarrow real$
where *range* $\Phi \subseteq \{h.\ (\forall\, x\ y.\ h\ (x + y) = h\ x + h\ y) \wedge (\forall\, c.\ h \neq (\lambda x.\ x * c))\}$
using *UNIV-fun-lepoll-nonlinear* **by** (*auto simp*: *lepoll-def*)
thus *?thesis*
by *blast*
qed

lemma *Cauchy-equation-nonlinear-solution*:
obtains $h :: real \Rightarrow real$
where $\bigwedge x\ y.\ h\ (x + y) = h\ x + h\ y$
and $\bigwedge c.\ h \neq (\lambda x.\ x * c)$
using *Cauchy-equation-nonlinear-solution-ex that* **by** *blast*

2.2 Cardinality

lemma *card-Cauchy-equation-nonlinear-solutions*:
$\{h :: real \Rightarrow real.\ (\forall\, x\ y.\ h\ (x + y) = h\ x + h\ y) \wedge (\forall\, c.\ h \neq (\lambda x.\ x * c))\}$
$\approx (UNIV :: (real \Rightarrow real)\ set)$
by (*intro lepoll-antisym subset-imp-lepoll subset-UNIV UNIV-fun-lepoll-nonlinear*)

2.3 Density of the Graph of a Non-Linear Solution

lemma *Cauchy-equation-nonlinear-graph-indep*:
fixes $h :: real \Rightarrow real$
assumes $\bigwedge c.\ h \neq (\lambda x.\ x * c)$
obtains $x_1\ x_2$ **where** $x_1 * h\ x_2 - x_2 * h\ x_1 \neq 0$
using *assms* **by** (*metis mult-cancel-right2 right-minus-eq*)

theorem *Cauchy-equation-nonlinear-graph-dense*:
fixes $h :: real \Rightarrow real$
assumes *h-eq*:$\bigwedge x\ y.\ h\ (x + y) = h\ x + h\ y$
and *nonlin*:$\bigwedge c.\ h \neq (\lambda x.\ x * c)$
shows *closure* (*range* $(\lambda x.\ (x,\ h\ x))$) = *UNIV*
proof −
obtain $x_1\ x_2$ **where** *D*:$x_1 * h\ x_2 - x_2 * h\ x_1 \neq 0$

```
  using Cauchy-equation-nonlinear-graph-indep assms by blast
have key:∃ x. |x − a| < e ∧ |h x − b| < e if e:0 < e for a b e
proof −
  define M where M = |x1| + |x2| + |h x1| + |h x2| + 1
  have M-pos:0 < M
    unfolding M-def by (intro add-nonneg-pos add-nonneg-nonneg) auto
  have M1:|x1| + |x2| < M and M2:|h x1| + |h x2| < M
    by (auto simp: M-def)
  define δ where δ = e / M
  have δ-pos:0 < δ
    using e M-pos by (simp add: δ-def)
  define q1 where q1 = (a * h x2 − x2 * b) / (x1 * h x2 − x2 * h x1)
  define q2 where q2 = (x1 * b − a * h x1) / (x1 * h x2 − x2 * h x1)
  have quot:q1 * s1 + q2 * s2
          = ((a * h x2 − x2 * b) * s1 + (x1 * b − a * h x1) * s2)
            / (x1 * h x2 − x2 * h x1) for s1 s2
    by (simp add: q1-def q2-def add-divide-distrib)
  have a-eq:q1 * x1 + q2 * x2 = a and b-eq:q1 * h x1 + q2 * h x2 = b
    unfolding quot using D by (simp-all add: field-simps)
  obtain r1 where r1:r1 ∈ ℚ and r1':q1 < r1 r1 < q1 + δ
    using Rats-dense-in-real[of q1 q1 + δ] δ-pos by auto
  obtain r2 where r2:r2 ∈ ℚ and r2':q2 < r2 r2 < q2 + δ
    using Rats-dense-in-real[of q2 q2 + δ] δ-pos by auto
  have r1δ:|r1 − q1| < δ and r2δ:|r2 − q2| < δ
    using r1' r2' by auto
  have est:|(r1 − q1) * s1 + (r2 − q2) * s2| < e if s:|s1| + |s2| < M
    for s1 s2
  proof −
    have |(r1 − q1) * s1 + (r2 − q2) * s2| ≤ |r1 − q1| * |s1| + |r2 − q2| * |s2|
      by (metis abs-mult abs-triangle-ineq)
    also have ... ≤ δ * |s1| + δ * |s2|
      using r1δ r2δ by (intro add-mono mult-right-mono) auto
    also have ... = δ * (|s1| + |s2|)
      by (simp add: distrib-left)
    also have ... < δ * M
      using δ-pos s by (intro mult-strict-left-mono) auto
    also have ... = e
      using M-pos by (simp add: δ-def)
    finally show ?thesis .
  qed
  define x where x = r1 * x1 + r2 * x2
  have hx:h x = r1 * h x1 + r2 * h x2
    unfolding x-def h-eq[of r1 * x1 r2 * x2]
    by (simp add: Cauchy-equation-Q-linear[of h r1 x1, OF h-eq r1]
                  Cauchy-equation-Q-linear[of h r2 x2, OF h-eq r2])
  have |x − a| < e
    using est[OF M1]
    unfolding x-def a-eq[symmetric]
    by (simp add: algebra-simps)
```

```
    moreover have |h x − b| < e
      using est[OF M2]
      unfolding hx b-eq[symmetric]
      by (simp add: algebra-simps)
    ultimately show ?thesis
      by blast
  qed
  have z ∈ closure (range (λx. (x, h x))) for z
  proof (cases z)
    case (Pair a b)
    show ?thesis
      unfolding Pair closure-approachable
    proof (intro allI impI)
      fix e :: real
      assume e:0 < e
      then obtain x where x:|x − a| < e / 2 |h x − b| < e / 2
        using key[of e / 2 a b] by auto
      have dist (x, h x) (a, b) ≤ |x − a| + |h x − b|
        using sqrt-sum-squares-le-sum[of |x − a| |h x − b|]
        by (simp add: dist-Pair-Pair dist-real-def)
      also have ... < e
        using x by simp
      finally show ∃ y∈range (λx. (x, h x)). dist y (a, b) < e
        by (intro bexI[of - (x, h x)]) auto
    qed
  qed
  thus ?thesis
    by blast
qed

end
```

References

[1] G. Hamel. Eine Basis aller Zahlen und die unstetigen Lösungen der Funktionalgleichung $f(x + y) = f(x) + f(y)$. *Mathematische Annalen*, 60:459–462, 1905.