Extreme Value Theory

Michikazu Hirata

November 4, 2026

Abstract

This entry is a formalization of extreme value theory (EVT). EVT is a branch of probability theory and statistics that studies *rare events*. We formalize two main results of EVT, mainly following the books by de Haan and Ferreira [2] and by Embrechts et al. [3].

- The Fisher–Tippett–Gnedenko theorem, stating that if the properly normalized maximum of i.i.d. random variables converges in distribution to a non-degenerate limit, then the limit is an extreme value distribution.
- The Pickands–Balkema–de Haan theorem, stating that the conditional distribution of the excesses over a threshold u converges uniformly in the excess, with a suitable scale $\beta(u)$, to a generalized Pareto distribution as u tends to the right endpoint of the support, if and only if the underlying distribution belongs to the maximum domain of attraction of an extreme value distribution.

Contents

1 Lemmas **2**
- 1.1 Lemmas for Measure Theory . . . 2
- 1.2 Lemmas for Limits . . . 7
- 1.3 Lemmas for Weak Convergence . . . 14
- 1.4 Generalized Inverses . . . 15
- 1.5 Lemmas for Infinite Sums . . . 23

2 Distributions **46**
- 2.1 Generalized Extreme Value Distribution . . . 46
- 2.2 Generalized Pareto Distribution . . . 62

3 Domain of Attraction **79**
- 3.1 Definitions . . . 79
  - 3.1.1 Degenerate Distributions . . . 80
  - 3.1.2 Domain of Attraction . . . 81
- 3.2 Fisher–Tippett–Gnedenko Theorem . . . 85

3.3 Alternative Characterizations of the Domain of Attraction . . 111
3.3.1 Theorem 1.1.2 Part 111
3.3.2 Theorem 1.1.6 Part 122
3.4 von Mises' Conditions 141
3.5 Relation to Regularly Varying Functions 157

4 Peaks Over Threshold **191**
4.1 Excess (Peaks-Over-Threshold) Distribution Function 191
4.2 Pickands–Balkema–de Haan Theorem 197
4.2.1 Direct Half 197
4.2.2 Converse Half 198
4.2.3 Explicit Choice of β 221

5 Concrete Examples **230**
5.1 Maximum of Random Variables 231
5.2 Excesses of a Random Variable over a High Threshold 233
5.3 Domain of Attraction 238
5.3.1 Normal Distribution 239
5.3.2 Erlang Distribution 246
5.3.3 Uniform Distribution 253
5.3.4 Cauchy Distribution 253

1 Lemmas

theory *Lemmas-EVT*
imports
HOL−Probability.Probability
Regular-Variation.Regular-Variation
begin

lemma *power-of-nat-log-less-Suc*:
fixes *Z t* :: *real*
assumes *Z*: *1* < *Z*
shows *t* < *Z* ^ (*nat* ⌊*log Z t*⌋ + *1*)
using *assms le-nat-floor*[*OF le-log-of-power*, *of Z - t*]
by (*meson less-add-one not-less*)

1.1 Lemmas for Measure Theory

lemma *measure-density-on*:
assumes *integ*:*set-integrable M A f*
and [*measurable*]: *f* ∈ *borel-measurable M*
and [*measurable*]:*A* ∈ *sets M*
and *f-nonneg*:*AE x* ∈*A in M*. *0* ≤ *f x*
shows *measure* (*density M f*) *A* = (∫ *x*∈*A*. *f x* ∂*M*)
proof −
have *measure* (*density M f*) *A* = *enn2real* (∫⁺*x*∈*A*. *ennreal* (*f x*)∂*M*)

by(*simp add*: *measure-def emeasure-density*)
also have ... = *enn2real* ($\int^+ x$. *ennreal* (*f x*)∂(*restrict-space M A*))
by(*simp add*: *nn-integral-restrict-space*)
also have ... = *enn2real* (*ennreal* ($\int$ *x*. *f x* ∂(*restrict-space M A*)))
using *f-nonneg integ*
by(*subst nn-integral-eq-integral*)
(*auto simp*: *set-integrable-eq AE-restrict-space-iff*)
also have ... = *enn2real* (*ennreal* ($\int_{x \in A}$. *f x* ∂*M*))
by(*simp add*: *integral-restrict-space set-lebesgue-integral-def*)
also have ... = ($\int_{x \in A}$. *f x* ∂*M*)
using *f-nonneg*
by(*intro enn2real-ennreal*)
(*auto intro*!: *integral-nonneg-AE simp*: *set-lebesgue-integral-def*)
finally show *?thesis* **.**
qed

corollary *measure-density*:
assumes *integ*[*measurable*]:*integrable M f*
and [*measurable*]:*A* ∈ *sets M*
and *f-nonneg*:*AE x in M*. *0* ≤ *f x*
shows *measure* (*density M f*) *A* = ($\int_{x \in A}$. *f x* ∂*M*)
using *f-nonneg integrable-mult-indicator*[*OF - integ*]
by(*auto intro*!: *measure-density-on simp*: *set-integrable-def*)

lemma *nn-integral-FTC-greaterThan*:
fixes *f* :: *real* ⇒ *real* **and** *F* :: *real* ⇒ *real*
assumes *f-borel*: *f* ∈ *borel-measurable borel*
and *cont*: *continuous* (*at-right a*) *F*
and *f*: ⋀*x*. *a* < *x* ⟹ *DERIV F x* :> *f x*
and *nonneg*: ⋀*x*. *a* < *x* ⟹ *0* ≤ *f x*
and *lim*: (*F* ⟶ *T*) *at-top*
shows ($\int^+ x$. *ennreal* (*f x*) ∗ *indicator* {*a* <..} *x* ∂*lborel*) = *T* − *F a*
proof −
define *lo* **where** *lo n* = *a* + *1* / (*real n* + *1*) **for** *n* :: *nat*
define *hi* **where** *hi n* = *a* + (*real n* + *1*) **for** *n* :: *nat*
define *g* **where** *g n* = (λ*x*. *ennreal* (*f x*) ∗ *indicator* {*lo n* .. *hi n*} *x*) **for** *n* ::
nat
have *lo-gt*: ⋀*n*. *a* < *lo n* **and** *lo-le-hi*: ⋀*n*. *lo n* ≤ *hi n*
by (*auto simp*: *lo-def hi-def field-simps*)
have *g-eq*: *integral*^{N} *lborel* (*g n*) = *ennreal* (*F* (*hi n*) − *F* (*lo n*)) **for** *n*
unfolding *g-def*
by (*rule nn-integral-FTC-Icc*[*OF f-borel*]) (*use f nonneg lo-gt*[*of n*] *lo-le-hi*[*of n*]
in *auto*)
have *incseq-g*: *incseq g*
unfolding *incseq-def le-fun-def g-def*
proof (*intro allI impI*)
fix *m n* :: *nat* **and** *x* :: *real*
assume *m* ≤ *n*
hence *lo n* ≤ *lo m hi m* ≤ *hi n*

```
    by (auto simp: lo-def hi-def field-simps)
   thus ennreal (f x) * indicator {lo m..hi m} x ≤ ennreal (f x) * indicator {lo
n..hi n} x
    using nonneg[of x] lo-gt[of m]
    by (auto split: split-indicator)
 qed
 have lo-tendsto: lo ⟶ a
  unfolding lo-def by real-asymp
 have hi-tendsto: filterlim hi at-top sequentially
  unfolding hi-def by real-asymp
 have lo-at-right: filterlim lo (at-right a) sequentially
  using lo-tendsto lo-gt by (rule tendsto-imp-filterlim-at-right[OF - eventuallyI])
 have pointwise: (λn. g n x) ⟶ ennreal (f x) * indicator {a <..} x for x
 proof (cases a < x)
  case True
  have ∀F n in sequentially. lo n ≤ x
   using order-tendstoD(2)[OF lo-tendsto True] by (auto elim: eventually-mono)
  moreover have ∀F n in sequentially. x ≤ hi n
   using hi-tendsto[unfolded filterlim-at-top] by simp
  ultimately have ∀F n in sequentially. g n x = ennreal (f x) * indicator {a<..}
x
   by eventually-elim (use True in ‹auto simp: g-def split: split-indicator›)
  thus ?thesis
   by (rule tendsto-eventually)
 next
  case False
  hence g n x = 0 for n
   using lo-gt[of n] by (auto simp: g-def split: split-indicator)
  thus ?thesis
   using False by simp
 qed
 have limseq1: (λn. integral^N lborel (g n)) ⟶ (∫+x∈{a<..}. ennreal (f
x)∂lborel)
  using f-borel pointwise
  by (intro nn-integral-LIMSEQ incseq-g) (auto simp: g-def)
 have limseq2: (λn. integral^N lborel (g n)) ⟶ ennreal (T − F a)
 proof −
  have (λn. F (hi n) − F (lo n)) ⟶ T − F a
   using cont
   by (auto intro!: tendsto-diff filterlim-compose[OF - hi-tendsto]
       filterlim-compose[OF - lo-at-right] lim
       simp: continuous-within)
  from tendsto-ennrealI[OF this]
  show ?thesis
   by (simp add: g-eq)
 qed
 from LIMSEQ-unique[OF limseq1 limseq2]
 show ?thesis .
qed
```

lemma *nn-integral-FTC-lessThan*:
fixes $f :: real \Rightarrow real$ **and** $F :: real \Rightarrow real$
assumes *f-borel*: $f \in borel\text{-}measurable\ borel$
and *cont*: *continuous* (*at-left b*) *F*
and *f*: $\bigwedge x.\ x < b \Longrightarrow DERIV\ F\ x :> f\ x$
and *nonneg*: $\bigwedge x.\ x < b \Longrightarrow 0 \leq f\ x$
and *lim*: $(F \longrightarrow S)$ *at-bot*
shows $(\int^+ x.\ ennreal\ (f\ x) * indicator\ \{..< b\}\ x\ \partial lborel) = F\ b - S$
proof −
define *g* **where** $g\ x = f\ (-\ x)$ **for** $x :: real$
define *G* **where** $G\ x = -\ F\ (-\ x)$ **for** $x :: real$
note *f-borel*[*measurable*]
have *g-borel*: $g \in borel\text{-}measurable\ borel$
unfolding *g-def* **by** (*rule measurable-compose*[*OF - f-borel*]) *simp*
have *G-deriv*: $DERIV\ G\ x :> g\ x$ **if** $-b < x$ **for** *x*
proof −
have *hx*: $-\ x < b$
using *that* **by** *simp*
have *Fd*: (*F has-real-derivative f* $(-\ x)$) (*at* $(-\ x)$)
using *f*[*OF hx*] **.**
have *DERIV* $(\lambda z.\ F\ (-\ z))\ x :> f\ (-\ x) * (-\ 1)$
by (*rule DERIV-chain2*[**where** *g*=*uminus*, *OF Fd*]) (*auto intro*!: *derivative-eq-intros*)
with *DERIV-minus*[*OF this*]
show *?thesis*
unfolding *G-def g-def* **by** *simp*
qed
have *G-nonneg*: $0 \leq g\ x$ **if** $-b < x$ **for** *x*
unfolding *g-def* **using** *nonneg*[*of* $-x$] *that* **by** *simp*
have *G-cont*: *continuous* (*at-right* $(-b)$) *G*
using *cont*[*unfolded continuous-within*]
unfolding *G-def continuous-within*
by (*simp add*: *at-right-minus*[*of* $-b$] *filterlim-filtermap comp-def tendsto-minus*)
have *G-lim*: $(G \longrightarrow -\ S)$ *at-top*
unfolding *G-def*
using *filterlim-compose*[*OF lim filterlim-uminus-at-bot-at-top*]
by (*intro tendsto-minus*) *simp*
have *eq*: $(\int^+ x.\ ennreal\ (g\ x) * indicator\ \{-b <..\}\ x\ \partial lborel) = F\ b - S$
using *nn-integral-FTC-greaterThan*[*OF g-borel G-cont G-deriv G-nonneg G-lim*]
by (*simp add*: *G-def*)
have $(\int^+ x.\ ennreal\ (f\ x) * indicator\ \{..< b\}\ x\ \partial lborel)$
$= (\int^+ x.\ ennreal\ (f\ (0 + (-1) * x)) * indicator\ \{..< b\}\ (0 + (-1) * x)\ \partial lborel)$
by (*subst nn-integral-real-affine*[**where** *c*=-1 **and** *t*=*0*
and *f*=$\lambda x.$ *ennreal* (*f x*) * *indicator* {..<*b*} *x*]) *auto*
also have $\ldots = (\int^+ x.\ ennreal\ (g\ x) * indicator\ \{-b <..\}\ x\ \partial lborel)$
by (*intro nn-integral-cong*) (*auto simp*: *g-def split*: *split-indicator*)
finally show *?thesis*

using *eq* **by** *simp*
qed

lemma
fixes $f :: real \Rightarrow real$ **and** $F :: real \Rightarrow real$
assumes *f-borel*[*measurable*]: $f \in borel\text{-}measurable\ borel$
and *cont*: *continuous* (*at-right a*) *F*
and *f*: $\bigwedge x.\ a < x \Longrightarrow DERIV\ F\ x :> f\ x$
and *nonneg*: $\bigwedge x.\ a < x \Longrightarrow 0 \le f\ x$
and *lim*: $(F \longrightarrow T)\ at\text{-}top$
shows *set-integral-FTC-atLeast*:$(\int x{\in}\{a..\}.\ f\ x\ \partial lborel) = T - F\ a$
and *set-integrable-FTC-atLeast*:*set-integrable lborel* $\{a..\}\ f$
proof −
have *mono*:$F\ x \le F\ y$ **if** *ax*:$a < x$ **and** *xy*:$x \le y$ **for** *x y*
by(*intro DERIV-nonneg-imp-nondecreasing*[*OF xy*])
(*meson ax f nonneg order.strict-trans2*)
have *le-T*:$F\ x \le T$ **if** $a < x$ **for** *x*
using *mono*[*OF that*] **by**(*intro tendsto-lowerbound*[*OF lim*])
(*auto intro*!: *eventually-at-top-linorderI*[**where** *c*=*x*])
have *T-le*:$F\ a \le T$
using *cont le-T eventually-at-right-less*[*of a*]
by(*intro tendsto-upperbound*[**where** *F*=*at-right a*])
(*auto simp*: *continuous-within elim*!: *eventually-mono*)
have *1*:$(\int^+ x{\in}\{a<..\}.\ ennreal\ (f\ x)\partial lborel)$
$= (\int^+ x.\ ennreal\ |indicat\text{-}real\ \{a<..\}\ x * f\ x|\ \partial lborel)$
using *nonneg* **by**(*auto intro*!: *nn-integral-cong simp*: *indicator-def*)
show *set-integrable lborel* $\{a..\}\ f$
unfolding *set-integrable-atLeast-eq*[*symmetric*]
using *nn-integral-FTC-greaterThan*[*OF f-borel cont f nonneg lim*]
by(*simp add*: *set-integrable-def integrable-iff-bounded 1*)
have $(\int x{\in}\{a..\}.\ f\ x\ \partial lborel) = (\int x{\in}\{a<..\}.\ f\ x\ \partial lborel)$
by (*simp add*: *set-integral-atLeast-eq*)
also have $... = enn2real\ (\int^+ x{\in}\{a<..\}.\ ennreal\ (f\ x)\partial lborel)$
unfolding *set-lebesgue-integral-def* **using** *nonneg*
by(*subst integral-eq-nn-integral*)
(*auto simp*: *indicator-def intro*!: *arg-cong*[**where** *f*=*enn2real*] *nn-integral-cong*)
also have $... = enn2real\ (ennreal\ (T - F\ a))$
by(*simp add*: *nn-integral-FTC-greaterThan*[*OF f-borel cont f nonneg lim*])
also have $... = T - F\ a$
by(*simp add*: *T-le*)
finally show $(\int x{\in}\{a..\}.\ f\ x\ \partial lborel) = T - F\ a$ **.**
qed

lemma
fixes $f :: real \Rightarrow real$ **and** $F :: real \Rightarrow real$
assumes *f-borel*[*measurable*]: $f \in borel\text{-}measurable\ borel$
assumes *cont*: *continuous* (*at-left b*) *F*
assumes *f*: $\bigwedge x.\ x < b \Longrightarrow DERIV\ F\ x :> f\ x$
assumes *nonneg*: $\bigwedge x.\ x < b \Longrightarrow 0 \le f\ x$

assumes *lim*: $(F \longrightarrow S)$ *at-bot*
shows *set-integral-FTC-atMost*:$(\int x{\in}\{..b\}.\ f\ x\ \partial lborel) = F\ b - S$
and *set-integrable-FTC-atMost*:*set-integrable lborel* $\{..b\}$ *f*
proof −
have *mono*:$F\ x \le F\ y$ **if** *xy*:$x \le y$ **and** *yb*:$y < b$ **for** *x y*
by(*intro DERIV-nonneg-imp-nondecreasing*[*OF xy*])
(*meson yb f nonneg order.strict-trans1*)
have *S-ge*:$S \le F\ x$ **if** $x < b$ **for** *x*
using *mono*[*OF - that*] **by**(*intro tendsto-upperbound*[*OF lim*])
(*auto intro*!: *eventually-at-bot-linorderI*[**where** *c*=*x*])
have *b-left*:$\forall_F\ x\ in\ at\text{-}left\ b.\ x < b$
by(*simp add*: *eventually-at-filter*)
have *S-le*:$S \le F\ b$
using *cont S-ge b-left*
by(*intro tendsto-lowerbound*[**where** *F*=*at-left b*])
(*auto simp*: *continuous-within elim*!: *eventually-mono*)
have *1*:$(\int^+ x{\in}\{..<b\}.\ ennreal\ (f\ x)\partial lborel)$
$= (\int^+ x.\ ennreal\ |indicat\text{-}real\ \{..<b\}\ x * f\ x|\ \partial lborel)$
using *nonneg* **by**(*auto intro*!: *nn-integral-cong simp*: *indicator-def*)
show *set-integrable lborel* $\{..b\}$ *f*
unfolding *set-integrable-atMost-eq*[*symmetric*]
using *nn-integral-FTC-lessThan*[*OF f-borel cont f nonneg lim*]
by(*simp add*: *set-integrable-def integrable-iff-bounded 1*)
have $(\int x{\in}\{..b\}.\ f\ x\ \partial lborel) = (\int x{\in}\{..<b\}.\ f\ x\ \partial lborel)$
by (*simp add*: *set-integral-atMost-eq*)
also have ... = *enn2real* $(\int^+ x{\in}\{..<b\}.\ ennreal\ (f\ x)\partial lborel)$
unfolding *set-lebesgue-integral-def* **using** *nonneg*
by(*subst integral-eq-nn-integral*)
(*auto simp*: *indicator-def intro*!: *arg-cong*[**where** *f*=*enn2real*] *nn-integral-cong*)
also have ... = *enn2real* (*ennreal* $(F\ b - S)$)
by(*simp add*: *nn-integral-FTC-lessThan*[*OF f-borel cont f nonneg lim*])
also have ... = $F\ b - S$
by(*simp add*: *S-le*)
finally show $(\int x{\in}\{..b\}.\ f\ x\ \partial lborel) = F\ b - S$ **.**
qed

1.2 Lemmas for Limits

lemma *filterlim-log-at-top-gt-1*:
fixes *a* :: *real*
assumes *a*:$1 < a$
shows *filterlim* (*log a*) *at-top at-top*
unfolding *filterlim-at-top*
proof *safe*
fix *Z*
show $\forall_F\ x\ in\ at\text{-}top.\ Z \le log\ a\ x$
by(*auto intro*!: *eventually-at-top-linorderI*[**where** *c*=*max* (*a powr Z*) *1*]
le-log-iff[*THEN iffD2*] *assms*)
qed

lemma *LIMSEQ-power-at-top*:
fixes $c :: real$
assumes $1 < c$
shows *filterlim* $(\lambda n.\ c \mathbin{\hat{}} n)$ *at-top sequentially*
using *Archimedean-eventually-pow assms filterlim-at-top-dense* **by** *blast*

lemma *floor-log-at-top*:
fixes $Z :: real$
assumes Z: $1 < Z$
shows *filterlim* $(\lambda t.\ nat\ \lfloor log\ Z\ t \rfloor)$ *at-top at-top*
by(*intro filterlim-compose*[*OF filterlim-nat-sequentially*]
filterlim-compose[*OF filterlim-floor-sequentially*] *filterlim-log-at-top-gt-1 Z*)

lemma *squeeze-tendsto-cont-points*:
fixes $f :: real \Rightarrow real$ **and** $h :: 'a \Rightarrow real$ **and** $gr :: real \Rightarrow 'a \Rightarrow real$
and $F :: 'a\ filter$ **and** $y :: real$
and $A :: real\ set$
assumes *mono*: *mono-on A f*
and *y*:$y \in interior\ A$
and *cont*: *isCont f y*
and *conv*: $\bigwedge y'.\ y' \in A \Longrightarrow isCont\ f\ y' \Longrightarrow ((gr\ y') \longrightarrow f\ y')\ F$
and *low*: $\bigwedge y'.\ y' < y \Longrightarrow y' \in A \Longrightarrow \forall_F\ t\ in\ F.\ gr\ y'\ t \leq h\ t$
and *up*: $\bigwedge y'.\ y < y' \Longrightarrow y' \in A \Longrightarrow \forall_F\ t\ in\ F.\ h\ t \leq gr\ y'\ t$
shows $(h \longrightarrow f\ y)\ F$
proof −
obtain A' **where** A':*open* $A'\ y \in A'\ A' \subseteq A$
using *y interiorE* **by** *blast*
have *cd*: *countable* $\{s \in A'.\ \neg\ isCont\ f\ s\}$
by (*intro mono-on-ctble-discont-open*) (*auto intro*!: *mono-onI mono-on-subset*[*OF mono*] A')
obtain $\delta 1$ **where** $\delta 1$: $0 < \delta 1\ \bigwedge x.\ dist\ x\ y < \delta 1 \Longrightarrow x \in A'$
using $A'(1,2)$ *open-dist* **by** *blast*
show *?thesis*
unfolding *tendsto-iff dist-real-def*
proof *safe*
fix $e :: real$
assume e: $0 < e$
obtain $\delta 0$ **where** $\delta 0$: $0 < \delta 0$
$\bigwedge y''.\ y'' \neq y \wedge |y'' - y| < \delta 0 \Longrightarrow |f\ y'' - f\ y| < e$
using *LIM-D*[*OF cont*[*unfolded isCont-def*], *of e*] *e* **by** *auto*
define δ **where** $\delta = min\ \delta 0\ \delta 1\ /\ 2$
have δpos: $0 < \delta$ **and** δlt: $\delta < \delta 0\ \delta < \delta 1$
using $\delta 0(1)\ \delta 1(1)$ **by** (*auto simp*: δ-*def*)
obtain *ylo* **where** *ylo*: $ylo \in \{y - \delta <..< y\}$ $ylo \notin \{s \in A'.\ \neg\ isCont\ f\ s\}$
using *real-interval-avoid-countable-set*[*OF - cd*, *of* $y - \delta\ y$] δpos **by** *auto*
obtain *yhi* **where** *yhi*: $yhi \in \{y <..< y + \delta\}$ $yhi \notin \{s \in A'.\ \neg\ isCont\ f\ s\}$
using *real-interval-avoid-countable-set*[*OF - cd*, *of* $y\ y + \delta$] δpos **by** *auto*
have *ylo-lt*: $ylo < y$ **and** *yhi-gt*: $y < yhi$

```
    using ylo(1) yhi(1) by auto
  have ylo-A': ylo ∈ A' and yhi-A': yhi ∈ A'
    using δ1(2) ylo(1) yhi(1) δlt(2) by (auto simp: dist-real-def)
  have ylo-A: ylo ∈ A and yhi-A: yhi ∈ A
    using ylo-A' yhi-A' A'(3) by auto
  have ylo-cont: isCont f ylo and yhi-cont: isCont f yhi
    using ylo(2) yhi(2) ylo-A' yhi-A' by auto
  have f y − e < f ylo
    using δ0(2)[of ylo] ylo(1) δlt(1) by auto
  hence lo: ∀F t in F. f y − e < gr ylo t
    by (intro order-tendstoD(1)[OF conv[OF ylo-A ylo-cont]])
  have f yhi < f y + e
    using δ0(2)[of yhi] yhi(1) δlt(1) by auto
  hence hi: ∀F t in F. gr yhi t < f y + e
    by (intro order-tendstoD(2)[OF conv[OF yhi-A yhi-cont]])
  show ∀F t in F. |h t − f y| < e
    using lo hi low[OF ylo-lt ylo-A] up[OF yhi-gt yhi-A]
    by eventually-elim (simp add: abs-less-iff)
 qed
qed

lemma squeeze-tendsto-local:
 fixes f :: real ⇒ real and h :: 'a ⇒ real and gr :: real ⇒ 'a ⇒ real
   and F :: 'a filter and y η :: real
 assumes eta: 0 < η
   and cont: isCont f y
   and conv: ⋀y'. |y' − y| < η ⟹ ((gr y') ⟶ f y') F
   and low: ⋀y'. y − η < y' ⟹ y' < y ⟹ ∀F t in F. gr y' t ≤ h t
   and up: ⋀y'. y < y' ⟹ y' < y + η ⟹ ∀F t in F. h t ≤ gr y' t
 shows (h ⟶ f y) F
 unfolding tendsto-iff dist-real-def
proof safe
 fix e :: real
 assume e: 0 < e
 obtain δ0 where δ0: 0 < δ0
   ⋀y''. y'' ≠ y ∧ |y'' − y| < δ0 ⟹ |f y'' − f y| < e
   using LIM-D[OF cont[unfolded isCont-def], of e] e by auto
 define δ where δ = min δ0 η / 2
 have δpos: 0 < δ and δlt: δ < η δ < δ0
   using δ0(1) eta by (auto simp: δ-def)
 define ylo yhi where ylo = y − δ and yhi = y + δ
 have ylo1: y − η < ylo ylo < y ylo ≠ y |ylo − y| < δ0 |ylo − y| < η
   using δpos δlt by (auto simp: ylo-def)
 have yhi1: y < yhi yhi < y + η yhi ≠ y |yhi − y| < δ0 |yhi − y| < η
   using δpos δlt by (auto simp: yhi-def)
 have f y − e < f ylo
   using δ0(2)[of ylo] ylo1(3,4) by auto
 hence lo: ∀F t in F. f y − e < gr ylo t
   by (intro order-tendstoD(1)[OF conv[OF ylo1(5)]])
```

have *f yhi* < *f y* + *e*
using *δ0*(*2*)[*of yhi*] *yhi1*(*3*,*4*) **by** *auto*
hence *hi*: ∀ *F t in F*. *gr yhi t* < *f y* + *e*
by (*intro order-tendstoD*(*2*)[*OF conv*[*OF yhi1*(*5*)]])
show ∀ *F t in F*. |*h t* − *f y*| < *e*
using *lo hi low*[*OF ylo1*(*1*,*2*)] *up*[*OF yhi1*(*1*,*2*)]
by *eventually-elim* (*simp add*: *abs-less-iff*)
qed

corollary *squeeze-tendsto-cont-points-pos*:
fixes *f* :: *real* ⇒ *real* **and** *h* :: *'a* ⇒ *real* **and** *gr* :: *real* ⇒ *'a* ⇒ *real*
and *F* :: *'a filter* **and** *y* :: *real*
assumes *mono*: *mono-on* {*0*<..} *f*
and *y*: *0* < *y* **and** *cont*: *isCont f y*
and *conv*: ⋀*r*. *0* < *r* ⟹ *isCont f r* ⟹ ((*gr r*) ⟶ *f r*) *F*
and *low*: ⋀*r*. *0* < *r* ⟹ *r* < *y* ⟹ ∀ *F t in F*. *gr r t* ≤ *h t*
and *up*: ⋀*r*. *y* < *r* ⟹ ∀ *F t in F*. *h t* ≤ *gr r t*
shows (*h* ⟶ *f y*) *F*
using *assms*
by(*auto intro*!: *squeeze-tendsto-cont-points*
[**where** *f*=*f* **and** *F*=*F* **and** *h*=*h* **and** *gr*=*gr* **and** *A*={*0*<..}] *simp*: *interior-open*)

corollary *squeeze-tendsto-cont-points-UNIV*:
fixes *f* :: *real* ⇒ *real* **and** *h* :: *'a* ⇒ *real* **and** *gr* :: *real* ⇒ *'a* ⇒ *real*
and *F* :: *'a filter* **and** *y* :: *real*
assumes *mono*: *mono f*
and *cont*: *isCont f y*
and *conv*: ⋀*y'*. *isCont f y'* ⟹ ((*gr y'*) ⟶ *f y'*) *F*
and *low*: ⋀*y'*. *y'* < *y* ⟹ ∀ *F t in F*. *gr y' t* ≤ *h t*
and *up*: ⋀*y'*. *y* < *y'* ⟹ ∀ *F t in F*. *h t* ≤ *gr y' t*
shows (*h* ⟶ *f y*) *F*
using *assms*
by(*auto intro*!: *squeeze-tendsto-cont-points*
[**where** *f*=*f* **and** *F*=*F* **and** *h*=*h* **and** *gr*=*gr* **and** *A*=*UNIV*] *simp*: *interior-open*)

lemma *left-cont-agree-cocountable-eq*:
fixes *f h* :: *real* ⇒ *real* **and** *N* :: *real set* **and** *y* :: *real*
assumes *N*: *countable N* **and** *y*: *0* < *y*
and *left-cont*: (*f* ⟶ *f y*) (*at-left y*)
and *left-cont-h*: (*h* ⟶ *h y*) (*at-left y*)
and *eq*: ⋀*x*. *0* < *x* ⟹ *x* < *y* ⟹ *x* ∉ *N* ⟹ *f x* = *h x*
shows *f y* = *h y*
proof −
define *P* **where** *P* = (λ*n z*. *z* ∈ {*max 0* (*y* − *1* / (*real n* + *1*))<..<*y*} ∧ *z* ∉ *N*)
have *exP*: ∃ *z*. *P n z* **for** *n*
unfolding *P-def*

```
    using real-interval-avoid-countable-set[OF - N, of max 0 (y − 1 / (real n +
1)) y]
      y by auto
  define zseq where zseq = (λn. SOME z. P n z)
  have zseqP: P n (zseq n) for n
    unfolding zseq-def using exP by (rule someI-ex)
  have z-pos: 0 < zseq n and z-lt: zseq n < y and z-notin: zseq n ∉ N for n
    using zseqP by (auto simp: P-def)
  have z-gt: y − 1 / (real n + 1) < zseq n for n
    using zseqP by (auto simp: P-def)
  have diff-bound: y − zseq n < 1 / (real n + 1) for n
    using z-gt by (simp add: cross3-simps)
  have diff-pos: 0 ≤ y − zseq n for n
    using z-lt less-eq-real-def by auto
  have (λn. y − zseq n) ⟶ 0
  proof (rule tendsto-sandwich)
    show ∀F n in sequentially. 0 ≤ y − zseq n
      using diff-pos by simp
    show ∀F n in sequentially. y − zseq n ≤ 1 / (real n + 1)
      using diff-bound by (simp add: less-le-not-le)
    show (λn. 1 / (real n + 1)) ⟶ (0::real)
     using LIMSEQ-inverse-real-of-nat by (simp add: inverse-eq-divide add.commute)
  qed simp
  from tendsto-diff[OF tendsto-const this, of y]
  have z-tendsto: (λn. zseq n) ⟶ y
    by simp
  have z-at-left: filterlim zseq (at-left y) sequentially
    by (rule tendsto-imp-filterlim-at-left[OF z-tendsto]) (simp add: z-lt)
  have f-lim: (λn. f (zseq n)) ⟶ f y
    using filterlim-compose[OF left-cont z-at-left] by (simp add: comp-def)
  have h-lim: (λn. h (zseq n)) ⟶ h y
    using filterlim-compose[OF left-cont-h z-at-left] by (simp add: comp-def)
  have f (zseq n) = h (zseq n) for n
    using eq[OF z-pos z-lt z-notin] .
  with h-lim have (λn. f (zseq n)) ⟶ h y
    by simp
  from LIMSEQ-unique[OF f-lim this] show ?thesis .
qed

lemma power-tendsto-iff-tendsto-tail:
  fixes p :: nat ⇒ real and q :: real
  assumes p-nonneg: ⋀n. 0 ≤ p n and p-le1: ⋀n. p n ≤ 1
    and q-pos: 0 < q and q-le1: q ≤ 1
  shows (λn. p n ^ n) ⟶ q ⟷ (λn. real n * (1 − p n)) ⟶ − ln q
proof −
  have lb: 1 − p n ≤ − ln (p n) if 0 < p n for n
    using ln-le-minus-one[OF that] by simp
  have ub: p n * (− ln (p n)) ≤ 1 − p n if 0 < p n for n
  proof −
```

```
  have ln (1 / p n) ≤ 1 / p n − 1
    using that by (intro ln-le-minus-one) simp
  hence − ln (p n) ≤ (1 − p n) / p n
    using that by (simp add: ln-div field-simps)
  thus ?thesis
    using that by (simp add: field-simps)
 qed
 show ?thesis
 proof
  assume T: (λn. real n ∗ (1 − p n)) ⟶ − ln q
  have p-to-1: (λn. p n) ⟶ 1
  proof −
   have (λn. (real n ∗ (1 − p n)) ∗ (1 / real n)) ⟶ 0
    using tendsto-mult[OF T lim-1-over-n] by simp
   moreover have ∀F n in sequentially. (real n ∗ (1 − p n)) ∗ (1 / real n) =
1 − p n
    using eventually-gt-at-top[of 0] by eventually-elim auto
   ultimately have (λn. 1 − p n) ⟶ 0
    by (rule Lim-transform-eventually)
   hence (λn. 1 − (1 − p n)) ⟶ 1 − 0
    by (intro tendsto-diff tendsto-const)
   thus ?thesis
    by simp
  qed
  have ev-pos: ∀F n in sequentially. 1 / 2 < p n
   by (rule order-tendstoD(1)[OF p-to-1]) simp
  have nlnp: (λn. real n ∗ (− ln (p n))) ⟶ − ln q
  proof (rule tendsto-sandwich)
   show ∀F n in sequentially. real n ∗ (1 − p n) ≤ real n ∗ (− ln (p n))
    using ev-pos by eventually-elim (fastforce intro!: mult-left-mono lb)
   show ∀F n in sequentially. real n ∗ (− ln (p n)) ≤ real n ∗ (1 − p n) / p n
    using ev-pos
    by eventually-elim
      (use mult-left-mono[OF ub, of - real n for n] in ‹simp add: field-simps›)
   show (λn. real n ∗ (1 − p n)) ⟶ − ln q
    by fact
   show (λn. real n ∗ (1 − p n) / p n) ⟶ − ln q
    using tendsto-divide[OF T p-to-1] by simp
  qed
  have ∀F n in sequentially. exp (real n ∗ ln (p n)) = p n ^ n
   using ev-pos by eventually-elim (simp add: exp-of-nat-mult)
  moreover have (λn. exp (real n ∗ ln (p n))) ⟶ exp (ln q)
   using tendsto-minus[OF nlnp] by (intro tendsto-exp) simp
  ultimately show (λn. p n ^ n) ⟶ q
   using q-pos by (auto intro: Lim-transform-eventually)
 next
  assume P: (λn. p n ^ n) ⟶ q
  have p-to-1: (λn. p n) ⟶ 1
  proof (rule tendsto-sandwich)
```

```
  have ev1: ∀F n in sequentially. q / 2 < p n ^ n
    by (rule order-tendstoD(1)[OF P]) (use q-pos q-le1 in simp)
  hence ev2: ∀F n in sequentially. q / 2 < p n ^ n ∧ 0 < n
    using eventually-gt-at-top[of 0] by (auto intro: eventually-conj)
  show ∀F n in sequentially. (q / 2) powr (1 / real n) ≤ p n
    using ev2
  proof eventually-elim
    case (elim n)
    hence pn-pos: 0 < p n
      using q-pos p-nonneg[of n] by (cases p n = 0) (auto simp: zero-power)
    have (q / 2) powr (1 / real n) ≤ (p n ^ n) powr (1 / real n)
      using elim q-pos by (intro powr-mono2) auto
   also have (p n ^ n) powr (1 / real n) = (p n powr real n) powr (1 / real n)
      using pn-pos by (simp add: powr-realpow)
    also have ... = p n
      using elim pn-pos by (simp add: powr-powr)
    finally show ?case .
  qed
  show ∀F n in sequentially. p n ≤ 1
    using p-le1 by simp
  have (λn. (q / 2) powr (1 / real n)) ⟶ (q / 2) powr 0
    using q-pos by (intro tendsto-powr tendsto-const lim-1-over-n) auto
  thus (λn. (q / 2) powr (1 / real n)) ⟶ 1
    using q-pos q-le1 by simp
  show (λn. 1::real) ⟶ 1
    by simp
qed
have ev-pos: ∀F n in sequentially. 1 / 2 < p n
  by (rule order-tendstoD(1)[OF p-to-1]) simp
have nlnp: (λn. real n * (− ln (p n))) ⟶ − ln q
proof −
  have (λn. ln (p n ^ n)) ⟶ ln q
    using q-pos by (intro isCont-tendsto-compose[OF - P] isCont-ln) simp
  moreover have ∀F n in sequentially. ln (p n ^ n) = real n * ln (p n)
    using ev-pos by eventually-elim (simp add: ln-realpow)
  ultimately have (λn. real n * ln (p n)) ⟶ ln q
    by (simp add: filterlim-cong)
  thus ?thesis
    by(auto dest: tendsto-minus)
qed
show (λn. real n * (1 − p n)) ⟶ − ln q
proof (rule tendsto-sandwich)
 show ∀F n in sequentially. p n * (real n * (− ln (p n))) ≤ real n * (1 − p n)
    using ev-pos
    by eventually-elim
      (use mult-left-mono[OF ub, of - real n for n] in ‹simp add: field-simps›)
  show ∀F n in sequentially. real n * (1 − p n) ≤ real n * (− ln (p n))
    using ev-pos by eventually-elim (fastforce intro!: mult-left-mono lb)
qed(use tendsto-mult[OF p-to-1 nlnp] nlnp in auto)
```

qed
qed

lemma *filterlim-shift-scale-at-top*:
fixes μ σ ξ :: *real*
assumes [*arith*]:$0 < \sigma$ $0 < \xi$
shows *LIM x at-top.* $1 + \xi * ((x - \mu) / \sigma)$:> *at-top*
using *filterlim-tendsto-pos-mult-at-top*[*OF tendsto-const*[*of* $1 / \sigma$]
- *filterlim-tendsto-add-at-top*[*OF tendsto-const*[*of* $-\mu$ *at-top*],*of* $\lambda x.\ x$]]
by(*intro filterlim-tendsto-add-at-top*[*OF tendsto-const*]
filterlim-tendsto-pos-mult-at-top[*OF tendsto-const*])
(*auto simp*: *filterlim-ident*)

lemma *filterlim-shift-scale-at-bot*:
fixes μ σ ξ :: *real*
assumes *sigma*: $0 < \sigma$ **and** *xi-neg*: $\xi < 0$
shows *LIM x at-bot.* $1 + \xi * ((x - \mu) / \sigma)$:> *at-top*
proof −
have *eq*: $(\lambda x.\ 1 + \xi * ((x - \mu) / \sigma)) = (\lambda x.\ 1 + (-\xi / \sigma) * (\mu - x))$
by (*rule ext*) (*use sigma* **in** ‹*simp add*: *field-simps*›)
have *LIM x at-bot.* $\mu - x$:> *at-top*
by *real-asymp*
hence *LIM x at-bot.* $(-\xi / \sigma) * (\mu - x)$:> *at-top*
by (*intro filterlim-tendsto-pos-mult-at-top*[*OF tendsto-const*])
(*use assms* **in** ‹*auto intro*: *divide-neg-pos*›)
hence *LIM x at-bot.* $1 + (-\xi / \sigma) * (\mu - x)$:> *at-top*
by (*intro filterlim-tendsto-add-at-top*[*OF tendsto-const*])
thus *?thesis*
unfolding *eq* **.**
qed

1.3 Lemmas for Weak Convergence

lemma *weak-conv-ignore-initial-segment*:
assumes *weak-conv a A*
shows *weak-conv* $(\lambda n.\ a\ (n + k))\ A$
using *assms* **by**(*auto simp*: *weak-conv-def intro*!: *LIMSEQ-ignore-initial-segment*)

lemma *weak-conv-offset*:
assumes *weak-conv* $(\lambda n.\ a\ (n + k))\ A$
shows *weak-conv a A*
using *assms LIMSEQ-offset* **by**(*fastforce simp*: *weak-conv-def*)

lemma *weak-conv-cong*:
assumes *ev*:$\bigwedge x.$ *isCont A x* $\Longrightarrow \forall_F\ n$ *in sequentially. a n x* = *b n x*
shows *weak-conv a A* $\longleftrightarrow$ *weak-conv b A*
using *tendsto-cong*[*OF ev*] **by**(*simp add*: *weak-conv-def*)

lemma *LIMSEQ-imp-weak-conv*:

$(\bigwedge x.\ (\lambda n.\ a\ n\ x) \longrightarrow A\ x) \Longrightarrow$ *weak-conv a A*
by(*simp add*: *weak-conv-def*)

1.4 Generalized Inverses

definition *ginv* :: ('*a* :: *conditionally-complete-linorder* ⇒ '*a*) ⇒ '*a* ⇒ '*a* (-$^{\leftarrow}$ [*81*])
where
ginv F y = *Inf* {*x*. *y* ≤ *F x*}

The constant *ginv* is the same as *right-continuous-mono.I*. We define *ginv* because *right-continuous-mono.I* is defined inside the locale *right-continuous-mono*, which makes it inconvenient to use.

lemma *bdd-below-ginv-set*:
assumes *mono*:*mono F* **and** *Fw*:*F w* < *y*
shows *bdd-below* {*x*::- :: *linorder*. *y* ≤ *F x*}
proof (*intro bdd-belowI*[*of* - *w*])
fix *x*
assume *x*:*x* ∈ {*x*. *y* ≤ *F x*}
show *w* ≤ *x*
using *x monoD*[*OF mono*, *of x w*] *Fw* **by** *force*
qed

lemma *ginv-le-iff1*:
fixes *y* :: '*a*::{*linorder-topology*,*conditionally-complete-linorder*}
assumes *mono*: *mono F* **and** *cont*: $\bigwedge x.$ *continuous* (*at-right x*) *F*
and *yFz*: *y* ≤ *F z* **and** *Fwy*: *F w* < *y*
shows *F*$^{\leftarrow}$ *y* ≤ *x* ⟷ *y* ≤ *F x*
proof
have *ne*: {*x*. *y* ≤ *F x*} ≠ {}
using *yFz* **by** *auto*
show *y* ≤ *F x* ⟹ *ginv F y* ≤ *x*
unfolding *ginv-def* **by** (*auto intro*!: *cInf-lower bdd-below-ginv-set*[*OF mono Fwy*])
assume ∗: *ginv F y* ≤ *x*
have *y* ≤ (*INF s*∈{*x*. *y* ≤ *F x*}. *F s*)
by (*rule cINF-greatest*[*OF ne*]) *auto*
also have ... = *F* (*Inf* {*x*. *y* ≤ *F x*})
using *continuous-at-Inf-mono*[*OF mono cont ne bdd-below-ginv-set*[*OF mono Fwy*]] **..**
also have ... ≤ *F x*
using ∗ *monoD*[*OF mono*] **by** (*simp add*: *ginv-def*)
finally show *y* ≤ *F x* **.**
qed

lemma *ginv-mono*:
assumes *mono*: *mono F* **and** *y12*: *y1* ≤ *y2*
and *ex-z*: ∃ *z*. *y2* ≤ *F z* **and** *ex-w*: ∃ *w*. *F w* < *y1*
shows *ginv F y1* ≤ *ginv F y2*
proof −

obtain z w **where** z: $y2 \leq F\ z$ **and** w: $F\ w < y1$
using *ex-z ex-w* **by** *auto*
have *ne*: $\{x.\ y2 \leq F\ x\} \neq \{\}$
using z **by** *auto*
have *sub*: $\{x.\ y2 \leq F\ x\} \subseteq \{x.\ y1 \leq F\ x\}$
using *y12* **by** *auto*
show *?thesis*
unfolding *ginv-def* **by** (*rule cInf-superset-mono*[*OF ne bdd-below-ginv-set*[*OF mono w*] *sub*])
qed

lemma *ginv-compose-mono*:
assumes *mono*: *mono F* **and** *monog*:*mono-on A g*
and *ex*:$\forall\, s{\in}A.\ \exists\, z.\ g\ s \leq F\ z\ \forall\, r{\in}A.\ \exists\, w.\ F\ w < g\ r$
shows *mono-on A* ($\lambda x.$ *ginv F* (*g x*))
using *ex*
by(*intro ginv-mono mono-onI mono-onD*[*OF monog*] *monoD*[*OF mono*])
auto

lemma *ginv-attained1*:
fixes y :: *'a*::{*linorder-topology*,*conditionally-complete-linorder*}
assumes *mono*: *mono F* **and** *cont*: $\bigwedge z.$ *continuous* (*at-right z*) *F*
and *ex-z*: $\exists\, z.\ y \leq F\ z$ **and** *ex-w*: $\exists\, w.\ F\ w < y$
shows $y \leq F$ (*ginv F y*)
proof −
obtain z w **where** $y \leq F\ z\ F\ w < y$
using *ex-z ex-w* **by** *auto*
thus *?thesis*
using *ginv-le-iff1*[*OF mono cont*, *of y z w ginv F y*]
by *simp*
qed

Convergence of generalized inverses (Lemma 1.1.1 of the book by de Haan and Ferreira [2]).

lemma *tendsto-ginv*:
fixes *Fn* :: *nat* ⇒ *real* ⇒ *real* **and** *g* :: *real* ⇒ *real* **and** *y* :: *real*
assumes *mono-Fn*: $\bigwedge n.$ *mono* (*Fn n*) **and** *mono-g*: *mono g*
and *wc'*:*weak-conv Fn g*
and *cont-ginv*: *isCont* (*ginv g*) *y*
and *yz*: $y < g\ z$ **and** *wy*: $g\ w < y$
shows $(\lambda n.\ (Fn\ n)^{\leftarrow}\ y) \longrightarrow g^{\leftarrow}\ y$
proof (*rule metric-LIMSEQ-I*)
have *wc*:$\bigwedge x.$ *isCont g x* ⟹ $(\lambda n.\ Fn\ n\ x) \longrightarrow g\ x$
using *wc'* **by**(*simp add*: *weak-conv-def*)
fix ε :: *real*
assume *eps*: $0 < \varepsilon$
define *x0* **where** *x0* = *ginv g y*
note *cd* = *mono-ctble-discont*[*OF mono-g*]
have *bdd*: *bdd-below* $\{x.\ y \leq g\ x\}$

```
    using wy by (intro bdd-below-ginv-set[OF mono-g])
  have ne: {x. y ≤ g x} ≠ {}
    using yz by (auto intro!: exI[of - z] less-imp-le)
  obtain x1 where x1: x1 ∈ {x0 − ε<..<x0} isCont g x1
    using real-interval-avoid-countable-set[OF - cd, of x0 − ε x0] eps by auto
  have gx1: g x1 < y
  proof (rule ccontr)
    assume ¬ g x1 < y
    hence x0 ≤ x1
      unfolding x0-def ginv-def by (auto intro!: cInf-lower bdd)
    with x1(1) show False
      by auto
  qed
  obtain s where s: 0 < s
    ⋀y'. y' ≠ y ∧ |y' − y| < s ⟹ |ginv g y' − ginv g y| < ε / 2
    using LIM-D[OF cont-ginv[unfolded isCont-def], of ε / 2] eps by auto
  define y' where y' = min (y + s / 2) ((y + g z) / 2)
  have y'1: y < y'
    using s(1) yz by (simp add: y'-def)
  have y'2: y' ≤ g z
    using yz by (auto simp: y'-def min-le-iff-disj)
  have y'3: |y' − y| < s
    using s(1) yz unfolding y'-def by (auto simp: min-def)
  have upper': ginv g y' < x0 + ε / 2
    using s(2)[of y'] y'1 y'3 unfolding x0-def
    by (auto simp: abs-real-def split: if-split-asm)
  obtain x2 where x2: x2 ∈ {ginv g y'<..<x0 + ε} isCont g x2
    using real-interval-avoid-countable-set[OF - cd, of ginv g y' x0 + ε] upper' eps
by auto
  have gx2: y < g x2
  proof −
    have ne': {x. y' ≤ g x} ≠ {}
      using y'2 by auto
    have Inf {x. y' ≤ g x} < x2
      using x2(1) by (auto simp: ginv-def)
    from cInf-lessD[OF ne' this] obtain x' where x': y' ≤ g x' x' < x2
      by auto
    hence y' ≤ g x2
      using monoD[OF mono-g, of x' x2] by auto
    with y'1 show ?thesis
      by auto
  qed
  have ev1: ∀F n in sequentially. Fn n x1 < y
    by (rule order-tendstoD(2)[OF wc[OF x1(2)] gx1])
  have ev2: ∀F n in sequentially. y < Fn n x2
    by (rule order-tendstoD(1)[OF wc[OF x2(2)] gx2])
  have ∀F n in sequentially. dist (ginv (Fn n) y) x0 < ε
    using ev1 ev2
  proof eventually-elim
```

```
    case (elim n)
    have ne-n: {x. y ≤ Fn n x} ≠ {}
      using elim(2) by (auto intro!: exI[of - x2] less-imp-le)
    have up: ginv (Fn n) y ≤ x2
      unfolding ginv-def
       by (intro cInf-lower bdd-below-ginv-set[OF mono-Fn elim(1)]) (use elim(2)
in auto)
    have low: x1 ≤ ginv (Fn n) y
      unfolding ginv-def
      by (intro cInf-greatest[OF ne-n])
           (metis mono-Fn elim(1) leI mem-Collect-eq monoD order-less-le-trans
less-le-not-le)
    show ?case
      using up low x1(1) x2(1) unfolding dist-real-def by auto
  qed
  thus∃ no. ∀ n≥no. dist (ginv (Fn n) y) (ginv g y) < ε
    unfolding eventually-sequentially x0-def by blast
qed

context right-continuous-mono
begin

lemma ex-gt-b:
  assumes p < b
  shows ∃ z. p < f z
  using order-tendstoD[OF top,of p] assms
  by (auto dest!: eventually-happens'[OF trivial-limit-at-top-linorder])

lemma ex-ge-b:
  assumes p < b
  shows ∃ z. p ≤ f z
  using ex-gt-b[OF assms] by(auto intro!: less-imp-le)

lemma ex-less-a:
  assumes a < p
  shows ∃ w. f w < p
  using order-tendstoD[OF bot,of p] assms
  by (auto dest!: eventually-happens'[OF trivial-limit-at-bot-linorder])

lemma ex-le-a:
  assumes a < p
  shows ∃ w. f w ≤ p
  using ex-less-a[OF assms] by(auto intro!: less-imp-le)

lemma a-le-b:a ≤ b
proof −
  have ∀ F w in at-bot. f w ≤ f 0
     by (auto intro!: eventually-mono[OF eventually-le-at-bot[of 0]] monoD[OF
mono])
```

```
  from tendsto-upperbound[OF bot this]
  have a ≤ f 0
    by simp
  moreover
  have ∀F w in at-top. f 0 ≤ f w
     by (auto intro!: eventually-mono[OF eventually-ge-at-top[of 0]] monoD[OF
mono])
  from tendsto-lowerbound[OF top this]
  have f 0 ≤ b
    by simp
  ultimately show ?thesis
    by linarith
qed

lemma ginv-le-iff:
  assumes a < p p < b
  shows ginv f p ≤ x ⟷ p ≤ f x
  unfolding ginv-def
  by(intro pseudoinverse[symmetric] assms)

lemma ginv-attained:
  assumes a < p p < b
  shows p ≤ f (ginv f p)
  using ginv-le-iff[OF assms, of ginv f p] by simp

lemma
  shows a-le-f: a ≤ f x
    and f-le-b: f x ≤ b
proof −
  have ∀F w in at-bot. f w ≤ f x
     by (auto intro!: eventually-mono[OF eventually-le-at-bot[of x]] monoD[OF
mono])
  from tendsto-upperbound[OF bot this]
  show a ≤ f x
    by simp
  have ∀F w in at-top. f x ≤ f w
     by (auto intro!: eventually-mono[OF eventually-ge-at-top[of x]] monoD[OF
mono])
  from tendsto-lowerbound[OF top this]
  show f x ≤ b
    by simp
qed

lemma ginv-left-cont:
  assumes ap: a < p and pb:p < b
  shows (ginv f ⟶ ginv f p) (at-left p)
proof (rule order-tendstoI)
  fix c
  assume c: c < ginv f p
```

```
  hence fc-less: f c < p
    using ginv-le-iff[OF assms, of c] by simp
  define p0 where p0 = (f c + p) / 2
  have p0-pos: a < p0 and p0-lt-p: p0 < p and Fc-lt-p0: f c < p0
    using a-le-f[of c] ap fc-less by(auto simp: p0-def)
  have p0-lt-1: p0 < b
    using p0-lt-p pb by simp
  have c-lt-p0: c < ginv f p0
    using ginv-le-iff[OF p0-pos p0-lt-1, of c] Fc-lt-p0 by force
  have ∀F x in at-left p. p0 < x ∧ x < p
    using eventually-at-left-field[where x = p] p0-lt-p by auto
  thus ∀F x in at-left p. c < ginv f x
    by eventually-elim
      (metis c-lt-p0 pb dual-order.strict-trans ginv-le-iff linorder-not-less p0-pos)
next
  fix c
  assume c: ginv f p < c
  have ∀F x in at-left p. a < x ∧ x < p
    using eventually-at-left-field[where x = p] ap by auto
  thus ∀F x in at-left p. ginv f x < c
    by eventually-elim (smt (verit) c pb ginv-le-iff)
qed

lemma f-eq-b:
  assumes f y = b
    and y ≤ x
  shows f x = b
  using assms f-le-b mono monoD order-antisym by blast

lemma f-eq-a:
  assumes f y = a
    and x ≤ y
  shows f x = a
  by (metis a-le-f assms dual-order.eq-iff mono monoD)

corollary regularly-varying-less-b:
  assumes regularly-varying (λx. b − f x) ϱ
  shows f x < b
proof(rule ccontr)
  assume ¬ f x < b
  hence eq:⋀y. x ≤ y ⟹ f y = b
    by (metis f-eq-b f-le-b not-le order-antisym)
  show False
    using regularly-varying-pos[OF assms,of max 1 x]
    by(cases 1 ≤ x) (auto simp: eq)
qed

lemma ex-gt-less:
  assumes z: f z < c
```

```
  shows ∃ w>z. f w < c
proof −
  have ∀ F y in at-right z. f y < c ∧ z < y
    using order-tendstoD(2)[OF cont[of z, unfolded continuous-within] z]
      eventually-at-right-less by eventually-elim simp
  from eventually-happens'[OF trivial-limit-at-right-real this]
  show ?thesis
    by blast
qed

lemma upper-semicontinuous: upper-semicontinuous-map euclidean f
proof −
  have open {x. f x < c} for c
  proof −
    have ∗:(⋃ w∈{w. f w < c}. {..<w}) = {x. f x < c}
    proof (intro set-eqI iffI)
      fix x
      assume x ∈ (⋃ w∈{w. f w < c}. {..<w})
      then obtain w where f w < c x < w
        by auto
      thus x ∈ {x. f x < c}
        using monoD[OF mono, of x w] by simp
    next
      fix x
      assume x ∈ {x. f x < c}
      thus x ∈ (⋃ w∈{w. f w < c}. {..<w})
        using ex-gt-less[of x c] by auto
    qed
    have open (⋃ w∈{w. f w < c}. {..<w})
      by fastforce
    thus ?thesis
      unfolding ∗ .
  qed
  thus ?thesis
    by(auto simp: upper-semicontinuous-map-def)
qed

corollary open-less: open {x. f x < c}
  using upper-semicontinuous by(simp add: upper-semicontinuous-map-def)

lemma strict-mono-of-deriv:
  assumes deriv: ⋀z. x0 < z ⟹ f z < c ⟹ (f has-real-derivative f' z) (at z)
    and pos: ⋀z. x0 < z ⟹ f z < c ⟹ 0 < f' z
    and x: x0 < x and xy: x < y and fy: f y < c
  shows f x < f y
proof (rule DERIV-pos-imp-increasing[OF xy])
  fix z
  assume z: x ≤ z z ≤ y
  hence x0 < z f z < c
```

```
    using x fy monoD[OF mono, of z y] by auto
  thus ∃ w. (f has-real-derivative w) (at z) ∧ 0 < w
    using deriv pos by blast
qed

lemma ginv-f-eq:
  assumes fmono: ⋀x y. x0 < x ⟹ x < y ⟹ f y < b ⟹ f x < f y
    and z: x0 < z and fz: f z < b
  shows ginv f (f z) = z
proof −
  have w: x0 < (x0 + z) / 2 (x0 + z) / 2 < z
    using z by simp-all
  have f ((x0 + z) / 2) < f z
    by (rule fmono[OF w fz])
  hence p: a < f z
    using a-le-f[of (x0 + z) / 2] by simp
  have z ≤ ginv f (f z)
  proof (rule ccontr)
    assume ¬ z ≤ ginv f (f z)
    hence lt: ginv f (f z) < z
      by simp
    define v where v = (max x0 (ginv f (f z)) + z) / 2
    have vx0: x0 < v and vginv: ginv f (f z) < v and vz: v < z
      using lt z by (auto simp: v-def)
    have f z ≤ f v
      using ginv-le-iff[OF p fz, of v] vginv by simp
    moreover have f v < f z
      by (rule fmono[OF vx0 vz fz])
    ultimately show False
      by simp
  qed
  moreover have ginv f (f z) ≤ z
    using ginv-le-iff[OF p fz, of z] by simp
  ultimately show ?thesis
    by simp
qed

end

sublocale real-distribution ⊆ right-continuous-mono cdf M 0 1
  by standard
    (auto intro!: monoI cdf-nondecreasing cdf-is-right-cont cdf-lim-at-bot cdf-lim-at-top-prob)

corollary right-continuous-mono-cdf:
  assumes real-distribution M
  shows right-continuous-mono (cdf M) 0 1
proof −
  interpret real-distribution M by fact
  show ?thesis
```

by *unfold-locales*
qed

lemma *one-less-of-inverse-tail-less*:
fixes $F :: real \Rightarrow real$
assumes F: *right-continuous-mono F 0 1* **and** $x0$: $F\ x0 < 1$
and t: $1 / (1 - F\ x0) < t$
shows $1 < t$
proof −
have $1 \leq 1 / (1 - F\ x0)$
using *x0 right-continuous-mono.a-le-f*[*OF F*, *of x0*] **by** *simp*
thus *?thesis*
using t **by** *linarith*
qed

1.5 Lemmas for Infinite Sums

The following are used to prove relation (1.2.21) of [2].

lemma *geometric-ratio-chain-ge*:
fixes $d :: nat \Rightarrow real$
assumes *pos*: $\bigwedge n.\ 0 < d\ n$
and r: $0 < r$
and *bound*: $\bigwedge j.\ M \leq j \Longrightarrow r \leq d\ (Suc\ j) / d\ j$
and k: $M \leq k$
shows $d\ k * r \hat{} m \leq d\ (k + m)$
proof (*induction m*)
case (*Suc m*)
hence $d\ k * r \hat{} Suc\ m \leq r * d\ (k + m)$
using r **by** (*auto intro*!: *mult-left-mono*)
also have $\ldots \leq d\ (Suc\ (k + m))$
using *bound*[*of k + m*] *k pos*[*of k + m*] **by** (*simp add*: *field-simps*)
finally show *?case*
by *simp*
qed *simp*

lemma *geometric-ratio-chain-le*:
fixes $d :: nat \Rightarrow real$
assumes *pos*: $\bigwedge n.\ 0 < d\ n$
and r: $0 < r$
and *bound*: $\bigwedge j.\ M \leq j \Longrightarrow d\ (Suc\ j) / d\ j \leq r$
and k: $M \leq k$
shows $d\ (k + m) \leq d\ k * r \hat{} m$
proof (*induction m*)
case (*Suc m*)
have $d\ (Suc\ (k + m)) \leq r * d\ (k + m)$
using *bound*[*of k + m*] *k pos*[*of k + m*] **by** (*simp add*: *field-simps*)
also have $\ldots \leq d\ k * r \hat{} Suc\ m$
using *Suc r* **by** (*auto intro*!: *mult-left-mono*)
finally show *?case*

by *simp*
qed *simp*

lemma *geometric-ratio-tailsum-tendsto*:
fixes $d :: nat \Rightarrow real$ **and** $L :: real$
assumes *pos*: $\bigwedge n.\ 0 < d\ n$
and *L*: $0 < L\ L < 1$
and *ratio*: $(\lambda n.\ d\ (Suc\ n)\ /\ d\ n) \longrightarrow L$
shows *geometric-ratio-tailsum-summable*: *summable d*
and *geometric-ratio-tailsum-div*: $(\lambda n.\ (\sum k.\ d\ (k + n))\ /\ d\ n) \longrightarrow 1\ /\ (1 - L)$
and *geometric-ratio-tailsum-ratio*: $(\lambda n.\ (\sum k.\ d\ (k + Suc\ n))\ /\ (\sum k.\ d\ (k + n))) \longrightarrow L$
proof −
define *cs* **where** $cs = (1 + L)\ /\ 2$
have *cs-bounds*: $L < cs\ cs < 1\ 0 < cs$
using *L* **by** (*simp-all add*: *cs-def*)
obtain *N0* **where** *N0*: $\bigwedge n.\ N0 \leq n \Longrightarrow d\ (Suc\ n)\ /\ d\ n < cs$
using *order-tendstoD*(*2*)[*OF ratio cs-bounds*(*1*)] **unfolding** *eventually-sequentially*
by *blast*
have *step-le0*: $d\ (Suc\ n) \leq cs * d\ n$ **if** $N0 \leq n$ **for** *n*
using *N0*[*OF that*] *pos*[*of n*] **by** (*simp add*: *field-simps*)
show *summable*: *summable d*
using *cs-bounds step-le0 pos*
by(*auto intro*!: *summable-ratio-test*[*of cs N0 d*] *simp*: *abs-of-pos*)
note *chain-ge* = *geometric-ratio-chain-ge*[**where** *d*=*d*, *OF pos*]
note *chain-le* = *geometric-ratio-chain-le*[**where** *d*=*d*, *OF pos*]
define *T* **where** $T = (\lambda n.\ \sum k.\ d\ (k + n))$
have *shift-summable*: *summable* $(\lambda k.\ d\ (k + n))$ **for** *n*
using *summable* **by** *simp*
have *T-eq*: $T\ n = suminf\ d - (\sum i{<}n.\ d\ i)$ **for** *n*
unfolding *T-def* **using** *suminf-minus-initial-segment*[*OF summable*, *of n*] **by** *simp*
have *partial-tendsto*: $(\lambda n.\ \sum i{<}n.\ d\ i) \longrightarrow suminf\ d$
using *summable* **by** (*rule summable-LIMSEQ*)
have *T-fun-eq*: $T = (\lambda n.\ suminf\ d - (\sum i{<}n.\ d\ i))$
using *T-eq* **by** (*intro ext*) *simp*
have *T-tendsto0*: $T \longrightarrow 0$
unfolding *T-fun-eq*
using *tendsto-diff*[*OF tendsto-const partial-tendsto*, *of suminf d*] **by** *simp*
have *T-pos*: $0 < T\ n$ **for** *n*
proof −
have *I*: *finite* $\{0{::}nat\}$ $\bigwedge k.\ k \in -\ \{0{::}nat\} \Longrightarrow 0 \leq d\ (k + n)$
using *pos* **by** (*auto simp*: *less-eq-real-def*)
have $(0{::}real) < d\ (0 + n)$
using *pos* **by** *simp*
also have $d\ (0 + n) = sum\ (\lambda k.\ d\ (k + n))\ \{0\}$
by *simp*
also have $\ldots \leq suminf\ (\lambda k.\ d\ (k + n))$

```
    by (rule sum-le-suminf[OF shift-summable[of n] I(1) I(2)])
  finally show ?thesis
    by (simp add: T-def)
qed
have T-diff: T n − T (Suc n) = d n for n
  using T-eq[of n] T-eq[of Suc n] by simp
have geom-sum: (∑ k. d n * r ^ k) = d n / (1 − r) if r: 0 < r r < 1 for n r
  using suminf-geometric[of r] r by (simp add: suminf-mult)
have geom-summable: summable (λk. d n * r ^ k) if r: 0 < r r < 1 for n r
  using summable-geometric[of r] r by (intro summable-mult) simp
have div-tendsto: (λn. T n / d n) ⟶ 1 / (1 − L)
proof (rule order-tendstoI)
  fix c :: real
  assume c: 1 / (1 − L) < c
  have c-pos: 0 < c
    using L c by (smt (verit) divide-pos-pos zero-less-divide-1-iff)
  define r where r = (L + (1 − 1 / c)) / 2
  have Lc: L < 1 − 1 / c
    using c c-pos L by (simp add: divide-less-eq field-simps)
  have r-bounds: L < r r < 1 − 1 / c
    using Lc by (simp-all add: r-def)
  have r-pos: 0 < r
    using L r-bounds(1) by simp
  have r-lt1: r < 1
    using r-bounds(2) c-pos by (smt (verit) zero-less-divide-1-iff)
  have r-val: 1 / (1 − r) < c
    using r-bounds(2) c-pos r-lt1 by (simp add: field-simps)
  obtain N where N: ⋀j. N ≤ j ⟹ d (Suc j) / d j < r
    using order-tendstoD(2)[OF ratio r-bounds(1)]
    unfolding eventually-sequentially by blast
  have bound-le: ⋀j. N ≤ j ⟹ d (Suc j) / d j ≤ r
    using N by (intro less-imp-le) simp
  show ∀F n in sequentially. T n / d n < c
    using eventually-ge-at-top[of N]
  proof eventually-elim
    case (elim n)
    have term-le: d (n + m) ≤ d n * r ^ m for m
      by (rule chain-le[OF r-pos bound-le]) (use elim in auto)
    have term-le': d (k + n) ≤ d n * r ^ k for k
      using term-le[of k] by (simp add: add.commute)
    have T n ≤ (∑ k. d n * r ^ k)
      unfolding T-def
     by (rule suminf-le[OF term-le' shift-summable[of n] geom-summable[OF r-pos
r-lt1]])
    also have ... = d n / (1 − r)
      by (rule geom-sum[OF r-pos r-lt1])
    finally have T n ≤ d n / (1 − r) .
    hence T n / d n ≤ 1 / (1 − r)
      using pos[of n] by (simp add: field-simps)
```

```
    thus ?case
      using r-val by simp
  qed
next
  fix c :: real
  assume c: c < 1 / (1 - L)
  show ∀F n in sequentially. c < T n / d n
  proof (cases c ≤ 0)
    case True
    have ∀x. c < T x / d x
      using T-pos pos True by (auto intro: divide-pos-pos order-le-less-trans)
    thus ?thesis
      by (rule always-eventually)
  next
    case False
    hence c-pos: 0 < c
      by simp
    have Lc: 1 - 1 / c < L
      using c c-pos L by (simp add: divide-less-eq field-simps)
    define ε where ε = min ((L - (1 - 1 / c)) / 2) (L / 2)
    have ε-le1: ε ≤ (L - (1 - 1 / c)) / 2
      unfolding ε-def by (rule min.cobounded1)
    have ε-le2: ε ≤ L / 2
      unfolding ε-def by (rule min.cobounded2)
    have ε-pos: 0 < ε
      using Lc L by (simp add: ε-def)
    define r where r = L - ε
    have r-pos: 0 < r
      unfolding r-def using ε-le2 L by linarith
    have r-lt-L: r < L
      unfolding r-def using ε-pos by simp
    have r-lt1: r < 1
      using r-lt-L L by simp
    have r-gt: 1 - 1 / c < r
      using ε-le1 Lc unfolding r-def by argo
    have r-val: c < 1 / (1 - r)
      using r-gt c-pos r-lt-L L by (simp add: field-simps)
    have ev-le: ∀F n in sequentially. r ≤ d (Suc n) / d n
      using order-tendstoD(1)[OF ratio r-lt-L] by eventually-elim simp
    obtain N where N: ⋀j. N ≤ j ⟹ r ≤ d (Suc j) / d j
      using ev-le unfolding eventually-sequentially by blast
    show ∀F n in sequentially. c < T n / d n
      using eventually-ge-at-top[of N]
    proof eventually-elim
      case (elim n)
      have term-ge: d n * r ^ m ≤ d (n + m) for m
        by (rule chain-ge[OF r-pos N]) (use elim in auto)
      have term-ge': d n * r ^ k ≤ d (k + n) for k
        using term-ge[of k] by (simp add: add.commute)
```

```
      have d n / (1 − r) = (∑ k. d n ∗ r ^ k)
        by (rule geom-sum[OF r-pos r-lt1, symmetric])
      also have ... ≤ T n
        unfolding T-def
      by (rule suminf-le[OF term-ge' geom-summable[OF r-pos r-lt1] shift-summable[of
n]])
      finally have d n / (1 − r) ≤ T n .
      hence 1 / (1 − r) ≤ T n / d n
        using pos[of n] by (simp add: field-simps)
      thus ?case
        using r-val by simp
    qed
  qed
  show div-goal: (λn. (∑ k. d (k + n)) / d n) ⟶ 1 / (1 − L)
    using div-tendsto by (simp add: T-def)
  have d-over-T: (λn. d n / T n) ⟶ (1 − L)
  proof −
    have inv: (λn. inverse (T n / d n)) ⟶ inverse (1 / (1 − L))
      by (rule tendsto-inverse[OF div-tendsto]) (use L in simp)
    have eq1: inverse (1 / (1 − L)) = 1 − L
      by simp
    have eq2: ⋀n. inverse (T n / d n) = d n / T n
      using pos T-pos by (simp add: field-simps)
    show ?thesis
      using inv by (simp add: eq1 eq2)
  qed
  have T-ratio: (λn. T (Suc n) / T n) ⟶ L
  proof −
    have eqn: T (Suc n) / T n = 1 − d n / T n for n
      using T-diff[of n] T-pos[of n] by (simp add: field-simps)
    have (λn. 1 − d n / T n) ⟶ 1 − (1 − L)
      by (intro tendsto-diff[OF tendsto-const d-over-T])
    thus ?thesis
      using eqn by simp
  qed
  show (λn. (∑ k. d (k + Suc n)) / (∑ k. d (k + n))) ⟶ L
    using T-ratio by (simp add: T-def)
qed

lemma geometric-ratio-sum-tendsto:
  fixes d :: nat ⇒ real and L :: real
  assumes pos: ⋀n. 0 < d n
    and L: 1 < L
    and ratio: (λn. d (Suc n) / d n) ⟶ L
  shows geometric-ratio-sum-div-last: (λn. (∑ k≤n. d k) / d n) ⟶ L / (L −
1)
    and geometric-ratio-sum-ratio: (λn. (∑ k≤Suc n. d k) / (∑ k≤n. d k)) ⟶
L
```

```
proof −
  have d-at-top: filterlim d at-top sequentially
  proof −
    define r0 :: real where r0 = (1 + L) / 2
    have r0-bounds: 1 < r0 r0 < L
      using L by (simp-all add: r0-def)
    obtain N where N: ⋀n. N ≤ n ⟹ r0 < d (Suc n) / d n
      using order-tendstoD(1)[OF ratio r0-bounds(2)]
      unfolding eventually-sequentially by blast
    have grow: r0 ^ k * d N ≤ d (N + k) for k
    proof (induction k)
      case 0
      show ?case
        by simp
    next
      case (Suc k)
      have step: r0 * d (N + k) < d (N + Suc k)
        using N[of N + k] pos[of N + k] by (simp add: field-simps)
      have r0 ^ Suc k * d N = r0 * (r0 ^ k * d N)
        by simp
      also have ... ≤ r0 * d (N + k)
        using Suc r0-bounds by (intro mult-left-mono) auto
      also have ... < d (N + Suc k)
        by (rule step)
      finally show ?case
        by simp
    qed
    have r0k-at-top: filterlim (λk. r0 ^ k) at-top sequentially
    proof −
      have inv0: (λk. inverse (r0 ^ k)) ⟶ 0
        using r0-bounds(1) by (rule LIMSEQ-inverse-realpow-zero)
      have posr: ∀F k in sequentially. 0 < inverse (r0 ^ k)
        using r0-bounds(1) by (auto intro!: always-eventually)
      from filterlim-inverse-at-top[OF inv0 posr] show ?thesis
        by simp
    qed
    have shifted: filterlim (λk. d (N + k)) at-top sequentially
    proof (rule filterlim-at-top-mono)
      have ((λk::nat. d N) ⟶ d N)
        by (rule tendsto-const)
      from filterlim-tendsto-pos-mult-at-top[OF this pos[of N] r0k-at-top]
      show filterlim (λk. d N * r0 ^ k) at-top sequentially .
      show ∀F k in sequentially. d N * r0 ^ k ≤ d (N + k)
        using grow by (intro always-eventually) (simp add: mult.commute)
    qed
    show ?thesis
      unfolding filterlim-at-top
    proof
      fix Z :: real
```

```
    obtain M where M: ⋀k. M ≤ k ⟹ Z ≤ d (N + k)
      using shifted unfolding filterlim-at-top eventually-sequentially by blast
    show ∀F n in sequentially. Z ≤ d n
      unfolding eventually-sequentially
    proof (intro exI[of - N + M] allI impI)
      fix n
      assume N + M ≤ n
      hence M ≤ n - N N ≤ n
        by simp-all
      with M[of n - N] show Z ≤ d n
        by simp
    qed
  qed
qed
define S where S = (λn. sum d {..n})
have S-pos: 0 < S n for n
  unfolding S-def by (rule sum-pos) (auto simp: pos)
have S-split: S n = sum d {..<N} + sum d {N..n} if N ≤ n for N n
  using that
  by(subst sum.union-disjoint[symmetric])
    (auto simp: S-def intro!: arg-cong[where f=sum d])
note chain-ge = geometric-ratio-chain-ge[where d=d, OF pos]
note chain-le = geometric-ratio-chain-le[where d=d, OF pos]
have div-last: (λn. S n / d n) ⟶ L / (L - 1)
proof (rule order-tendstoI)
  fix c :: real
  assume c: L / (L - 1) < c
  have c-gt1: 1 < c
    using L c by (smt (verit) divide-le-eq-1 le-divide-eq-1-pos)
  have Lc: c / (c - 1) < L
    using c c-gt1 L by (simp add: divide-less-eq field-simps)
  define ε where ε = (L - c / (c - 1)) / 2
  have ε-pos: 0 < ε
    using Lc by (simp add: ε-def)
  define r where r = L - ε
  have cc1-gt1: 1 < c / (c - 1)
    using c-gt1 by simp
  have r-bounds: 1 < r r < L
  proof -
    have h: (2::real) < L + c / (c - 1)
      using L cc1-gt1 by linarith
    show 1 < r
      unfolding r-def ε-def using h by argo
    show r < L
      using ε-pos by (simp add: r-def)
  qed
  have r-gt: c / (c - 1) < r
    using Lc ε-def r-def by auto
  have r-c: r / (r - 1) < c
```

```
  using r-bounds(1) r-gt c-gt1 by (simp add: field-simps)
 obtain N where N: ⋀j. N ≤ j ⟹ r < d (Suc j) / d j
  using order-tendstoD(1)[OF ratio r-bounds(2)]
  unfolding eventually-sequentially by blast
 have N-ge: ⋀j. N ≤ j ⟹ r ≤ d (Suc j) / d j
  using N by (intro less-imp-le) simp
 have geo-bound: (∑j≤m. (1 / r) ^ j) < r / (r − 1) for m
 proof −
  have ne1: (1::real) / r ≠ 1
   using r-bounds(1) by simp
  have (∑j≤m. (1 / r) ^ j) = (∑j<Suc m. (1 / r) ^ j)
   by (simp add: lessThan-Suc-atMost)
  also have ... = (1 − (1 / r) ^ Suc m) / (1 − 1 / r)
   using sum-gp-strict[of 1 / r Suc m] ne1 by simp
  also have ... < 1 / (1 − 1 / r)
  proof −
   have 0 < (1 / r) ^ Suc m
    using r-bounds(1) by simp
   hence 1 − (1 / r) ^ Suc m < 1
    by simp
   thus ?thesis
    using r-bounds(1) by (intro divide-strict-right-mono) auto
  qed
  also have ... = r / (r − 1)
   using r-bounds(1) by (simp add: field-simps)
  finally show ?thesis .
 qed
 have sum-bound-upper: sum d {N..n} ≤ d n * (∑j≤n − N. (1 / r) ^ j)
  if n: N ≤ n for n
 proof −
  have term-bound: d k ≤ d n * (1 / r) ^ (n − k) if k: k ∈ {N..n} for k
  proof −
   have d k * r ^ (n − k) ≤ d (k + (n − k))
    by (rule chain-ge[OF - N-ge]) (use k r-bounds in auto)
   also have k + (n − k) = n
    using k by simp
   finally have d k * r ^ (n − k) ≤ d n .
   thus ?thesis
    using r-bounds(1) by (simp add: field-simps)
  qed
  have sum d {N..n} ≤ sum (λk. d n * (1 / r) ^ (n − k)) {N..n}
   by (rule sum-mono) (use term-bound in auto)
  also have ... = d n * sum (λk. (1 / r) ^ (n − k)) {N..n}
   by (simp add: sum-distrib-left)
  also have sum (λk. (1 / r) ^ (n − k)) {N..n} = sum (λj. (1 / r) ^ j) {..n
− N}
   by (rule sum.reindex-bij-witness[where i = λk. n − k and j = λj. n − j])
    (use n in auto)
  finally show ?thesis
```

```
    by simp
  qed
  have S-bound: S n ≤ sum d {..<N} + d n * (r / (r - 1)) if n: N ≤ n for n
  proof -
    have S n = sum d {..<N} + sum d {N..n}
      by (rule S-split[OF n])
    also have ... ≤ sum d {..<N} + d n * (∑j≤n - N. (1 / r) ^ j)
      using sum-bound-upper[OF n] by simp
    also have ... < sum d {..<N} + d n * (r / (r - 1))
      using geo-bound[of n - N] pos[of n]
      by (intro add-strict-left-mono mult-strict-left-mono) auto
    finally show ?thesis
      by simp
  qed
  have const-over-d: (λn. sum d {..<N} / d n) ⟶ 0
  proof -
    have (λn. inverse (d n)) ⟶ 0
      using d-at-top by (rule tendsto-inverse-0-at-top)
    hence (λn. sum d {..<N} * inverse (d n)) ⟶ sum d {..<N} * 0
      by (intro tendsto-mult tendsto-const)
    thus ?thesis
      by (simp add: inverse-eq-divide)
  qed
  have gap-pos: 0 < c - r / (r - 1)
    using r-c by simp
  have ev-small: ∀F n in sequentially. sum d {..<N} / d n < c - r / (r - 1)
    using order-tendstoD(2)[OF const-over-d gap-pos] .
  show ∀F n in sequentially. S n / d n < c
    using eventually-ge-at-top[of N] ev-small
  proof eventually-elim
    case (elim n)
    have S n / d n ≤ sum d {..<N} / d n + r / (r - 1)
      using S-bound[OF elim(1)] pos[of n] by (simp add: field-simps)
    also have ... < c
      using elim(2) by simp
    finally show ?case .
  qed
next
  fix c :: real
  assume c: c < L / (L - 1)
  show ∀F n in sequentially. c < S n / d n
  proof (cases c ≤ 1)
    case True
    show ?thesis
      using eventually-ge-at-top[of 1]
    proof eventually-elim
      case (elim n)
      have S n = sum d {..<n} + d n
        unfolding S-def by (simp add: lessThan-Suc-atMost[symmetric])
```

```
    moreover have 0 < sum d {..<n}
      using pos elim by(auto intro!: sum-pos simp: lessThan-empty-iff)
    ultimately have d n < S n
      by simp
    hence 1 < S n / d n
      using pos[of n] by (simp add: field-simps)
    with True show ?case
      by simp
  qed
next
  case False
  hence c-gt1: 1 < c
    by simp
  have cc1: L < c / (c - 1)
  proof -
    have c * (L - 1) < L / (L - 1) * (L - 1)
      using c L by (intro mult-strict-right-mono) auto
    hence step1: c * (L - 1) < L
      using L by simp
    have step2: L * (c - 1) < c
      using step1 by argo
    show ?thesis
      using step2 c-gt1 by (simp add: pos-less-divide-eq)
  qed
  define ε where ε = (c / (c - 1) - L) / 2
  have ε-pos: 0 < ε
    using cc1 by (simp add: ε-def)
  define r where r = L + ε
  have r-bounds: 1 < r
    using L ε-pos by (simp add: r-def)
  have r-lt: r < c / (c - 1)
    unfolding r-def ε-def using cc1 by argo
  have r-c: c < r / (r - 1)
  proof -
    have r * (c - 1) < c / (c - 1) * (c - 1)
      using r-lt c-gt1 by (intro mult-strict-right-mono) auto
    hence step1: r * (c - 1) < c
      using c-gt1 by simp
    have step2: c * (r - 1) < r
      using step1 by argo
    show ?thesis
      using step2 r-bounds by (simp add: pos-less-divide-eq)
  qed
  have L-lt-r: L < r
    using ε-pos by (simp add: r-def)
  obtain N where N: ⋀j. N ≤ j ⟹ d (Suc j) / d j < r
    using order-tendstoD(2)[OF ratio L-lt-r]
    unfolding eventually-sequentially by blast
  have N-le: ⋀j. N ≤ j ⟹ d (Suc j) / d j ≤ r
```

```
    using N by (intro less-imp-le) simp
  have sum-bound-lower: d n * (∑j≤n - N. (1 / r) ^ j) ≤ sum d {N..n}
    if n: N ≤ n for n
  proof -
    have term-bound: d n * (1 / r) ^ (n - k) ≤ d k if k: k ∈ {N..n} for k
    proof -
      have d (k + (n - k)) ≤ d k * r ^ (n - k)
        by (rule chain-le[OF - N-le]) (use k r-bounds in auto)
      also have k + (n - k) = n
        using k by simp
      finally have d n ≤ d k * r ^ (n - k) .
      thus ?thesis
        using r-bounds by (simp add: field-simps)
    qed
    have d n * sum (λk. (1 / r) ^ (n - k)) {N..n} ≤ sum d {N..n}
      unfolding sum-distrib-left by (rule sum-mono) (use term-bound in auto)
    moreover have sum (λk. (1 / r) ^ (n - k)) {N..n} = sum (λj. (1 / r) ^
j) {..n - N}
      by (rule sum.reindex-bij-witness[where i = λk. n - k and j = λj. n - j])
        (use n in auto)
    ultimately show ?thesis
      by simp
  qed
  have S-bound: d n * (∑j≤n - N. (1 / r) ^ j) ≤ S n if n: N ≤ n for n
  proof -
    have d n * (∑j≤n - N. (1 / r) ^ j) ≤ sum d {N..n}
      by (rule sum-bound-lower[OF n])
    also have ... ≤ sum d {..<N} + sum d {N..n}
      using sum-nonneg[of {..<N} d] pos by (simp add: less-imp-le)
    also have ... = S n
      using S-split[OF n] by simp
    finally show ?thesis .
  qed
  have geo-tendsto: (λm. ∑j≤m. (1 / r) ^ j) ⟶ r / (r - 1)
  proof -
    have norm (1 / r) < 1
      using r-bounds by simp
    from geometric-sums[OF this]
    have (λn. ∑i<n. (1 / r) ^ i) ⟶ 1 / (1 - 1 / r)
      unfolding sums-def by simp
    from LIMSEQ-Suc[OF this]
    show ?thesis
      using r-bounds by (simp add: lessThan-Suc-atMost field-simps)
  qed
  have ev-close: ∀F m in sequentially. r / (r - 1) - (r / (r - 1) - c)
                                        < (∑j≤m. (1 / r) ^ j)
    using order-tendstoD(1)[OF geo-tendsto, of r / (r - 1) - (r / (r - 1) -
c)]
      r-c by simp
```

```
    obtain M where M: ⋀m. M ≤ m ⟹
        r / (r − 1) − (r / (r − 1) − c) < (∑j≤m. (1 / r) ^ j)
      using ev-close unfolding eventually-sequentially by blast
    show ?thesis
      unfolding eventually-sequentially
    proof (intro exI[of - N + M] allI impI)
      fix n
      assume N + M ≤ n
      hence nN: N ≤ n and Mn: M ≤ n − N
        by simp-all
      have d n * (r / (r − 1) − (r / (r − 1) − c)) < d n * (∑j≤n − N. (1 /
r) ^ j)
        using M[OF Mn] pos[of n] by (intro mult-strict-left-mono) auto
      also have ... ≤ S n
        by (rule S-bound[OF nN])
      finally have d n * c < S n
        by simp
      thus c < S n / d n
        using pos[of n] by (simp add: field-simps)
    qed
  qed
 qed
 show geometric-ratio-sum-div-last: (λn. (∑k≤n. d k) / d n) ⟶ L / (L −
1)
  using div-last by (simp add: S-def)
 show geometric-ratio-sum-ratio: (λn. (∑k≤Suc n. d k) / (∑k≤n. d k)) ⟶
L
 proof −
  have L1ne0: L / (L − 1) ≠ 0
    using L by simp
  have d-div-S: (λn. d n / S n) ⟶ (L − 1) / L
  proof −
    have (λn. inverse (S n / d n)) ⟶ inverse (L / (L − 1))
      by (rule tendsto-inverse[OF div-last L1ne0])
    moreover have ⋀n. inverse (S n / d n) = d n / S n
      using pos S-pos by (simp add: inverse-eq-divide)
    moreover have inverse (L / (L − 1)) = (L − 1) / L
      using L by (simp add: field-simps)
    ultimately show ?thesis
      by simp
  qed
  note prod-lim = tendsto-mult[OF ratio d-div-S]
  have step: (λn. 1 + (d (Suc n) / d n) * (d n / S n)) ⟶ 1 + L * ((L −
1) / L)
    by (intro tendsto-add tendsto-const prod-lim)
  have eq: 1 + (d (Suc n) / d n) * (d n / S n) = S (Suc n) / S n for n
  proof −
    have (d (Suc n) / d n) * (d n / S n) = d (Suc n) / S n
      using pos[of n] by (simp add: field-simps)
```

```
    hence 1 + (d (Suc n) / d n) * (d n / S n) = (S n + d (Suc n)) / S n
      using S-pos[of n] by (simp add: field-simps)
    also have S n + d (Suc n) = S (Suc n)
      unfolding S-def by simp
    finally show ?thesis .
  qed
  have val-eq: 1 + L * ((L - 1) / L) = L
    using L by (simp add: field-simps)
  have (λn. S (Suc n) / S n) ⟶ L
    using step unfolding eq val-eq .
  thus ?thesis
    by (simp add: S-def)
 qed
qed

lemma geometric-ratio-forward-telescope:
 fixes g :: real ⇒ real and Z c :: real
 assumes power-ratio: ⋀t0. 0 < t0 ⟹ (λn. g (t0 * Z ^ Suc n) / g (t0 * Z ^
n)) ⟶ c
   and Z: 1 < Z and c: c ≠ 0
 shows (λn. g (Z ^ (n + j)) / g (Z ^ n)) ⟶ c ^ j
proof (induction j)
 case 0
 have ratio0: (λn. g (Z ^ Suc n) / g (Z ^ n)) ⟶ c
   using power-ratio[OF zero-less-one] by simp
 have ev-ne0: ∀F n in sequentially. g (Z ^ n) ≠ 0
   using tendsto-imp-eventually-ne[OF ratio0 c] by eventually-elim force
 have eq0: ∀F n in sequentially. (1::real) = g (Z ^ (n + 0)) / g (Z ^ n)
   using ev-ne0 by eventually-elim simp
 show ?case
   using Lim-transform-eventually[OF tendsto-const eq0] by simp
next
 case (Suc j)
 have Zjpos: 0 < Z ^ j
   using Z by simp
 have step: (λn. g (Z ^ (Suc n + j)) / g (Z ^ (n + j))) ⟶ c
   using power-ratio[OF Zjpos] by (simp add: algebra-simps power-add)
 have ev-ne: ∀F n in sequentially. g (Z ^ (n + j)) ≠ 0
   using tendsto-imp-eventually-ne[OF step c] by eventually-elim force
 have eq: ∀F n in sequentially.
     g (Z ^ (Suc n + j)) / g (Z ^ (n + j)) * (g (Z ^ (n + j)) / g (Z ^ n))
       = g (Z ^ (Suc n + j)) / g (Z ^ n)
   by(auto intro!: eventually-mono[OF ev-ne])
 show ?case
   using Lim-transform-eventually[OF tendsto-mult[OF step Suc.IH] eq] by simp
qed

lemma monotone-geometric-ratio-imp-ratio:
 fixes g :: real ⇒ real and ϱ y :: real
```

assumes *gmono*: $\bigwedge s\ t.\ 1 < s \Longrightarrow s \leq t \Longrightarrow g\ s \leq g\ t$
and *gpos*: $\forall_F\ t$ *in at-top*. $0 < g\ t$
and *rho*: $0 < \varrho$
and *power-ratio*: $\bigwedge Z\ t0.\ 1 < Z \Longrightarrow 0 < t0 \Longrightarrow$
$(\lambda n.\ g\ (t0 * Z \mathbin{\hat{}} Suc\ n) / g\ (t0 * Z \mathbin{\hat{}} n)) \longrightarrow Z\ powr\ \varrho$
and *y*: $0 < y$
shows $((\lambda t.\ g\ (t * y) / g\ t) \longrightarrow y\ powr\ \varrho)\ at\text{-}top$
proof −
have *Zseq-at-top*: *filterlim* $(\lambda n.\ Z \mathbin{\hat{}} n)$ *at-top sequentially*
if *Z*: $1 < Z$ **for** *Z* :: *real*
by (*rule LIMSEQ-power-at-top*[*OF Z*])
have *n-at-top*: *filterlim* $(\lambda t{::}real.\ nat\ \lfloor log\ Z\ t \rfloor)$ *at-top at-top*
if *Z*: $1 < Z$ **for** *Z* :: *real*
using *Z* **by** (*rule floor-log-at-top*)
have *Zn-cont-at-top*: *filterlim* $(\lambda t{::}real.\ Z \mathbin{\hat{}} nat\ \lfloor log\ Z\ t \rfloor)$ *at-top at-top*
if *Z*: $1 < Z$ **for** *Z* :: *real*
by (*rule filterlim-compose*[*OF Zseq-at-top*[*OF Z*] *n-at-top*[*OF Z*]])
have *Un-pos-ev*: $\forall_F\ t$ *in at-top*. $0 < g\ (Z \mathbin{\hat{}} nat\ \lfloor log\ Z\ t \rfloor)$ **if** *Z*: $1 < Z$
for *Z* :: *real*
using *eventually-compose-filterlim*[*OF gpos Zn-cont-at-top*[*OF Z*]] **.**
have *Zpowr-ne*: $Z\ powr\ \varrho \neq 0$ **if** *Z*: $1 < Z$ **for** *Z* :: *real*
using *Z* **by** *simp*
have *telescope*: $(\lambda n.\ g\ (Z \mathbin{\hat{}} (n + j)) / g\ (Z \mathbin{\hat{}} n))$
$\longrightarrow (Z\ powr\ \varrho) \mathbin{\hat{}} j$
if *Z*: $1 < Z$ **for** *Z* :: *real* **and** *j* :: *nat*
by (*rule geometric-ratio-forward-telescope*[*OF power-ratio*[*OF Z*] *Z Zpowr-ne*[*OF Z*]])
have *telescope-cont*: $((\lambda t.\ g\ (Z \mathbin{\hat{}} (nat\ \lfloor log\ Z\ t \rfloor + j))$
$/\ g\ (Z \mathbin{\hat{}} nat\ \lfloor log\ Z\ t \rfloor))$
$\longrightarrow (Z\ powr\ \varrho) \mathbin{\hat{}} j)\ at\text{-}top$
if *Z*: $1 < Z$ **for** *Z* :: *real* **and** *j* :: *nat*
by (*rule filterlim-compose*[*OF telescope*[*OF Z*] *n-at-top*[*OF Z*]])
have *t-bracket*: $\forall_F\ t$ *in at-top*. $Z \mathbin{\hat{}} nat\ \lfloor log\ Z\ t \rfloor \leq t \wedge t < Z \mathbin{\hat{}} (nat\ \lfloor log\ Z\ t \rfloor + 1)$
if *Z*: $1 < Z$ **for** *Z* :: *real*
using *eventually-ge-at-top*[*of 1*::*real*]
by *eventually-elim* (*use power-of-nat-log-le power-of-nat-log-less-Suc Z* **in** *auto*)
have *n-ge1*: $\forall_F\ t$ *in at-top*. $1 \leq nat\ \lfloor log\ Z\ t \rfloor$ **if** *Z*: $1 < Z$ **for** *Z* :: *real*
using *n-at-top*[*OF Z*, *unfolded filterlim-at-top*, *rule-format*, *of 1*]
by *eventually-elim simp*

have *sandwich-upper*: $\forall_F\ t$ *in at-top*. $g\ (t * y) / g\ t$
$\leq g\ (Z \mathbin{\hat{}} (nat\ \lfloor log\ Z\ t \rfloor + m + 2)) / g\ (Z \mathbin{\hat{}} nat\ \lfloor log\ Z\ t \rfloor)$
if *y1*: $1 \leq y$ **and** *Z1*: $1 < Z$ **and** *mlo*: $Z \mathbin{\hat{}} m \leq y$ **and** *mhi*: $y < Z \mathbin{\hat{}} (m + 1)$
for *Z* :: *real* **and** *m* :: *nat* **and** *y* :: *real*
using *t-bracket*[*OF Z1*] *n-ge1*[*OF Z1*] *Un-pos-ev*[*OF Z1*]
proof *eventually-elim*
case (*elim t*)
define *n* **where** $n = nat\ \lfloor log\ Z\ t \rfloor$

```
  have tb1: Z ^ n ≤ t and tb2: t < Z ^ (n + 1)
    using elim unfolding n-def by auto
  have n1: 1 ≤ n
    using elim unfolding n-def by simp
  have Zn-gt1: 1 < Z ^ n
    using Z1 n1 by simp
  have Un-pos: 0 < g (Z ^ n)
    using elim unfolding n-def by simp
  have tpos: 0 < t
    using Zn-gt1 tb1 by simp
  have ypos: 0 < y
    using y1 by simp
  have ty-lt: t * y < Z ^ (n + m + 2)
  proof −
    have t * y < Z ^ (n + 1) * Z ^ (m + 1)
      using tb2 mhi tpos ypos by (intro mult-strict-mono') auto
    also have ... = Z ^ (n + m + 2)
      by (simp add: power-add)
    finally show ?thesis .
  qed
  have t-gt1: 1 < t
    using Zn-gt1 tb1 by simp
  have ty-gt1: 1 < t * y
    using y1 t-gt1 mult-left-mono[of 1 y t] by linarith
  have Uty-le: g (t * y) ≤ g (Z ^ (n + m + 2))
    using gmono[OF ty-gt1] ty-lt by simp
  have Ut-ge: g (Z ^ n) ≤ g t
    using gmono[OF Zn-gt1 tb1] .
  have Ut-pos: 0 < g t
    using Un-pos Ut-ge by simp
  have Unm2-ge: Z ^ n ≤ Z ^ (n + m + 2)
    using Z1 by (intro power-increasing) auto
  have Unm2-gt1: 1 < Z ^ (n + m + 2)
    using Unm2-ge Zn-gt1 by simp
  have Utym2-pos: 0 < g (Z ^ (n + m + 2))
    using gmono[OF Zn-gt1 Unm2-ge] Un-pos by simp
  show ?case
    unfolding n-def[symmetric]
    using frac-le[OF Utym2-pos[THEN less-imp-le] Uty-le Un-pos Ut-ge] .
qed
have sandwich-lower:
  ∀F t in at-top. g (Z ^ (nat ⌊log Z t⌋ + m)) / g (Z ^ (nat ⌊log Z t⌋ + 1))
    ≤ g (t * y) / g t
  if y1: 1 ≤ y and Z1: 1 < Z and mlo: Z ^ m ≤ y
  for Z :: real and m :: nat and y :: real
  using t-bracket[OF Z1] n-ge1[OF Z1] Un-pos-ev[OF Z1]
proof eventually-elim
  case (elim t)
  define n where n = nat ⌊log Z t⌋
```

```
  have tb1: Z ^ n ≤ t and tb2: t < Z ^ (n + 1)
    using elim unfolding n-def by auto
  have n1: 1 ≤ n
    using elim unfolding n-def by simp
  have Zn-gt1: 1 < Z ^ n
    using Z1 n1 by simp
  have Un-pos: 0 < g (Z ^ n)
    using elim unfolding n-def by simp
  have Znm-ge: Z ^ n ≤ Z ^ (n + m)
    using Z1 by (intro power-increasing) auto
  have Znm-gt1: 1 < Z ^ (n + m)
    using Znm-ge Zn-gt1 by simp
  have ty-ge: Z ^ (n + m) ≤ t * y
    using tb1 mlo Z1 Zn-gt1 by(auto intro!: mult-mono simp: power-add)
  have Unm-pos: 0 < g (Z ^ (n + m))
    using gmono[OF Zn-gt1 Znm-ge] Un-pos by simp
  have t-gt1: 1 < t
    using Zn-gt1 tb1 by simp
  show ?case
    unfolding n-def[symmetric]
    by(intro frac-le[OF less-imp-le] gmono Znm-gt1 ty-ge
        t-gt1 tb2[THEN less-imp-le]
        order.strict-trans2[OF Un-pos gmono[OF Zn-gt1 tb1]]
        order.strict-trans2[OF Unm-pos gmono[OF Znm-gt1 ty-ge]])
qed

have main: ((λt. g (t * y) / g t) ⟶ y powr ϱ) at-top
  if y1: 1 ≤ y for y
proof −
  have Z-of-ratio: ∃ Z. 1 < Z ∧ Z ^ 2 < R if Rgt1: 1 < R for R :: real
  proof −
    define B where B = sqrt R
    define Z where Z = (1 + B) / 2
    have B1: 1 < B
      unfolding B-def using real-sqrt-less-mono[OF Rgt1] by simp
    have Zbounds1: 1 < Z
      unfolding Z-def using B1 by argo
    have Zbounds2: Z < B
      unfolding Z-def using B1 by argo
    have ZltB2: Z ^ 2 < B ^ 2
      using Zbounds1 Zbounds2 by (simp add: power-strict-mono)
    have B2: B ^ 2 = R
      unfolding B-def using Rgt1 by simp
    show ?thesis
      using Zbounds1 ZltB2 B2 by (intro exI[of - Z]) simp
  qed
  show ?thesis
  proof (rule order-tendstoI)
    fix a :: real
```

assume *a*: $a < y$ *powr* ϱ
show $\forall_F$ *t in at-top*. $a < g\ (t * y)\ /\ g\ t$
proof (*cases* $a \leq 0$)
case *True*
have *ypos*: $0 < y$
using *y1* **by** *simp*
have *y-at-top*: *filterlim* (λt::*real*. $t * y$) *at-top at-top*
using *filterlim-tendsto-pos-mult-at-top*[*OF tendsto-const*[*of y*] *ypos filterlim-ident*]
by (*simp add*: *mult.commute*)
have *ev-pos*: $\forall_F$ *t in at-top*. $0 < g\ (t * y)\ /\ g\ t$
using *eventually-compose-filterlim*[*OF gpos y-at-top*] *gpos*
by *eventually-elim simp*
show *?thesis*
using *ev-pos*
by *eventually-elim* (*use True* **in** *linarith*)
next
case *False*
hence *apos*: $0 < a$
by *simp*
have *alt*: *a powr* $(1\ /\ \varrho) < y$
using *powr-less-mono2*[*OF - - a, of* $1\ /\ \varrho$] *apos rho y1*
by (*simp add*: *powr-powr*)
have *ratio-gt1*: $1 < y\ /\ a$ *powr* $(1\ /\ \varrho)$
using *alt apos* **by** (*simp add*: *field-simps*)
obtain *Z* **where** *Zbounds1*: $1 < Z$ **and** *ZltR*: $Z \hat{} 2 < y\ /\ a$ *powr* $(1\ /\ \varrho)$
using *Z-of-ratio*[*OF ratio-gt1*] **by** *auto*
have *Zpos*: $0 < Z$
using *Zbounds1* **by** *simp*
have *key*: $a < (y\ /\ Z \hat{} 2)$ *powr* ϱ
proof −
have *a powr* $(1\ /\ \varrho) < y\ /\ Z \hat{} 2$
using *ZltR Zpos apos* **by** (*simp add*: *field-simps*)
hence (*a powr* $(1\ /\ \varrho)$) *powr* $\varrho < (y\ /\ Z \hat{} 2)$ *powr* ϱ
using *rho apos Zpos y1* **by** (*intro powr-less-mono2*) *auto*
also have (*a powr* $(1\ /\ \varrho)$) *powr* $\varrho = a$
using *rho apos* **by** (*simp add*: *powr-powr*)
finally show *?thesis* **.**
qed
define *m* **where** $m = nat\ \lfloor log\ Z\ y \rfloor$
have *ybracket*: $Z \hat{} m \leq y\ y < Z \hat{} (m + 1)$
using *power-of-nat-log-le power-of-nat-log-less-Suc Zbounds1 y1*
unfolding *m-def* **by** *auto*
have *tel-m*: $((\lambda t.\ g\ (Z \hat{} (nat\ \lfloor log\ Z\ t \rfloor + m))$
$/\ g\ (Z \hat{} nat\ \lfloor log\ Z\ t \rfloor)) \longrightarrow (Z\ powr\ \varrho) \hat{} m)$ *at-top*
using *telescope-cont*[*OF Zbounds1, of m*] **.**
have *tel-1*: $((\lambda t.\ g\ (Z \hat{} (nat\ \lfloor log\ Z\ t \rfloor + 1))$
$/\ g\ (Z \hat{} nat\ \lfloor log\ Z\ t \rfloor)) \longrightarrow Z\ powr\ \varrho)$ *at-top*
using *telescope-cont*[*OF Zbounds1, of 1*] **by** *simp*

```
    have Zϱne: Z powr ϱ ≠ 0
      using Zpos by simp
    have div-lim: ((λt. (g (Z ^ (nat ⌊log Z t⌋ + m))
                          / g (Z ^ nat ⌊log Z t⌋))
                        / (g (Z ^ (nat ⌊log Z t⌋ + 1))
                          / g (Z ^ nat ⌊log Z t⌋)))
                   ⟶ (Z powr ϱ) ^ m / Z powr ϱ) at-top
      by (intro tendsto-divide[OF tel-m tel-1 Zϱne])
    have eq-ev: ∀F t in at-top. (g (Z ^ (nat ⌊log Z t⌋ + m)) / g (Z ^ nat ⌊log
Z t⌋))
                / (g (Z ^ (nat ⌊log Z t⌋ + 1)) / g (Z ^ nat ⌊log Z t⌋))
              = g (Z ^ (nat ⌊log Z t⌋ + m)) / g (Z ^ (nat ⌊log Z t⌋ + 1))
      using Un-pos-ev[OF Zbounds1]
      by eventually-elim (simp add: field-simps)
    have ratio-lim: ((λt. g (Z ^ (nat ⌊log Z t⌋ + m))
                          / g (Z ^ (nat ⌊log Z t⌋ + 1)))
                    ⟶ (Z powr ϱ) ^ m / Z powr ϱ) at-top
      using Lim-transform-eventually[OF div-lim eq-ev] .
    have val-gt: a < (Z powr ϱ) ^ m / Z powr ϱ
    proof −
      have y / Z ^ 2 < Z ^ m / Z
        using ybracket(2) Zpos by (simp add: field-simps power2-eq-square)
      moreover have 0 ≤ y / Z ^ 2
        using y1 Zpos by simp
      ultimately have (y / Z ^ 2) powr ϱ < (Z ^ m / Z) powr ϱ
        using rho by (intro powr-less-mono2)
      also have (Z ^ m / Z) powr ϱ = (Z ^ m) powr ϱ / Z powr ϱ
        by (simp add: powr-divide)
      also have (Z ^ m) powr ϱ = (Z powr ϱ) ^ m
        by (metis Zpos less-eq-real-def linorder-not-less powr-power powr-powr
powr-realpow)
      finally show ?thesis
        using key by simp
    qed
    have ev-val: ∀F t in at-top. a <
          g (Z ^ (nat ⌊log Z t⌋ + m)) / g (Z ^ (nat ⌊log Z t⌋ + 1))
      using order-tendstoD(1)[OF ratio-lim val-gt] .
    show ?thesis
      using ev-val sandwich-lower[OF y1 Zbounds1 ybracket(1)]
      by eventually-elim linarith
  qed
 next
  fix c :: real
  assume c: y powr ϱ < c
  have cpos: 0 < c
    by(auto intro!: order.strict-trans1[OF - c])
  have alt: y < c powr (1 / ϱ)
    using powr-less-mono2[OF - - c, of 1 / ϱ] rho y1
    by (simp add: powr-powr)
```

```
    have ratio-gt1: 1 < c powr (1 / ϱ) / y
      using alt y1 by (simp add: field-simps)
    obtain Z where Zbounds1: 1 < Z and ZltR: Z ^ 2 < c powr (1 / ϱ) / y
      using Z-of-ratio[OF ratio-gt1] by auto
    have Zpos: 0 < Z
      using Zbounds1 by simp
    have key: (y * Z ^ 2) powr ϱ < c
    proof −
      have y * Z ^ 2 < c powr (1 / ϱ)
        using ZltR y1 by (simp add: field-simps)
      hence (y * Z ^ 2) powr ϱ < (c powr (1 / ϱ)) powr ϱ
        using rho y1 Zpos by (intro powr-less-mono2) auto
      also have (c powr (1 / ϱ)) powr ϱ = c
        using rho cpos by (simp add: powr-powr)
      finally show ?thesis .
    qed
    define m where m = nat ⌊log Z y⌋
    have ybracket: Z ^ m ≤ y y < Z ^ (m + 1)
      using  power-of-nat-log-le power-of-nat-log-less-Suc Zbounds1 y1
      unfolding m-def by auto
    have tel-m2: ((λt. g (Z ^ (nat ⌊log Z t⌋ + (m + 2)))
                    / g (Z ^ nat ⌊log Z t⌋)) ⟶ (Z powr ϱ) ^ (m + 2)) at-top
      using telescope-cont[OF Zbounds1, of m + 2] .
    have val-lt: (Z powr ϱ) ^ (m + 2) < c
    proof −
      have step1: Z ^ (m + 2) = Z ^ m * Z ^ 2
        by (rule power-add)
      have step2: Z ^ m * Z ^ 2 ≤ y * Z ^ 2
        using ybracket(1) Zpos by (intro mult-right-mono) auto
      have step3: Z ^ (m + 2) ≤ y * Z ^ 2
        unfolding step1 by (rule step2)
      have step3pos: 0 ≤ Z ^ (m + 2)
        using Zpos by simp
      have step4: (Z ^ (m + 2)) powr ϱ ≤ (y * Z ^ 2) powr ϱ
        by (rule powr-mono2[OF less-imp-le[OF rho] step3pos step3])
      have step5: (Z ^ (m + 2)) powr ϱ < c
        using step4 key by linarith
      have step6: (Z ^ (m + 2)) powr ϱ = (Z powr ϱ) ^ (m + 2)
          by (metis Zpos less-eq-real-def linorder-not-less powr-power powr-powr
powr-realpow)
      show ?thesis
        using step5 step6 by simp
    qed
    have ev-val:
     ∀F t in at-top. g (Z ^ (nat ⌊log Z t⌋ + (m + 2))) / g (Z ^ nat ⌊log Z t⌋) <
c
      using order-tendstoD(2)[OF tel-m2 val-lt] .
    have ev-val′:
     ∀F t in at-top. g (Z ^ (nat ⌊log Z t⌋ + m + 2)) / g (Z ^ nat ⌊log Z t⌋) < c
```

```
      using ev-val by (simp add: add.assoc)
    show ∀F t in at-top. g (t * y) / g t < c
      using ev-val' sandwich-upper[OF y1 Zbounds1 ybracket(1) ybracket(2)]
      by eventually-elim linarith
  qed
 qed
 show ?thesis
 proof (cases 1 ≤ y)
  case False
  hence y01: 0 < y y < 1
   using y by auto
  have yinv1: 1 ≤ 1 / y
   using y01 by (simp add: field-simps)
  have base: ((λt. g (t * (1 / y)) / g t)
          ⟶ (1 / y) powr ϱ) at-top
   using main[OF yinv1] .
  have ty-at-top: filterlim (λs::real. s * y) at-top at-top
    using filterlim-tendsto-pos-mult-at-top[OF tendsto-const[of y] y01(1) filter-
lim-ident]
   by (simp add: mult.commute)
  have comp: ((λs. g (s * y * (1 / y)) / g (s * y))
            ⟶ (1 / y) powr ϱ) at-top
   by (rule filterlim-compose[OF base ty-at-top])
  have simp1: s * y * (1 / y) = s for s :: real
   using y01 by simp
  have base2: ((λs. g s / g (s * y)) ⟶ (1 / y) powr ϱ) at-top
   using comp y01 by (simp add: simp1)
  have nz: (1 / y) powr ϱ ≠ 0
   using y01 by simp
  have recip: ((λs. inverse (g s / g (s * y)))
            ⟶ inverse ((1 / y) powr ϱ)) at-top
   by (rule tendsto-inverse[OF base2 nz])
  note ev-ne = tendsto-imp-eventually-ne[OF base2 nz]
  have eq: ∀F s in at-top. inverse (g s / g (s * y))
                = g (s * y) / g s
   using ev-ne by eventually-elim (simp add: field-simps)
  have final: ((λs. g (s * y) / g s)
            ⟶ inverse ((1 / y) powr ϱ)) at-top
   using Lim-transform-eventually[OF recip eq] .
  have val: inverse ((1 / y) powr ϱ) = y powr ϱ
   by (simp add: powr-divide)
  show ?thesis
   using final val by simp
 qed(simp add: main)
qed

lemma galois-inverse-ratio-tendsto:
 fixes W s :: real ⇒ real and ϱ x :: real
 assumes W: filterlim W at-top at-top
```

```
    and s: filterlim s at-top at-top
    and s-pos: ⋀w. 0 < w ⟹ 0 < s w
    and le-iff: ⋀t w. 1 < t ⟹ (W t ≤ w) = (t ≤ s w)
    and rho[arith]: 0 < ϱ
    and lim: ⋀y. 0 < y ⟹ ((λt. W (t * y) / W t) ⟶ y powr ϱ) at-top
    and x[arith]: 0 < x
  shows ((λu. s (u * x) / s u) ⟶ x powr (1 / ϱ)) at-top
proof −
  have galois-strict: (w < W t) = (s w < t) if t1: 1 < t for t w
    using le-iff[OF t1, of w] by linarith
  have s-gt1: ∀F u in at-top. 1 < s u
    using s unfolding filterlim-at-top-dense by blast
  have Ws-pos: ∀F u in at-top. 0 < W (s u)
    using filterlim-compose[OF W s] unfolding filterlim-at-top-dense by blast
  have s-gt1-mult: ∀F u in at-top. 1 < s u * c if c: 0 < c for c
  proof −
    have ∀F u in at-top. 1 / c < s u
      using s unfolding filterlim-at-top-dense by blast
    thus ?thesis
      by eventually-elim (use c in simp add: field-simps)
  qed
  have u-pos: ∀F u in at-top. (0::real) < u
    by (rule eventually-gt-at-top)
  have upper: ∀F u in at-top. W (s u) ≤ u
    using s-gt1 by eventually-elim (use le-iff in auto)
  have near-inv: ((λu. W (s u) / u) ⟶ 1) at-top
    unfolding tendsto-iff dist-real-def
  proof safe
    fix e :: real
    assume e: 0 < e
    define e' where e' = min (1 / 2) e
    have e'-pos[arith]: 0 < e' and e'-le: e' ≤ e and e'-lt1: e' < 1
      using e by (simp-all add: e'-def)
    have cpos[arith]: 0 < 1 − e'
      using e'-lt1 by simp
    have c-lt1: 1 − e' < 1
      using e'-pos by simp
    define K where K = (1 / (1 − e')) powr (1 / ϱ)
    have K-gt1[arith]: 1 < K
      using rho cpos c-lt1 by (simp add: K-def)
    define δ where δ = (K − 1) / 2
    have δ-pos[arith]: 0 < δ
      using K-gt1 by (simp add: δ-def)
    have δ1-pos: 0 < 1 + δ
      using δ-pos by simp
    have δ-lt-K: 1 + δ < K
      using K-gt1 by (simp add: δ-def field-simps)
    have δ-lt: (1 + δ) powr ϱ < 1 / (1 − e')
      using powr-less-mono2[OF rho - δ-lt-K] δ-pos rho cpos
```

```
    by (simp add: K-def powr-powr)
  have ev-ratio: ∀F u in at-top. W (s u * (1 + δ)) / W (s u) < 1 / (1 - e')
    using order-tendstoD(2)[OF filterlim-compose[OF lim[OF δ1-pos] s] δ-lt] .
  have ev-lower: ∀F u in at-top. u < W (s u * (1 + δ))
    using s-gt1-mult[OF δ1-pos] s-gt1
  proof eventually-elim
    case (elim u)
    hence s u < s u * (1 + δ)
      using δ-pos by simp
    thus ?case
      using galois-strict[OF elim(1), of u] by simp
  qed
  have lower: ∀F u in at-top. 1 - e' < W (s u) / u
    using ev-ratio ev-lower upper Ws-pos u-pos
  proof eventually-elim
    case (elim u)
    have u / W (s u) < W (s u * (1 + δ)) / W (s u)
      using elim(2,4) by (rule divide-strict-right-mono)
    also have ... < 1 / (1 - e')
      using elim(1) .
    finally have lt: u / W (s u) < 1 / (1 - e') .
    have u * (1 - e') < W (s u)
      using lt elim(4,5) cpos by (simp add: field-simps)
    thus ?case
      using elim(5) cpos by (simp add: field-simps)
  qed
  have e'1: 1 < 1 + e'
    using e'-pos by simp
  have upperb: ∀F u in at-top. W (s u) / u < 1 + e'
    using upper u-pos
  proof eventually-elim
    case (elim u)
    have W (s u) < u * (1 + e')
      using elim(1) mult-strict-left-mono[OF e'1 elim(2)] by simp
    thus ?case
      using elim(2) by (simp add: divide-less-eq mult.commute)
  qed
  show ∀F u in at-top. |W (s u) / u - 1| < e
    using lower upperb
    by eventually-elim (use e'-le in linarith)
qed
have Winv-ratio: ((λu. W (s u * y) / u) ⟶ y powr ϱ) at-top if y: 0 < y for
y
proof -
  have eq: ∀F u in at-top.
      (W (s u * y) / W (s u)) * (W (s u) / u) = W (s u * y) / u
    using Ws-pos by eventually-elim (simp add: field-simps)
  from Lim-transform-eventually[OF
      tendsto-mult[OF filterlim-compose[OF lim[OF y] s] near-inv] eq]
```

```
  show ?thesis
    by simp
qed
define L where L = x powr (1 / ϱ)
have L-pos: 0 < L
  using x by (simp add: L-def)
show ?thesis
  unfolding L-def[symmetric] tendsto-iff dist-real-def
proof safe
  fix e :: real
  assume e: 0 < e
  define e' where e' = min (L / 2) e
  have e'-pos: 0 < e' and e'-le: e' ≤ e and e'-lt: e' < L
    using e L-pos by (simp-all add: e'-def)
  have cpos: 0 < L − e'
    using e'-lt by simp
  define ylo where ylo = (L − e' + L) / 2
  have ylo-pos: 0 < ylo
    using cpos L-pos by (simp add: ylo-def)
  have c-lt-ylo: L − e' < ylo
    using e'-pos by (simp add: ylo-def)
  have ylo-lt-L: ylo < L
    using e'-pos by (simp add: ylo-def)
  have ylo-powr-lt-x: ylo powr ϱ < x
    using powr-less-mono2[OF rho less-imp-le[OF ylo-pos] ylo-lt-L] x rho
    by (simp add: L-def powr-powr)
  have ev-lt: ∀ F u in at-top. W (s u * ylo) < u * x
    using order-tendstoD(2)[OF Winv-ratio[OF ylo-pos] ylo-powr-lt-x] u-pos
    by eventually-elim (simp add: field-simps)
  have lower: ∀ F u in at-top. L − e' < s (u * x) / s u
    using ev-lt s-gt1-mult[OF ylo-pos] u-pos
  proof eventually-elim
    case (elim u)
    have le: s u * ylo ≤ s (u * x)
      using le-iff[OF elim(2), of u * x] elim(1) by simp
    have (L − e') * s u < ylo * s u
      using c-lt-ylo s-pos[OF elim(3)] by (intro mult-strict-right-mono) auto
    also have ... ≤ s (u * x)
      using le by (simp add: mult.commute)
    finally have (L − e') * s u < s (u * x) .
    thus ?case
      using s-pos[OF elim(3)] by (simp add: field-simps)
  qed
  define yhi where yhi = (L + (L + e')) / 2
  have yhi-pos: 0 < yhi
    using L-pos e'-pos by (simp add: yhi-def)
  have yhi-lt-c: yhi < L + e'
    using e'-pos by (simp add: yhi-def)
  have L-lt-yhi: L < yhi
```

```
    using e'-pos by (simp add: yhi-def)
  have x-lt-yhi-powr: x < yhi powr ϱ
    using powr-less-mono2[OF rho less-imp-le[OF L-pos] L-lt-yhi] x rho
    by (simp add: L-def powr-powr)
  have ev-gt: ∀F u in at-top. u * x < W (s u * yhi)
    using order-tendstoD(1)[OF Winv-ratio[OF yhi-pos] x-lt-yhi-powr] u-pos
    by eventually-elim (simp add: field-simps)
  have upperb: ∀F u in at-top. s (u * x) / s u < L + e'
    using ev-gt s-gt1-mult[OF yhi-pos] u-pos
  proof eventually-elim
    case (elim u)
    have s (u * x) < s u * yhi
      using galois-strict[OF elim(2)] elim(1) by simp
    also have s u * yhi < s u * (L + e')
      using yhi-lt-c s-pos[OF elim(3)] by (intro mult-strict-left-mono) auto
    finally have s (u * x) < s u * (L + e') .
    thus ?case
      using s-pos[OF elim(3)] by (simp add: field-simps)
  qed
  show ∀F u in at-top. |s (u * x) / s u - L| < e
    using lower upperb
    by eventually-elim (use e'-le in linarith)
 qed
qed

end
```

2 Distributions

2.1 Generalized Extreme Value Distribution

```
theory Extreme-Value-Distribution
imports
  Lemmas-EVT
begin
```

We define the generalized extreme value (GEV) density and cdf. Our definition includes a location parameter μ and a scale parameter σ.

```
definition EVD-density :: [real, real, real, real] ⇒ real where
EVD-density ξ μ σ x ≡ let y = (x - μ) / σ in
                      if ξ = 0 then
                        exp (- y) * exp (- exp (- y)) / σ
                      else
                        if ξ * y > - 1 then
                        (1 + ξ * y) powr (- 1 / ξ - 1)
                         * exp (- ((1 + ξ * y) powr (- 1 / ξ))) / σ
                        else 0

lemma EVD-density-nonneg[simp]:
```

$0 \leq \sigma \Longrightarrow 0 \leq$ *EVD-density* $\xi\ \mu\ \sigma\ x$
by(*simp add*: *EVD-density-def Let-def*)

lemma *EVD-density-xi-0*:
EVD-density 0 $\mu\ \sigma\ x =$ *exp* $(-\ ((x - \mu)\ /\ \sigma)) *$ *exp* $(-$ *exp* $(-\ ((x - \mu)\ /\ \sigma)))\ /\ \sigma$
by(*simp add*: *EVD-density-def Let-def*)

lemma
assumes $\xi \neq 0$
shows *EVD-density-xi-non0-1*:
$\xi * (x - \mu)\ /\ \sigma \geq -\ 1$
$\Longrightarrow$ *EVD-density* $\xi\ \mu\ \sigma\ x =$ $(1 + \xi * (x - \mu)\ /\ \sigma)$ *powr* $(-\ 1\ /\ \xi - 1)$
$* exp\ (-\ ((1 + \xi * (x - \mu)\ /\ \sigma)\ powr\ (-\ 1\ /\ \xi)))\ /\ \sigma$
and *EVD-density-xi-non0-2*:
$\xi * (x - \mu)\ /\ \sigma \leq -\ 1 \Longrightarrow$ *EVD-density* $\xi\ \mu\ \sigma\ x = 0$
using *assms* **by**(*auto simp*: *EVD-density-def order.order-iff-strict*)

lemma *EVD-density-measurable*[*measurable*]:
EVD-density $\xi\ \mu\ \sigma \in$ *borel-measurable borel*
unfolding *EVD-density-def Let-def* **by** *simp*

lemma
assumes *sigma*[*arith*]:$0 < \sigma$
shows *EVD-density-set-integrable0*:*set-integrable lborel* $\{..a\}$ (*EVD-density 0* μ σ)
and *EVD-density-set-integral0*:
$(\int x{\in}\{..a\}.$ *EVD-density 0* $\mu\ \sigma\ x\ \partial lborel) =$ *exp* $(-$ *exp* $(-\ (a - \mu)\ /\ \sigma))$
and *EVD-density-set-nn-integral0*:
$(\int^{+} x{\in}\{..a\}.$ *EVD-density 0* $\mu\ \sigma\ x\ \partial lborel) =$ *ennreal* (*exp* $(-$ *exp* $(-\ (a - \mu)\ /\ \sigma)))$
proof −
define G **where** $G = (\lambda x{::}real.$ *exp* $(-$ *exp* $(-\ (x - \mu)\ /\ \sigma)))$
have *deriv*: (G *has-real-derivative EVD-density 0* $\mu\ \sigma\ x$) (*at* x) **for** x
unfolding *G-def EVD-density-def Let-def*
using *sigma* **by** (*auto intro*!: *derivative-eq-intros*
simp: *field-simps simp del*: *minus-diff-eq*)
have *tendsto-bot*: ($G \longrightarrow 0$) *at-bot*
unfolding *G-def* **using** *sigma*
by(*intro filterlim-compose*[*OF exp-at-bot filterlim-compose*
[*OF filterlim-uminus-at-bot-at-top*]]
filterlim-compose[*OF exp-at-top*])
real-asymp
have *cont*: *continuous* (*at-left a*) G
using *continuous-at-imp-continuous-at-within*[*OF DERIV-isCont*[*OF deriv*]] **by** *simp*
show *set-integrable lborel* $\{..a\}$ (*EVD-density 0* μ σ)
and $(\int x{\in}\{..a\}.$ *EVD-density 0* $\mu\ \sigma\ x\ \partial lborel) =$ *exp* $(-$ *exp* $(-\ (a - \mu)\ /\ \sigma))$
and $(\int^{+} x{\in}\{..a\}.$ *EVD-density 0* $\mu\ \sigma\ x\ \partial lborel) =$ *ennreal* (*exp* $(-$ *exp* $(-$ (*a*

```
− μ) / σ)))
    using set-integrable-FTC-atMost[OF - cont deriv - tendsto-bot]
      set-integral-FTC-atMost[OF - cont deriv - tendsto-bot]
      nn-integral-FTC-lessThan[OF - cont deriv - tendsto-bot]
    by(simp-all add: G-def set-nn-integral-atMost-eq)
qed

lemma
  fixes a μ σ
  defines y ≡ (a − μ) / σ
  assumes sigma[arith]:0 < σ
    and xi:ξ ≠ 0
    and xiy:ξ * y ≥ − 1
    and xiy-pos:0 < ξ ⟹ − 1 < ξ * y
  shows EVD-density-set-integrable1:set-integrable lborel {..a} (EVD-density ξ μ
σ)
    and EVD-density-set-integral1:
    (∫ x∈{..a}. EVD-density ξ μ σ x ∂lborel) = exp (− ((1 + ξ * y) powr (− 1 /
ξ)))
    and EVD-density-set-nn-integral1:
    (∫⁺x∈{..a}. EVD-density ξ μ σ x ∂lborel) = ennreal (exp (− ((1 + ξ * y)
powr (− 1 / ξ))))
proof −
  have deriv-raw: ((λx. exp (− ((1 + ξ * ((x − μ) / σ)) powr (− 1 / ξ))))
has-real-derivative
                  EVD-density ξ μ σ x) (at x)
    if pos: 0 < 1 + ξ * ((x − μ) / σ) for x
  proof −
    have cond: − 1 < ξ * ((x − μ) / σ)
      using pos by linarith
    have dens: EVD-density ξ μ σ x
       = (1 + ξ * ((x − μ) / σ)) powr (− 1 / ξ − 1)
          * exp (− ((1 + ξ * ((x − μ) / σ)) powr (− 1 / ξ))) / σ
      using cond xi by(simp add: EVD-density-def Let-def)
    show ?thesis
      unfolding dens
      using pos xi by(auto intro!: derivative-eq-intros simp: field-simps)
  qed
  show nn: (∫⁺x∈{..a}. EVD-density ξ μ σ x ∂lborel)
          = ennreal (exp (− ((1 + ξ * y) powr (− 1 / ξ))))
  proof(cases 0 < ξ)
    case False
    with xi have xi-neg: ξ < 0
      by simp
    have p-pos: 0 < − 1 / ξ
      using xi-neg by (simp add: divide-pos-neg)
    define F where F ≡ λx. exp (− ((1 + ξ * ((x − μ) / σ)) powr (− 1 / ξ)))
    have pos: 0 < 1 + ξ * ((x − μ) / σ) if x < a for x
    proof −
```

```
    have (x − μ) / σ < (a − μ) / σ
      using that sigma by (intro divide-strict-right-mono) auto
    hence ξ * ((a − μ) / σ) < ξ * ((x − μ) / σ)
      by (rule mult-strict-left-mono-neg[OF - xi-neg])
    with xiy show ?thesis
      unfolding y-def by linarith
  qed
  have deriv: (F has-real-derivative EVD-density ξ μ σ x) (at x) if x < a for x
    unfolding F-def by (rule deriv-raw[OF pos[OF that]])
  have nonneg: 0 ≤ EVD-density ξ μ σ x for x
    by simp
  have cont: continuous (at-left a) F
  proof (cases ξ * y = − 1)
    case False
    with xiy have pos-a: 0 < 1 + ξ * y
      by linarith
    have isCont F a
      unfolding F-def using pos-a
      by (intro continuous-intros) (simp-all add: y-def)
    thus ?thesis
      using continuous-at-imp-continuous-at-within by blast
  next
    case True
    have Fa: F a = 1
      using True unfolding F-def y-def by simp
    have ((λx. 1 + ξ * ((x − μ) / σ)) ⟶ 1 + ξ * ((a − μ) / σ)) (at-left a)
      by (intro tendsto-intros) arith
    with True have base0: ((λx. 1 + ξ * ((x − μ) / σ)) ⟶ 0) (at-left a)
      by (simp add: y-def)
    have ev-pos: ∀F x in at-left a. 0 ≤ 1 + ξ * ((x − μ) / σ)
      by (intro eventually-at-leftI[of a − 1]) (use pos[THEN less-imp-le] in auto)
    have ((λx. (1 + ξ * ((x − μ) / σ)) powr (− 1 / ξ)) ⟶ 0) (at-left a)
      by (rule tendsto-zero-powrI[OF base0 tendsto-const ev-pos p-pos])
    hence (F ⟶ exp (− 0)) (at-left a)
      unfolding F-def by (intro tendsto-intros)
    thus ?thesis
      by (simp add: continuous-within Fa)
  qed
  have lim: (F ⟶ 0) at-bot
    unfolding F-def
    using filterlim-compose[OF real-powr-at-top[OF p-pos]
        filterlim-shift-scale-at-bot[OF sigma xi-neg]]
    by (intro filterlim-compose[OF exp-at-bot
      filterlim-compose[OF filterlim-uminus-at-bot-at-top]])
  have (∫⁺x∈{..<a}. EVD-density ξ μ σ x ∂lborel)
      = ennreal (exp (− ((1 + ξ * y) powr (− 1 / ξ))))
    using nn-integral-FTC-lessThan[OF EVD-density-measurable cont deriv non-
neg lim]
    by (simp add: F-def y-def)
```

```
    thus ?thesis
      by (simp add: set-nn-integral-atMost-eq[symmetric])
  next
    case True
    have p-neg: − 1 / ξ < 0
      using True by (simp add: divide-neg-pos)
    define c where c ≡ μ − σ / ξ
    define F where F ≡ λx. if − 1 < ξ ∗ ((x − μ) / σ)
                        then exp (− ((1 + ξ ∗ ((x − μ) / σ)) powr (− 1 / ξ))) else 0
    have cond-iff: (− 1 < ξ ∗ ((x − μ) / σ)) ⟷ c < x for x
      using True sigma unfolding c-def by (auto simp: field-simps)
    have ca: c < a
      using xiy-pos[OF True] cond-iff[of a] by (simp add: y-def)
    have dens0: EVD-density ξ μ σ x = 0 if ¬ c < x for x
      using cond-iff[of x] that xi by (auto simp: EVD-density-def Let-def)
    have pos: 0 < 1 + ξ ∗ ((x − μ) / σ) if c < x for x
      using cond-iff[THEN iffD2, OF that] by linarith
    have F-eq: F x = exp (− ((1 + ξ ∗ ((x − μ) / σ)) powr (− 1 / ξ))) if c < x
for x
      using cond-iff that by (simp add: F-def)
    have deriv: (F has-real-derivative EVD-density ξ μ σ x) (at x) if c < x for x
    proof −
     have ((λx. exp (− ((1 + ξ ∗ ((x − μ) / σ)) powr (− 1 / ξ)))) has-real-derivative
            EVD-density ξ μ σ x) (at x)
       by (rule deriv-raw[OF pos[OF that]])
      thus ?thesis
        by (rule has-field-derivative-transform-within-open[where S={c<..}])
          (auto simp: F-eq that)
    qed
    have deriv': (F has-real-derivative EVD-density ξ μ σ x) (at x) if a < x for x
      by (rule deriv[OF less-trans[OF ca that]])
    have nonneg: 0 ≤ EVD-density ξ μ σ x for x
      by simp
    have Fc: F c = 0
      using cond-iff[of c] by (simp add: F-def)
    have contc: continuous (at-right c) F
    proof −
      have (λx. 1 + ξ ∗ ((x − μ) / σ)) ⟶ 1 + ξ ∗ ((c − μ) / σ)) (at-right c)
        by (intro tendsto-intros) arith
      moreover have 1 + ξ ∗ ((c − μ) / σ) = 0
        using True sigma by (simp add: c-def)
      ultimately have base0: ((λx. 1 + ξ ∗ ((x − μ) / σ)) ⟶ 0) (at-right c)
        by simp
      have ev-pos: ∀F x in at-right c. 0 < 1 + ξ ∗ ((x − μ) / σ)
        by (intro eventually-at-rightI[of c c + 1]) (use pos in auto)
     have small: ((λx. (1 + ξ ∗ ((x − μ) / σ)) powr (1 / ξ)) ⟶ 0) (at-right c)
        by (rule tendsto-zero-powrI[OF base0 tendsto-const])
          (use ev-pos True in ‹auto elim: eventually-mono›)
      have ev-pos': ∀F x in at-right c. 0 < (1 + ξ ∗ ((x − μ) / σ)) powr (1 / ξ)
```

```
    using ev-pos by (auto elim!: eventually-mono)
  have top: LIM x (at-right c). (1 + ξ * ((x − μ) / σ)) powr (− 1 / ξ) :>
at-top
    using filterlim-inverse-at-top[OF small ev-pos'] by (simp add: powr-minus)
  have ∗: ((λx. exp (− ((1 + ξ * ((x − μ) / σ)) powr (− 1 / ξ)))) ⟶ 0)
(at-right c)
    by (rule filterlim-compose[OF exp-at-bot
        filterlim-compose[OF filterlim-uminus-at-bot-at-top top]])
  have (F ⟶ 0) (at-right c)
    by (rule Lim-transform-eventually[OF ∗])
      (auto intro!: eventually-at-rightI[of c c + 1] simp: F-eq)
  thus ?thesis
    by (simp add: continuous-within Fc)
qed
have conta: continuous (at-right a) F
proof −
  have isCont (λx. exp (− ((1 + ξ * ((x − μ) / σ)) powr (− 1 / ξ)))) a
    using pos[OF ca] by (intro continuous-intros) auto
  hence continuous (at-right a) (λx. exp (− ((1 + ξ * ((x − μ) / σ)) powr (−
1 / ξ))))
    by (rule continuous-at-imp-continuous-at-within)
  hence ((λx. exp (− ((1 + ξ * ((x − μ) / σ)) powr (− 1 / ξ))))
        ⟶ exp (− ((1 + ξ * ((a − μ) / σ)) powr (− 1 / ξ)))) (at-right a)
    by (simp add: continuous-within)
  hence (F ⟶ exp (− ((1 + ξ * ((a − μ) / σ)) powr (− 1 / ξ)))) (at-right
a)
    by (rule Lim-transform-eventually)
      (auto intro!: eventually-at-rightI[of a a + 1] simp: F-eq less-trans[OF ca])
  thus ?thesis
    by (simp add: continuous-within F-eq[OF ca])
qed
have limtop: (F ⟶ 1) at-top
proof −
  have powr0: ((λx. (1 + ξ * ((x − μ) / σ)) powr (− 1 / ξ)) ⟶ 0) at-top
    by (rule tendsto-neg-powr[OF p-neg filterlim-shift-scale-at-top[OF sigma
True]])
  have ((λx. exp (− ((1 + ξ * ((x − μ) / σ)) powr (− 1 / ξ)))) ⟶ exp (−
0)) at-top
    by (intro tendsto-intros powr0)
  hence ∗: ((λx. exp (− ((1 + ξ * ((x − μ) / σ)) powr (− 1 / ξ)))) ⟶ 1)
at-top
    by simp
  have ev: ∀F x in at-top. exp (− ((1 + ξ * ((x − μ) / σ)) powr (− 1 / ξ)))
= F x
    using eventually-gt-at-top[of c] by eventually-elim (simp add: F-eq)
  show ?thesis
    by (rule Lim-transform-eventually[OF ∗ ev])
qed
have Fa: F a = exp (− ((1 + ξ * y) powr (− 1 / ξ)))
```

by (*simp add*: *F-eq*[*OF ca*] *y-def*)
have *Fa01*: $0 \leq F\ a\ F\ a \leq 1$
by (*auto simp*: *Fa*)
have *I1*: $(\int^+ x{\in}\{c{<}..\}.\ \textit{EVD-density}\ \xi\ \mu\ \sigma\ x\ \partial\textit{lborel}) = 1$
using *nn-integral-FTC-greaterThan*[*OF EVD-density-measurable contc deriv nonneg limtop*]
by (*simp add*: *Fc*)
have *I2*: $(\int^+ x{\in}\{a{<}..\}.\ \textit{EVD-density}\ \xi\ \mu\ \sigma\ x\ \partial\textit{lborel}) = 1 - \textit{ennreal}\ (F\ a)$
using *nn-integral-FTC-greaterThan*[*OF EVD-density-measurable conta deriv'' nonneg limtop*]
by (*metis ennreal-1 ennreal-minus Fa01*(*1*))
have $(\int^+ x{\in}\{c{<}..a\}.\ \textit{EVD-density}\ \xi\ \mu\ \sigma\ x\ \partial\textit{lborel}) + (\int^+ x{\in}\{a{<}..\}.\ \textit{EVD-density}\ \xi\ \mu\ \sigma\ x\ \partial\textit{lborel})$
$= (\int^+ x.\ \textit{ennreal}\ (\textit{EVD-density}\ \xi\ \mu\ \sigma\ x) * \textit{indicator}\ \{c{<}..a\}\ x$
$+\ \textit{ennreal}\ (\textit{EVD-density}\ \xi\ \mu\ \sigma\ x) * \textit{indicator}\ \{a{<}..\}\ x\ \partial\textit{lborel})$
by (*rule nn-integral-add*[*symmetric*]) *auto*
also have $\ldots = (\int^+ x{\in}\{c{<}..\}.\ \textit{EVD-density}\ \xi\ \mu\ \sigma\ x\ \partial\textit{lborel})$
using *ca* **by** (*auto intro*!: *nn-integral-cong simp*: *indicator-def*)
finally have *split*: $(\int^+ x{\in}\{c{<}..a\}.\ \textit{EVD-density}\ \xi\ \mu\ \sigma\ x\ \partial\textit{lborel}) + (1 - \textit{ennreal}\ (F\ a)) = 1$
by (*simp add*: *I1 I2*)
have *X*: $(\int^+ x{\in}\{c{<}..a\}.\ \textit{EVD-density}\ \xi\ \mu\ \sigma\ x\ \partial\textit{lborel}) = \textit{ennreal}\ (F\ a)$
proof −
have *le1*: $\textit{ennreal}\ (F\ a) \leq 1$
using *Fa01* **by** (*simp add*: *ennreal-leI*)
have *c-ne*: $1 - \textit{ennreal}\ (F\ a) \neq \top$
by *simp*
have $(\int^+ x{\in}\{c{<}..a\}.\ \textit{EVD-density}\ \xi\ \mu\ \sigma\ x\ \partial\textit{lborel})$
$= ((\int^+ x{\in}\{c{<}..a\}.\ \textit{EVD-density}\ \xi\ \mu\ \sigma\ x\ \partial\textit{lborel}) + (1 - \textit{ennreal}\ (F\ a)))$
$-\ (1 - \textit{ennreal}\ (F\ a))$
by (*simp add*: *c-ne*)
also have $\ldots = 1 - (1 - \textit{ennreal}\ (F\ a))$
by (*simp add*: *split*)
also have $\ldots = \textit{ennreal}\ (F\ a)$
using *le1 diff-diff-ennreal''* **by** *auto*
finally show *?thesis* **.**
qed
have $(\int^+ x{\in}\{..a\}.\ \textit{EVD-density}\ \xi\ \mu\ \sigma\ x\ \partial\textit{lborel}) = (\int^+ x{\in}\{c{<}..a\}.\ \textit{EVD-density}\ \xi\ \mu\ \sigma\ x\ \partial\textit{lborel})$
by (*auto intro*!: *nn-integral-cong simp*: *indicator-def dens0*)
thus *?thesis*
by (*simp add*: *X Fa*)
qed
have *1*: $(\int^+ x{\in}\{..a\}.\ \textit{ennreal}\ (\textit{EVD-density}\ \xi\ \mu\ \sigma\ x)\ \partial\textit{lborel})$
$= (\int^+ x.\ \textit{ennreal}\ |\textit{indicat-real}\ \{..a\}\ x * \textit{EVD-density}\ \xi\ \mu\ \sigma\ x|\ \partial\textit{lborel})$
by(*auto intro*!: *nn-integral-cong simp*: *indicator-def*)
show *set-integrable lborel* $\{..a\}$ (*EVD-density* $\xi\ \mu\ \sigma$)
using *nn* **by**(*simp add*: *set-integrable-def integrable-iff-bounded 1*)
have $(\int x{\in}\{..a\}.\ \textit{EVD-density}\ \xi\ \mu\ \sigma\ x\ \partial\textit{lborel})$

```
      = enn2real (∫⁺x∈{..a}. ennreal (EVD-density ξ μ σ x) ∂lborel)
    unfolding set-lebesgue-integral-def
    by(subst integral-eq-nn-integral)
     (auto simp: indicator-def intro!: arg-cong[where f=enn2real] nn-integral-cong)
  thus (∫ x∈{..a}. EVD-density ξ μ σ x ∂lborel) = exp (- ((1 + ξ * y) powr (-
1 / ξ)))
    by(simp add: nn)
qed

lemma
  fixes a μ σ
  defines y ≡ (a - μ) / σ
  assumes sigma[arith]:0 < σ
    and xiy:ξ * y ≤ - 1
    and xi:ξ > 0
  shows EVD-density-set-integrable2: set-integrable lborel {..a} (EVD-density ξ μ
σ)
    and EVD-density-set-integral2:
    (∫ x∈{..a}. EVD-density ξ μ σ x ∂lborel) = 0
    and EVD-density-set-nn-integral2:
    (∫⁺x∈{..a}. EVD-density ξ μ σ x ∂lborel) = 0
proof -
  have eq0[simp]:⋀x. x ≤ a ⟹ EVD-density ξ μ σ x = 0
    by (smt (verit, ccfv-SIG) EVD-density-def y-def sigma
        xi xiy divide-right-mono mult-le-cancel-left-pos)
  show set-integrable lborel {..a} (EVD-density ξ μ σ)
    by(subst set-integrable-cong[where f'=λx. 0,OF refl refl]) simp-all
  show (∫ x∈{..a}. EVD-density ξ μ σ x ∂lborel) = 0
    by(subst set-integral-cong[where g=λx. 0]) simp-all
  show (∫⁺x∈{..a}. EVD-density ξ μ σ x ∂lborel) = 0
    by(subst set-nn-integral-cong[where g=λx. 0]) simp-all
qed

lemma
  fixes a μ σ
  defines y ≡ (a - μ) / σ
  assumes sigma[arith]:0 < σ
    and xiy:ξ * y ≤ - 1
    and xi:ξ < 0
  shows EVD-density-set-integrable3: set-integrable lborel {..a} (EVD-density ξ μ
σ)
    and EVD-density-set-integral3:
    (∫ x∈{..a}. EVD-density ξ μ σ x ∂lborel) = 1
    and EVD-density-set-nn-integral3:
    (∫⁺x∈{..a}. EVD-density ξ μ σ x ∂lborel) = 1
proof -
  define b where b ≡ μ - σ / ξ
  have ba:b ≤ a
    using xiy xi unfolding b-def y-def by (simp add: field-simps)
```

have [*simp*]:$\bigwedge x.\ b \leq x \Longrightarrow$ *EVD-density* $\xi\ \mu\ \sigma\ x = 0$
by (*smt* (*verit*, *ccfv-SIG*) *EVD-density-def sigma xi b-def divide-le-eq le-divide-eq-1 minus-divide-left mult.commute times-divide-eq-right*)
have *b*:$\xi * ((b - \mu) / \sigma) \geq -1$
using *xi* **by**(*simp add*: *b-def*)
have *b'*:$0 < \xi \Longrightarrow -1 < \xi * ((b - \mu) / \sigma)$
using *xi* **by** *linarith*
have *h1*[*simp*]:*set-integrable lborel* $\{..b\}$ (*EVD-density* $\xi\ \mu\ \sigma$)
and *h2*:$(\int x{\in}\{..b\}.$ *EVD-density* $\xi\ \mu\ \sigma\ x\ \partial$*lborel*$) = 1$
and *h3*:$(\int^+ x{\in}\{..b\}.$ *EVD-density* $\xi\ \mu\ \sigma\ x\ \partial$*lborel*$) = 1$
using *EVD-density-set-integrable1*[*OF sigma less-imp-neq*[*OF xi*] *b b'*]
EVD-density-set-integral1[*OF sigma less-imp-neq*[*OF xi*] *b b'*]
EVD-density-set-nn-integral1[*OF sigma less-imp-neq*[*OF xi*] *b b'*] *xi*
by(*simp-all add*: *b-def*)
have [*simp*]:*set-integrable lborel* $\{b<..a\}$ (*EVD-density* $\xi\ \mu\ \sigma$)
by(*subst set-integrable-cong*[**where** $f'{=}\lambda x.\ 0$,*OF refl refl*]) *simp-all*
have *eq0*:$(\int x{\in}\{b<..a\}.$ *EVD-density* $\xi\ \mu\ \sigma\ x\ \partial$*lborel*$) = 0$
by(*subst set-integral-cong*[**where** $g{=}\lambda x.\ 0$]) *auto*
have *eq0'*:$(\int^+ x{\in}\{b<..a\}.$ *EVD-density* $\xi\ \mu\ \sigma\ x\ \partial$*lborel*$) = 0$
by(*subst set-nn-integral-cong*[**where** $g{=}\lambda x.\ 0$,*OF refl refl*]) *auto*
show *set-integrable lborel* $\{..a\}$ (*EVD-density* $\xi\ \mu\ \sigma$)
using *ba*
by(*subst set-integrable-cong*[**where** $A'{=}\{..b\} \cup \{b<..a\}$])
(*auto intro*!: *set-integrable-Un h1*)
have $(\int x{\in}\{..a\}.$ *EVD-density* $\xi\ \mu\ \sigma\ x\ \partial$*lborel*$) =$
$(\int x{\in}\{..b\}.$ *EVD-density* $\xi\ \mu\ \sigma\ x\ \partial$*lborel*$) + (\int x{\in}\{b<..a\}.$ *EVD-density* $\xi\ \mu\ \sigma\ x\ \partial$*lborel*$)$
using *ba*
by(*subst set-integral-Un*[*symmetric*])
(*auto intro*!: *arg-cong*[**where** $f{=}\lambda B.$ *set-lebesgue-integral lborel B* -])
also have $\ldots = 1$
by (*auto simp*: *h2 eq0*)
finally show $(\int x{\in}\{..a\}.$ *EVD-density* $\xi\ \mu\ \sigma\ x\ \partial$*lborel*$) = 1$ **.**
have $(\int^+ x{\in}\{..a\}.$ *EVD-density* $\xi\ \mu\ \sigma\ x\ \partial$*lborel*$) =$
$(\int^+ x.$ *ennreal* (*EVD-density* $\xi\ \mu\ \sigma\ x$) $*$ *indicator* $\{..b\}\ x$
$+$ *ennreal* (*EVD-density* $\xi\ \mu\ \sigma\ x$) $*$ *indicator* $\{b<..a\}\ x\ \partial$*lborel*$)$
using *ba* **by**(*auto intro*!: *nn-integral-cong simp*: *indicator-def*)
also have $\ldots = (\int^+ x{\in}\{..b\}.$ *EVD-density* $\xi\ \mu\ \sigma\ x\ \partial$*lborel*$)$
using *ba* **by**(*auto intro*!: *nn-integral-cong simp*: *indicator-def*)
finally show $(\int^+ x{\in}\{..a\}.$ *EVD-density* $\xi\ \mu\ \sigma\ x\ \partial$*lborel*$) = 1$
by(*simp add*: *h3*)
qed

lemma *EVD-density-set-integrable*:
assumes *sigma*[*arith*]:$0 < \sigma$
shows *set-integrable lborel* $\{..a\}$ (*EVD-density* $\xi\ \mu\ \sigma$)
proof −
consider $\xi = 0 \mid \xi \neq 0 - 1 < \xi * ((a - \mu) / \sigma)$

```
    | ξ * ((a - μ) / σ) ≤ -1 ξ > 0 | ξ * ((a - μ) / σ) ≤ -1 ξ < 0
    by argo
  thus ?thesis
    by cases
    (use EVD-density-set-integrable0[OF sigma]
      EVD-density-set-integrable1[OF sigma - less-imp-le]
      EVD-density-set-integrable2[OF sigma]
      EVD-density-set-integrable3[OF sigma] in auto)
qed

abbreviation EVD-cdf where EVD-cdf ≡ λξ μ σ x. cdf (density lborel (EVD-density
ξ μ σ)) x

lemma EVD-cdf-set-integral:
  assumes sigma[arith]:0 < σ
  shows EVD-cdf ξ μ σ a = (∫ x∈{..a}. EVD-density ξ μ σ x ∂lborel)
proof -
  consider ξ = 0 | ξ ≠ 0 - 1 < ξ * ((a - μ) / σ)
    | ξ > 0 ξ * ((a - μ) / σ) ≤ - 1
    | ξ < 0 ξ * ((a - μ) / σ) ≤ - 1
    by linarith
  thus ?thesis
    unfolding cdf-def
    using EVD-density-set-integrable3[OF sigma]
      EVD-density-set-integrable2[OF sigma]
      EVD-density-set-integrable1[OF sigma]
      EVD-density-set-integrable0[OF sigma]
    by (cases; subst measure-density-on) auto
qed

corollary EVD-cdf-def:
  assumes sigma[arith]:0 < σ
  shows EVD-cdf ξ μ σ x =
                  (let y = (x - μ) / σ in
                   if ξ = 0 then exp (- exp (- y))
                   else if ξ * y > - 1 then exp (- ((1 + ξ * y) powr (- 1 / ξ)))
                   else if ξ > 0 then 0 else 1)
proof -
  note cdf-simps = cdf-def measure-def emeasure-density
  consider ξ = 0 | ξ ≠ 0 - 1 < ξ * ((x - μ) / σ)
    | ξ * ((x - μ) / σ) ≤ -1 ξ > 0 | ξ * ((x - μ) / σ) ≤ -1 ξ < 0
    by argo
  thus ?thesis
  proof cases
    assume 2:ξ ≠ 0 - 1 < ξ * ((x - μ) / σ)
    thus ?thesis
      by(simp add: cdf-simps
          EVD-density-set-nn-integral1[OF sigma 2(1) less-imp-le[OF 2(2)] 2(2)])
  qed(simp-all add: cdf-simps minus-divide-left
```

```
    EVD-density-set-nn-integral0
    EVD-density-set-nn-integral2
    EVD-density-set-nn-integral3)
qed

lemma EVD-cdf-supp-bounds:
  fixes x γ :: real
  assumes [arith]:1 + γ * (x − μ) / σ > 0 0 < σ
  shows 0 < EVD-cdf γ μ σ x EVD-cdf γ μ σ x ≤ 1
  using assms by (auto simp add: EVD-cdf-def Let-def)

lemma EVD-cdf-supp-bounds01:
  fixes x γ :: real
  assumes [arith]:1 + γ * x > 0
  shows 0 < EVD-cdf γ 0 1 x EVD-cdf γ 0 1 x ≤ 1
  by(auto intro!: EVD-cdf-supp-bounds)

lemma ln-EVD-cdf-0:
  assumes 0 < σ
  shows − ln (EVD-cdf 0 μ σ x) = exp (− ((x − μ) / σ))
  using assms by(simp add: EVD-cdf-def[of σ])

lemma ln-EVD-cdf-1:
  assumes 0 < σ 1 + γ * (x − μ) / σ > 0 γ ≠ 0
  shows − ln (EVD-cdf γ μ σ x) = (1 + γ * (x − μ) / σ) powr (− 1 / γ)
  using assms by(auto simp add: EVD-cdf-def[of σ])

lemma EVD-cdf-location-scale:
  assumes [arith]:0 < σ
  shows EVD-cdf ξ 0 1 ((x − μ) / σ) = EVD-cdf ξ μ σ x
  by(simp add: EVD-cdf-def)

lemma EVD-cdf-at-bot-0:
  assumes sigma:0 < σ
  shows (EVD-cdf ξ μ σ ⟶ 0) at-bot
proof −
  consider ξ = 0 | ξ > 0 | ξ < 0
    by linarith
  thus ?thesis
  proof cases
    assume xi:ξ = 0
    have ∗: ((λx. exp (− exp (− ((x − μ) / σ)))) ⟶ 0) at-bot
      using sigma
      by(intro filterlim-compose[OF exp-at-bot
                filterlim-compose[OF filterlim-uminus-at-bot-at-top]]
              filterlim-compose[OF exp-at-top])
        real-asymp
    show ?thesis
      by(rule Lim-transform-eventually[OF ∗])
```

```
      (use xi in ‹simp add: EVD-cdf-def[OF sigma] Let-def›)
  next
    assume xi:ξ > 0
    have ev: ∀F x in at-bot. EVD-cdf ξ μ σ x = 0
      using xi sigma
      by (auto intro!: eventually-mono[OF eventually-le-at-bot[of μ − σ / ξ]]
          simp: EVD-cdf-def[OF sigma] Let-def field-simps)
    thus ?thesis
      by (rule tendsto-eventually)
  next
    assume xi:ξ < 0
    have p-pos: 0 < − 1 / ξ
      using xi by (simp add: divide-pos-neg)
    note base = filterlim-shift-scale-at-bot[OF sigma xi]
     have ∗: ((λx. exp (− ((1 + ξ ∗ ((x − μ) / σ)) powr (− 1 / ξ)))) ⟶ 0)
at-bot
      by (rule filterlim-compose[OF exp-at-bot
            filterlim-compose[OF filterlim-uminus-at-bot-at-top
              filterlim-compose[OF real-powr-at-top[OF p-pos] base]]])
    have evlt: ∀F x in at-bot. x < μ − σ / ξ
      using eventually-le-at-bot[of μ − σ / ξ − 1] by eventually-elim simp
    have ev:
     ∀F x in at-bot. exp (− ((1 + ξ ∗ ((x − μ) / σ)) powr (− 1 / ξ))) = EVD-cdf
ξ μ σ x
      using evlt
    proof eventually-elim
      case (elim x)
      have (x − μ) / σ < ((μ − σ / ξ) − μ) / σ
        using elim sigma by (intro divide-strict-right-mono) auto
      hence ξ ∗ (((μ − σ / ξ) − μ) / σ) < ξ ∗ ((x − μ) / σ)
        by (rule mult-strict-left-mono-neg[OF - xi])
      moreover have ξ ∗ (((μ − σ / ξ) − μ) / σ) = − 1
        using xi sigma by (simp add: field-simps)
      ultimately have − 1 < ξ ∗ ((x − μ) / σ)
        by simp
      thus ?case
        using xi by (simp add: EVD-cdf-def[OF sigma] Let-def)
    qed
    show ?thesis
      by (rule Lim-transform-eventually[OF ∗ ev])
  qed
qed

lemma EVD-cdf-at-top-1:
  assumes sigma:0 < σ
  shows (EVD-cdf ξ μ σ ⟶ 1) at-top
proof −
  consider ξ = 0 | ξ > 0 | ξ < 0
    by linarith
```

```
  thus ?thesis
  proof cases
    assume xi:ξ = 0
    have 1: ((λx. exp (− ((x − μ) / σ))) ⟶ 0) at-top
      using sigma
      by (intro filterlim-compose[OF exp-at-bot]) real-asymp
    have ((λx. exp (− exp (− ((x − μ) / σ)))) ⟶ exp (− 0)) at-top
      by (intro tendsto-intros 1)
    hence ∗: ((λx. exp (− exp (− ((x − μ) / σ)))) ⟶ 1) at-top
      by simp
    show ?thesis
      by(rule Lim-transform-eventually[OF ∗])
        (use xi in ‹simp add: EVD-cdf-def[OF sigma] Let-def›)
  next
    assume xi:ξ > 0
    have p-neg: − 1 / ξ < 0
      using xi by (simp add: divide-neg-pos)
    have powr0: ((λx. (1 + ξ ∗ ((x − μ) / σ)) powr (− 1 / ξ)) ⟶ 0) at-top
      by (rule tendsto-neg-powr[OF p-neg filterlim-shift-scale-at-top[OF sigma xi]])
    have ((λx. exp (− ((1 + ξ ∗ ((x − μ) / σ)) powr (− 1 / ξ)))) ⟶ exp (−
0)) at-top
      by (intro tendsto-intros powr0)
    hence ∗: ((λx. exp (− ((1 + ξ ∗ ((x − μ) / σ)) powr (− 1 / ξ)))) ⟶ 1)
at-top
      by simp
    have ev:
     ∀F x in at-top. exp (− ((1 + ξ ∗ ((x − μ) / σ)) powr (− 1 / ξ))) = EVD-cdf
ξ μ σ x
      using eventually-gt-at-top[of μ − σ / ξ]
    proof eventually-elim
      case (elim x)
      have ((μ − σ / ξ) − μ) / σ < (x − μ) / σ
        using elim sigma by (intro divide-strict-right-mono) auto
      hence ξ ∗ (((μ − σ / ξ) − μ) / σ) < ξ ∗ ((x − μ) / σ)
        by (rule mult-strict-left-mono[OF - xi])
      moreover have ξ ∗ (((μ − σ / ξ) − μ) / σ) = − 1
        using xi sigma by (simp add: field-simps)
      ultimately have − 1 < ξ ∗ ((x − μ) / σ)
        by simp
      thus ?case
        using xi by (simp add: EVD-cdf-def[OF sigma] Let-def)
    qed
    show ?thesis
      by (rule Lim-transform-eventually[OF ∗ ev])
  next
    assume xi:ξ < 0
    have ev: ∀F x in at-top. EVD-cdf ξ μ σ x = 1
      using eventually-ge-at-top[of μ − σ / ξ]
    proof eventually-elim
```

```
    case (elim x)
    have ((μ − σ / ξ) − μ) / σ ≤ (x − μ) / σ
      using elim sigma by (intro divide-right-mono) auto
    hence ξ ∗ ((x − μ) / σ) ≤ ξ ∗ (((μ − σ / ξ) − μ) / σ)
      by (rule mult-left-mono-neg[OF - less-imp-le[OF xi]])
    moreover have ξ ∗ (((μ − σ / ξ) − μ) / σ) = − 1
      using xi sigma by (simp add: field-simps)
    ultimately have ξ ∗ ((x − μ) / σ) ≤ − 1
      by simp
    thus ?case
      using xi by (simp add: EVD-cdf-def[OF sigma] Let-def)
  qed
  thus ?thesis
    by (rule tendsto-eventually)
 qed
qed

lemma
 assumes sigma[arith]:0 < σ
 shows EVD-density-integrable:integrable lborel (EVD-density ξ μ σ)
  and EVD-density-integral:(∫ x. EVD-density ξ μ σ x ∂lborel) = 1
  and EVD-density-nn-integral:(∫⁺x. EVD-density ξ μ σ x ∂lborel) = 1
proof −
 have lim:((λa. (∫ x∈{..a}. EVD-density ξ μ σ x ∂lborel)) ⟶ 1) at-top
 proof −
  have ∗:(∫ x∈{..a}. EVD-density ξ μ σ x ∂lborel) = EVD-cdf ξ μ σ a for a
    unfolding cdf-def
    by(subst measure-density-on)
      (auto simp: EVD-density-set-integrable)
  show ?thesis
    unfolding ∗ by(rule EVD-cdf-at-top-1[OF sigma])
 qed
 show integ:integrable lborel (EVD-density ξ μ σ)
  and eq2:(∫ x. EVD-density ξ μ σ x ∂lborel) = 1
  using lim EVD-density-set-integrable[OF sigma]
      integrable-monotone-convergence-at-top[where f=EVD-density ξ μ σ and
x=1,
       OF - AE-I2[OF EVD-density-nonneg[OF less-imp-le[OF sigma]]]]
    integral-monotone-convergence-at-top[where f=EVD-density ξ μ σ and x=1,
      OF - AE-I2[OF EVD-density-nonneg[OF less-imp-le[OF sigma]]]]
  by(auto simp: set-integrable-def set-lebesgue-integral-def mult.commute)
 thus (∫⁺x. EVD-density ξ μ σ x ∂lborel) = 1
  by(subst nn-integral-eq-integral) auto
qed

lemma prob-space-EVD:
 assumes sigma[arith]:0 < σ
 shows prob-space (density lborel (EVD-density ξ μ σ))
 by(auto intro!: prob-spaceI simp: emeasure-density EVD-density-nn-integral)
```

corollary *real-distribution-EVD*:
0 < σ ⟹ real-distribution (density lborel (EVD-density ξ μ σ))
by(*simp add*: *real-distribution-def real-distribution-axioms-def prob-space-EVD*)

corollary *right-continuous-mono-EVD-cdf*:
0 < σ ⟹ right-continuous-mono (EVD-cdf ξ μ σ) 0 1
by(*auto intro*!: *right-continuous-mono-cdf real-distribution-EVD*)

corollary *EVD-cdf-nonneg*:*0 < σ ⟹ 0 ≤ EVD-cdf ξ μ σ x*
and *EVD-cdf-le-1*:*0 < σ ⟹ EVD-cdf ξ μ σ x ≤ 1*
using *right-continuous-mono-EVD-cdf right-continuous-mono.a-le-f*[*of - 0 1*]
right-continuous-mono.f-le-b[*of - 0 1*]
by *fastforce*+

lemma *isCont-EVD-cdf*:
assumes *sigma*[*arith*]: *0 < σ*
shows *isCont (EVD-cdf ξ μ σ) x*
using *continuous-on-LBINT-com*[*OF EVD-density-integrable*[*OF sigma*]]
EVD-cdf-set-integral[*OF sigma*]
by *simp*

corollary *weak-conv-EVD-cdf-iff*:
assumes *sigma*:*0 < σ*
shows *weak-conv F (EVD-cdf ξ μ σ) ⟷ (∀ x. (λn. F n x) ⟶ EVD-cdf ξ μ σ x)*
using *isCont-EVD-cdf*[*OF sigma*] **by**(*simp add*: *weak-conv-def*)

Max-stability of the GEV distribution (Exercise 1.2 of the book [2]). The book's argument is $G_\xi^n(a_n x + b_n) = G_\xi(x)$ for $a_n = n^\xi$ and $b_n = (n^\xi - 1)/\xi$, whereas we state the equivalent solved form $G_\xi^n(x) = G_\xi((x - b_n)/a_n)$, in which $(x - b_n)/a_n = n^{-\xi}x + (n^{-\xi} - 1)/\xi$.

lemma *EVD-cdf-max-stable*:
assumes *0 < n*
shows *EVD-cdf ξ 0 1 x ^ n*
= EVD-cdf ξ 0 1 (if ξ = 0 then x − ln n
*else n powr (− ξ) * x + (real n powr (− ξ) − 1) / ξ)*
proof −
have [*arith*]: (*0*::*real*) < *1*
by *simp*
consider *ξ = 0 | ξ ≠ 0 − 1 < ξ * x | ξ ≠ 0 ξ * x ≤ − 1* **by** *argo*
thus *?thesis*
proof *cases*
case *1*
have *EVD-cdf ξ 0 1 x ^ n = exp (real n * (− exp (− x)))*
by (*simp add*: *EVD-cdf-def Let-def 1 exp-of-nat-mult*[*symmetric*])
also have *real n * (− exp (− x)) = − exp (− (x − ln (real n)))*
using *assms* **by** (*simp add*: *exp-diff exp-minus field-simps*)
finally show *?thesis*

```
    using 1 by (simp add: EVD-cdf-def Let-def)
  next
    case 2
    define c where c = real n powr (− ξ)
    have c-pos: 0 < c
      using assms by (simp add: c-def)
    have arg: 1 + ξ ∗ (c ∗ x + (c − 1) / ξ) = c ∗ (1 + ξ ∗ x)
      using 2(1) by (simp add: field-simps)
    have p1: 0 < c ∗ (1 + ξ ∗ x)
      using c-pos 2(2) by (intro mult-pos-pos) auto
    have argpos: − 1 < ξ ∗ (c ∗ x + (c − 1) / ξ)
      using arg p1 by linarith
    have (c ∗ (1 + ξ ∗ x)) powr (− 1 / ξ) = c powr (− 1 / ξ) ∗ (1 + ξ ∗ x) powr
(− 1 / ξ)
      by (rule powr-mult)
    also have c powr (− 1 / ξ) = real n powr ((− ξ) ∗ (− 1 / ξ))
      by (simp add: c-def powr-powr)
    also have (− ξ) ∗ (− 1 / ξ) = 1
      using 2(1) by simp
    also have real n powr 1 = real n
      by simp
    finally have ∗: (c ∗ (1 + ξ ∗ x)) powr (− 1 / ξ) = real n ∗ (1 + ξ ∗ x) powr
(− 1 / ξ) .
    have EVD-cdf ξ 0 1 x ^ n = exp (real n ∗ (− ((1 + ξ ∗ x) powr (− 1 / ξ))))
      using 2 by (simp add: EVD-cdf-def Let-def exp-of-nat-mult[symmetric])
    also have ... = exp (− ((c ∗ (1 + ξ ∗ x)) powr (− 1 / ξ)))
      using ∗ by simp
    also have ... = EVD-cdf ξ 0 1 (c ∗ x + (c − 1) / ξ)
      using 2(1) argpos by (simp add: EVD-cdf-def Let-def arg)
    finally show ?thesis
      using 2(1) by (simp add: c-def)
  next
    case 3
    define c where c = real n powr (− ξ)
    have c-pos: 0 < c
      using assms by (simp add: c-def)
    have le0: c ∗ (1 + ξ ∗ x) ≤ 0
      using 3(2) by (intro mult-nonneg-nonpos) (use c-pos in auto)
    have arg: ξ ∗ (c ∗ x + (c − 1) / ξ) = c ∗ (1 + ξ ∗ x) − 1
      using 3(1) by (simp add: field-simps)
    have argneg: ξ ∗ (c ∗ x + (c − 1) / ξ) ≤ − 1
      using le0 by (simp add: arg)
    consider 0 < ξ | ξ < 0 using 3(1) by argo
    thus ?thesis
    proof cases
      case pos: 1
      hence EVD-cdf ξ 0 1 x = 0
        using 3 by (simp add: EVD-cdf-def Let-def)
      moreover have EVD-cdf ξ 0 1 (c ∗ x + (c − 1) / ξ) = 0
```

```
      using pos argneg by (simp add: EVD-cdf-def Let-def not-less)
    ultimately show ?thesis
      using assms 3(1) by (simp add: c-def)
  next
    case neg: 2
    hence EVD-cdf ξ 0 1 x = 1
      using 3 by (simp add: EVD-cdf-def Let-def)
    moreover have EVD-cdf ξ 0 1 (c * x + (c - 1) / ξ) = 1
      using neg argneg by (simp add: EVD-cdf-def Let-def not-less)
    ultimately show ?thesis
      using 3(1) by (simp add: c-def)
  qed
 qed
qed

end
```

2.2 Generalized Pareto Distribution

```
theory Pareto-Distribution
imports
  Lemmas-EVT
  Extreme-Value-Distribution
begin
```

We define the generalized Pareto distribution (GPD) density and cdf.

```
definition GPD-density :: [real, real, real, real] ⇒ real where
GPD-density ξ μ σ x ≡ if ξ = 0 then
                       if μ ≤ x then exp (- (x - μ) / σ) / σ else 0
                     else if ξ > 0 then
                       if μ ≤ x then ((1 + ξ * (x - μ) / σ) powr (- 1 - 1 / ξ)) / σ
                         else 0
                     else
                       if μ ≤ x ∧ x ≤ μ - σ / ξ then
                        ((1 + ξ * (x - μ) / σ) powr (- 1 - 1 / ξ)) / σ
                       else 0

lemma GPD-density-nonneg[simp]:0 ≤ σ ⟹ 0 ≤ GPD-density ξ μ σ x
  by(simp add: GPD-density-def)

lemma GPD-density-measurable[measurable]:
  GPD-density ξ μ σ ∈ borel-measurable borel
  unfolding GPD-density-def by measurable

lemma GPD-density-eq-exponential-density:
  GPD-density 0 μ σ = (λx. exponential-density (1 / σ) (x - μ))
  by(auto simp: exponential-density-def GPD-density-def)

lemma GPD-density-xi-pos:
```

$0 < \xi \Longrightarrow \mu \leq x$
$\Longrightarrow$ *GPD-density* ξ μ σ x = $(1 + \xi * (x - \mu) / \sigma)$ *powr* $(- 1 - 1 / \xi) / \sigma$
by(*simp add*: *GPD-density-def*)

lemma *GPD-density-xi-neg*:
$\xi < 0 \Longrightarrow \mu \leq x \Longrightarrow x \leq \mu - \sigma / \xi$
$\Longrightarrow$ *GPD-density* ξ μ σ x = $(1 + \xi * (x - \mu) / \sigma)$ *powr* $(- 1 - 1 / \xi) / \sigma$
by(*simp add*: *GPD-density-def*)

lemma *GPD-density-eq-0*:
$x < \mu \Longrightarrow$ *GPD-density* ξ μ σ $x = 0$
by(*simp add*: *GPD-density-def*)

lemma *GPD-density-xi-neg-eq-0*:
$\xi < 0 \Longrightarrow \mu - \sigma / \xi < x \Longrightarrow$ *GPD-density* ξ μ σ $x = 0$
by(*simp add*: *GPD-density-def*)

lemma
fixes a μ σ :: *real*
assumes *sigma*[*arith*]:$0 < \sigma$
shows *GPD-density-set-integrable0*:*set-integrable lborel* $\{..a\}$ (*GPD-density 0* μ σ)
and *GPD-density-set-integral0*:
$(\int x{\in}\{..a\}.$ *GPD-density 0* μ σ x ∂*lborel*) = (*if* $a < \mu$ *then 0 else 1* $-$ *exp* $(-$ $(a - \mu) / \sigma))$
and *GPD-density-set-nn-integral0*:
$(\int^+ x{\in}\{..a\}.$ *GPD-density 0* μ σ x ∂*lborel*)
= *ennreal* (*if* $a < \mu$ *then 0 else 1* $-$ *exp* $(-$ $(a - \mu) / \sigma))$
proof $-$
have $*$: *indicator* $\{..a\}$ $(\mu + x)$ = *indicator* $\{..a{-}\mu\}$ x **for** x
by(*auto simp*: *indicator-def*)
have $(\int^+ x{\in}\{..a\}.$ *GPD-density 0* μ σ x ∂*lborel*)
= $(\int^+ x{\in}\{..a - \mu\}.$ *exponential-density* $(1 / \sigma)$ x ∂*lborel*)
by(*subst nn-integral-real-affine*
[**where** $t{=}\mu$ **and** $c{=}1$ **and** $f{=}\lambda x.$ *ennreal* (*GPD-density 0* μ σ x) $*$ *indicator* $\{..a\}$ x])
(*auto simp add*: *GPD-density-eq-exponential-density* $*$)
also have ... = *ennreal* (*if* $a < \mu$ *then 0 else 1* $-$ *exp* $(-$ $(a - \mu) / \sigma))$
by(*simp add*: *nn-integral-erlang-density erlang-CDF-def minus-divide-left*
del: *minus-divide-left*[*symmetric*])
finally show *nn-eq*: $(\int^+ x{\in}\{..a\}.$ *GPD-density 0* μ σ x ∂*lborel*)
= *ennreal* (*if* $a < \mu$ *then 0 else 1* $-$ *exp* $(-$ $(a - \mu) / \sigma))$.

show *integ0*: *set-integrable lborel* $\{..a\}$ (*GPD-density 0* μ σ)
unfolding *set-integrable-def*
using *nn-eq*
by(*auto intro*!: *integrableI-nn-integral-finite*
[**where** x=*if* $a < \mu$ *then 0 else 1* $-$ *exp* $(-$ $(a - \mu) / \sigma)$]
simp: *ennreal-indicator ennreal-mult' mult.commute*)

```
  have nonneg: 0 ≤ (∫ x∈{..a}. GPD-density 0 μ σ x ∂lborel)
    unfolding set-lebesgue-integral-def
    by (rule Bochner-Integration.integral-nonneg) (auto simp: indicator-def)
  have ennreal (∫ x∈{..a}. GPD-density 0 μ σ x ∂lborel)
         = (∫⁺x. ennreal (indicat-real {..a} x *R GPD-density 0 μ σ x) ∂lborel)
    unfolding set-lebesgue-integral-def
    by (rule nn-integral-eq-integral[symmetric])
      (use integ0[unfolded set-integrable-def] in ‹auto simp: indicator-def›)
  also have ... = (∫⁺x∈{..a}. GPD-density 0 μ σ x ∂lborel)
    by (simp add: ennreal-indicator ennreal-mult' mult.commute)
  also have ... = ennreal (if a < μ then 0 else 1 − exp (− (a − μ) / σ))
    using nn-eq .
  finally have eq: ennreal (∫ x∈{..a}. GPD-density 0 μ σ x ∂lborel)
                    = ennreal (if a < μ then 0 else 1 − exp (− (a − μ) / σ)) .
  show (∫ x∈{..a}. GPD-density 0 μ σ x ∂lborel)
       = (if a < μ then 0 else 1 − exp (− (a − μ) / σ))
    by(intro ennreal-inj[THEN iffD1,OF - - eq] nonneg)
      (auto simp: divide-nonpos-pos)
qed

lemma
  fixes a μ σ :: real
  defines y ≡ (a − μ) / σ
  assumes sigma[arith]:0 < σ
    and amu:a ≤ μ
  shows GPD-density-set-integrable2:set-integrable lborel {..a} (GPD-density ξ μ
σ)
    and GPD-density-set-integral2:(∫ x∈{..a}. GPD-density ξ μ σ x ∂lborel) = 0
    and GPD-density-set-nn-integral2:(∫⁺x∈{..a}. GPD-density ξ μ σ x ∂lborel)
= 0
proof −
  have ae: AE x in lborel. x ∈ {..a} ⟶ GPD-density ξ μ σ x = 0
    using amu by(intro AE-I[where N={μ}]) (auto intro!: GPD-density-eq-0)
  show set-integrable lborel {..a} (GPD-density ξ μ σ)
    unfolding set-integrable-def
    by(rule integrable-cong-AE[where g=λx. 0, THEN iffD2])
      (use ae in ‹auto elim!: eventually-mono simp: indicator-def›)
  show (∫ x∈{..a}. GPD-density ξ μ σ x ∂lborel) = 0
    unfolding set-lebesgue-integral-def
    by(subst integral-cong-AE[where g=λx. 0])
      (use ae in ‹auto elim!: eventually-mono simp: indicator-def›)
  show (∫⁺x∈{..a}. GPD-density ξ μ σ x ∂lborel) = 0
    by(intro nn-integral-zero')
      (use ae in ‹auto elim!: eventually-mono simp: indicator-def›)
qed

lemma
  fixes a μ σ :: real
  defines y ≡ (a − μ) / σ
```

```
  assumes sigma[arith]:0 < σ
    and xi:ξ ≠ 0
    and mua:μ ≤ a
    and xiy:- 1 ≤ ξ * y
  shows GPD-density-set-integrable1:set-integrable lborel {..a} (GPD-density ξ μ
σ)
    and GPD-density-set-integral1:
    (∫ x∈{..a}. GPD-density ξ μ σ x ∂lborel) = 1 - (1 + ξ * y) powr (- 1 / ξ)
    and GPD-density-set-nn-integral1:
    (∫⁺x∈{..a}. GPD-density ξ μ σ x ∂lborel) = ennreal (1 - (1 + ξ * y) powr
(- 1 / ξ))
proof -
  have main: set-integrable lborel {..a} (GPD-density ξ μ σ) ∧
             (∫ x∈{..a}. GPD-density ξ μ σ x ∂lborel) = 1 - (1 + ξ * y) powr (-
1 / ξ) ∧
             (∫⁺x∈{..a}. GPD-density ξ μ σ x ∂lborel)
              = ennreal (1 - (1 + ξ * y) powr (- 1 / ξ))
  proof(cases μ < a)
    assume ¬ μ < a
    with mua
    have y0: y = 0 and amu:a ≤ μ
      by (simp-all add: y-def)
   from GPD-density-set-integrable2[OF sigma amu] GPD-density-set-integral2[OF
sigma amu]
      GPD-density-set-nn-integral2[OF sigma amu]
    show ?thesis
      by (simp add: y0)
  next
    assume mua': μ < a
    define g where g ≡ λt. (1 + ξ * (t - μ) / σ) powr (- 1 - 1 / ξ) / σ
    define F where F ≡ λt. - ((1 + ξ * (t - μ) / σ) powr (- 1 / ξ))
    have g-nonneg: 0 ≤ g t for t
      by (simp add: g-def)
    have g-meas[measurable]: g ∈ borel-measurable borel
      unfolding g-def by measurable
    have Fmu: F μ = - 1
      by (simp add: F-def)
    have Fa: F a = - ((1 + ξ * y) powr (- 1 / ξ))
      by (simp add: F-def y-def)
    have pos: 0 < 1 + ξ * (t - μ) / σ if t: μ < t t < a for t
    proof(cases 0 < ξ)
      case True
      have 0 < ξ * (t - μ) / σ
        using t True by (intro divide-pos-pos mult-pos-pos) auto
      thus ?thesis
        by linarith
    next
      case False
      with xi have xi-neg: ξ < 0
```

```
    by simp
  have - σ ≤ ξ * (a - μ)
    using xiy sigma unfolding y-def by (simp add: pos-le-divide-eq)
  moreover have ξ * (a - μ) < ξ * (t - μ)
    using t xi-neg by (intro mult-strict-left-mono-neg) auto
  ultimately have - σ < ξ * (t - μ)
    by linarith
  hence - 1 < ξ * (t - μ) / σ
    using sigma by (simp add: pos-less-divide-eq)
  thus ?thesis
    by linarith
qed
have deriv: (F has-real-derivative g t) (at t) if t: μ < t t < a for t
  using pos[OF t] xi sigma unfolding F-def g-def
  by(auto intro!: derivative-eq-intros simp: field-simps) argo
have isCont-g: isCont g t if t: μ < t t < a for t
  using pos[OF t] sigma unfolding g-def
  by(intro continuous-intros) auto
have A: (F ⟶ - 1) (at-right μ)
proof -
  have isCont F μ
    unfolding F-def by(intro continuous-intros) auto
  from continuous-at-imp-continuous-at-within[OF this]
  show ?thesis
    by (simp add: continuous-within Fmu)
qed
have B: (F ⟶ F a) (at-left a)
proof(cases ξ * y = - 1)
  case False
  with xiy have - 1 < ξ * ((a - μ) / σ)
    unfolding y-def by (auto simp: le-less)
  hence pos-a: 0 < 1 + ξ * (a - μ) / σ
    by simp
  have isCont F a
    unfolding F-def using pos-a by(intro continuous-intros) auto
  from continuous-at-imp-continuous-at-within[OF this]
  show ?thesis
    by (simp add: continuous-within)
next
  case True
  have xi-neg: ξ < 0
  proof(rule ccontr)
    assume ¬ ξ < 0
    with xi have 0 < ξ
      by simp
    hence 0 ≤ ξ * y
      using mua by (auto intro: mult-nonneg-nonneg simp: y-def)
    with True show False
      by simp
```

```
    qed
    have p-pos: 0 < - 1 / ξ
      using xi-neg by (simp add: divide-pos-neg)
    have base-a: 1 + ξ * (a - μ) / σ = 0
      using True by (simp add: y-def)
    have Fa0: F a = 0
      using xi by (simp add: F-def base-a)
    have base0: ((λt. 1 + ξ * (t - μ) / σ) ⟶ 0) (at-left a)
      unfolding base-a[symmetric]
      by (intro tendsto-intros) auto
    have ev-nonneg: ∀⇩F t in at-left a. 0 ≤ 1 + ξ * (t - μ) / σ
     by (intro eventually-at-leftI[of μ]) (use pos[THEN less-imp-le] mua' in auto)
    have ((λt. (1 + ξ * (t - μ) / σ) powr (- 1 / ξ)) ⟶ 0) (at-left a)
      by (rule tendsto-zero-powrI[OF base0 tendsto-const ev-nonneg p-pos])
    hence (F ⟶ - 0) (at-left a)
      unfolding F-def by (rule tendsto-minus)
    thus ?thesis
      by (simp add: Fa0)
  qed
  have FTC1: set-integrable lborel (einterval (ereal μ) (ereal a)) g
    by (rule interval-integral-FTC-nonneg[where F=F and A=- 1 and B=F
a])
       (use mua' deriv isCont-g g-nonneg A B in
        ‹auto simp: ereal-tendsto-simps›)
  have FTC2: interval-lebesgue-integral lborel (ereal μ) (ereal a) g = F a - (-
1)
    by (rule interval-integral-FTC-nonneg[where F=F])
       (use mua' deriv isCont-g g-nonneg A B in
        ‹auto simp: ereal-tendsto-simps›)
  have intg-oo: set-integrable lborel {μ<..<a} g
    using FTC1 by simp
  have int-oo: (∫ x∈{μ<..<a}. g x ∂lborel) = 1 - (1 + ξ * y) powr (- 1 / ξ)
    using FTC2 mua' by (simp add: interval-lebesgue-integral-def Fa)
  have dens-eq: GPD-density ξ μ σ t = g t if t: μ < t t < a for t
  proof(cases 0 < ξ)
    case True
    thus ?thesis
      using t by (simp add: GPD-density-xi-pos g-def)
  next
    case False
    with xi have xi-neg: ξ < 0
      by simp
    have - σ ≤ ξ * (a - μ)
      using xiy sigma unfolding y-def by (simp add: pos-le-divide-eq)
    moreover have ξ * (a - μ) ≤ ξ * (t - μ)
      using t xi-neg by (intro mult-left-mono-neg) auto
    ultimately have - σ ≤ ξ * (t - μ)
      by linarith
    from divide-right-mono-neg[OF this, of ξ] xi-neg xi
```

```
    have t − μ ≤ − σ / ξ
      by simp
    thus ?thesis
      using t xi-neg by (simp add: GPD-density-xi-neg g-def)
  qed
  have ae-eq: AE x in lborel. indicat-real {..a} x *R GPD-density ξ μ σ x
                            = indicat-real {μ<..<a} x *R g x
    using AE-lborel-singleton[of μ] AE-lborel-singleton[of a]
  proof eventually-elim
    case (elim x)
    consider x < μ | μ < x x < a | a < x
      using elim mua' by argo
    thus ?case
      by cases (simp-all add: indicator-def GPD-density-eq-0 mua' less-le-not-le
dens-eq)
  qed
  have intg: set-integrable lborel {..a} (GPD-density ξ μ σ)
  proof −
    have integrable lborel (λx. indicat-real {..a} x *R GPD-density ξ μ σ x)
          ⟷ integrable lborel (λx. indicat-real {μ<..<a} x *R g x)
      using ae-eq by (intro integrable-cong-AE) auto
    thus ?thesis
      using intg-oo by (simp add: set-integrable-def)
  qed
  have integral-eq: (∫ x∈{..a}. GPD-density ξ μ σ x ∂lborel) = 1 − (1 + ξ ∗ y)
powr (− 1 / ξ)
  proof −
    have (∫ x∈{..a}. GPD-density ξ μ σ x ∂lborel)
          = (∫ x. indicat-real {μ<..<a} x *R g x ∂lborel)
      unfolding set-lebesgue-integral-def
      using ae-eq by (intro integral-cong-AE) auto
    also have ... = 1 − (1 + ξ ∗ y) powr (− 1 / ξ)
      using int-oo by (simp add: set-lebesgue-integral-def)
    finally show ?thesis .
  qed
  have nn-eq: (∫+x∈{..a}. GPD-density ξ μ σ x ∂lborel)
            = ennreal (1 − (1 + ξ ∗ y) powr (− 1 / ξ))
  proof −
    have (∫+x∈{..a}. GPD-density ξ μ σ x ∂lborel)
          = (∫+x. ennreal (indicat-real {μ<..<a} x *R g x) ∂lborel)
      using ae-eq by (intro nn-integral-cong-AE)
                     (auto elim!: eventually-mono simp: indicator-def)
    also have ... = ennreal (∫ x. indicat-real {μ<..<a} x *R g x ∂lborel)
      by (intro nn-integral-eq-integral)
         (use intg-oo g-nonneg in ‹auto simp: set-integrable-def indicator-def›)
    also have ... = ennreal (1 − (1 + ξ ∗ y) powr (− 1 / ξ))
      using int-oo by (simp add: set-lebesgue-integral-def)
    finally show ?thesis .
  qed
```

```
    show ?thesis
      using intg integral-eq nn-eq by blast
  qed
  from main show set-integrable lborel {..a} (GPD-density ξ μ σ)
    and (∫ x∈{..a}. GPD-density ξ μ σ x ∂lborel) = 1 - (1 + ξ * y) powr (- 1
/ ξ)
    and (∫⁺x∈{..a}. GPD-density ξ μ σ x ∂lborel)
        = ennreal (1 - (1 + ξ * y) powr (- 1 / ξ))
    by blast+
qed

lemma
  fixes a μ σ :: real
  defines y ≡ (a - μ) / σ
  assumes sigma[arith]:0 < σ
    and xi:ξ < 0
    and xiy:ξ * y ≤ - 1
  shows GPD-density-set-integrable3:set-integrable lborel {..a} (GPD-density ξ μ
σ)
    and GPD-density-set-integral3:(∫ x∈{..a}. GPD-density ξ μ σ x ∂lborel) = 1
    and GPD-density-set-nn-integral3:(∫⁺x∈{..a}. GPD-density ξ μ σ x ∂lborel)
= 1
proof -
  define b where b ≡ μ - σ / ξ
  have ba: b ≤ a
    using xiy xi sigma unfolding b-def y-def by (simp add: field-simps)
  have yb1: (b - μ) / σ = - 1 / ξ
    using xi sigma by (simp add: b-def field-simps)
  have mub:μ ≤ b
    using xi by(simp add: b-def divide-nonneg-neg)
  have yb3: - 1 ≤ ξ * ((b - μ) / σ)
    using xi by (simp add: yb1)
  have base0: 1 + ξ * (b - μ) / σ = 0
    using xi sigma by (simp add: b-def field-simps)
  have dens-b: GPD-density ξ μ σ x = 0 if b ≤ x for x
  proof(cases x = b)
    case True
    have b ≤ μ - σ / ξ
      by (simp add: b-def)
    hence GPD-density ξ μ σ b = (1 + ξ * (b - μ) / σ) powr (- 1 - 1 / ξ) / σ
      by (intro GPD-density-xi-neg[OF xi mub])
    thus ?thesis
      by (simp add: True base0)
  next
    case False
    with that
    have b < x
      by simp
    hence μ - σ / ξ < x
```

```
    unfolding b-def by argo
   thus ?thesis
    by (intro GPD-density-xi-neg-eq-0[OF xi])
 qed
 have h1[simp]: set-integrable lborel {..b} (GPD-density ξ μ σ)
  and h2: (∫ x∈{..b}. GPD-density ξ μ σ x ∂lborel) = 1
  and h3: (∫⁺x∈{..b}. GPD-density ξ μ σ x ∂lborel) = 1
  using GPD-density-set-integrable1[OF sigma less-imp-neq[OF xi] mub yb3]
   GPD-density-set-integral1[OF sigma less-imp-neq[OF xi] mub yb3]
   GPD-density-set-nn-integral1[OF sigma less-imp-neq[OF xi] mub yb3] xi
  by (simp-all add: yb1)
 have i2[simp]: set-integrable lborel {b<..a} (GPD-density ξ μ σ)
   by(subst set-integrable-cong[where f'=λx. 0, OF refl refl]) (simp-all add:
dens-b)
 have eq0: (∫ x∈{b<..a}. GPD-density ξ μ σ x ∂lborel) = 0
  by(subst set-integral-cong[where g=λx. 0]) (auto simp: dens-b)
 have eq0': (∫⁺x∈{b<..a}. GPD-density ξ μ σ x ∂lborel) = 0
  by(subst set-nn-integral-cong[where g=λx. 0, OF refl refl]) (auto simp: dens-b)
 show set-integrable lborel {..a} (GPD-density ξ μ σ)
  using ba
  by(subst set-integrable-cong[where A'={..b} ∪ {b<..a}])
   (auto intro!: set-integrable-Un)
 have (∫ x∈{..a}. GPD-density ξ μ σ x ∂lborel) =
    (∫ x∈{..b}. GPD-density ξ μ σ x ∂lborel) + (∫ x∈{b<..a}. GPD-density ξ μ
σ x ∂lborel)
  using ba
  by(subst set-integral-Un[symmetric])
   (auto intro!: arg-cong[where f=λB. set-lebesgue-integral lborel B -])
 also have ... = 1
  by (auto simp: h2 eq0)
 finally show (∫ x∈{..a}. GPD-density ξ μ σ x ∂lborel) = 1 .
 have (∫⁺x∈{..a}. GPD-density ξ μ σ x ∂lborel) =
     (∫⁺x. ennreal (GPD-density ξ μ σ x) * indicator {..b} x
      + ennreal (GPD-density ξ μ σ x) * indicator {b<..a} x ∂lborel)
  using ba by(auto intro!: nn-integral-cong simp: indicator-def)
 also have ... = (∫⁺x∈{..b}. GPD-density ξ μ σ x ∂lborel)
  using ba by(auto intro!: nn-integral-cong simp: indicator-def dens-b)
 finally show (∫⁺x∈{..a}. GPD-density ξ μ σ x ∂lborel) = 1
  by(simp add: h3)
qed

lemma GPD-density-set-integrable:
 assumes sigma[arith]:0 < σ
 shows set-integrable lborel {..a} (GPD-density ξ μ σ)
proof −
 consider ξ = 0
  | ¬ ξ = 0 a ≤ μ
  | ¬ ξ = 0 μ ≤ a − 1 ≤ ξ * ((a − μ) / σ)
  | ¬ ξ = 0 μ ≤ a ξ * ((a − μ) / σ) < − 1
```

```
    by argo
  thus ?thesis
  proof cases
    case 1
    thus ?thesis
      using GPD-density-set-integrable0
      by simp
  next
    case 2
    thus ?thesis
      by (intro GPD-density-set-integrable2[OF sigma])
    thm GPD-density-set-integrable2[OF sigma]
  next
    case 3
    thus ?thesis
      by (intro GPD-density-set-integrable1[OF sigma])
  next
    case 4
    hence xi-neg: ξ < 0
      by (smt (verit, best) assms divide-nonneg-pos mult-nonneg-nonneg)
    show ?thesis
      by (intro GPD-density-set-integrable3[OF sigma xi-neg] less-imp-le 4)
  qed
qed

abbreviation GPD-cdf where GPD-cdf ≡ λξ μ σ x. cdf (density lborel (GPD-density
ξ μ σ)) x

lemma GPD-cdf-set-integral:
  assumes sigma[arith]:0 < σ
  shows GPD-cdf ξ μ σ x = (∫ t∈{..x}. GPD-density ξ μ σ t ∂lborel)
  unfolding cdf-def
  by (intro measure-density-on GPD-density-set-integrable[OF sigma])
     (auto simp: less-imp-le)

corollary GPD-cdf-def:
  assumes sigma[arith]:0 < σ
  shows GPD-cdf ξ μ σ x =
                  (let y = (x − μ) / σ in
                   if ξ = 0 then
                      if x < μ then 0 else 1 − exp (− y)
                   else
                   if x < μ then 0
                   else if − 1 ≤ ξ ∗ y then 1 − (1 + ξ ∗ y) powr (− 1 / ξ)
                   else 1)
proof −
  consider ξ = 0
    | ξ ≠ 0 x < μ
    | ξ ≠ 0 μ ≤ x − 1 ≤ ξ ∗ ((x − μ) / σ)
```

```
    | ξ ≠ 0 μ ≤ x ξ * ((x − μ) / σ) < − 1
    by argo
  thus ?thesis
  proof cases
    case 1
    thus ?thesis
      unfolding GPD-cdf-set-integral[OF sigma]
      by(simp add: GPD-density-set-integral0 del: minus-diff-eq)
  next
    case 2
    thus ?thesis
      using GPD-density-set-integral2[OF sigma less-imp-le]
      by (auto simp add: GPD-cdf-set-integral Let-def)
  next
    case 3
    thus ?thesis
      using GPD-density-set-integral1[OF sigma]
      by (simp add: GPD-cdf-set-integral Let-def)
  next
    case 4
    hence xi-neg: ξ < 0
      by (smt (verit, best) assms mult-nonneg-nonneg zero-le-divide-iff)
    show ?thesis
      using GPD-density-set-integral3[OF sigma xi-neg less-imp-le] 4
      by (simp add: GPD-cdf-set-integral Let-def)
  qed
qed

lemma GPD-cdf-location-scale:
  assumes sigma: 0 < σ
  shows GPD-cdf γ 0 σ x = GPD-cdf γ 0 1 (x / σ)
  using sigma
   by (simp add: GPD-cdf-def[OF sigma] GPD-cdf-def[OF zero-less-one] Let-def
divide-less-0-iff)

lemma EVD-tail-eq-GPD-cdf:
  assumes supp: 1 + γ * x > 0 and x: 0 ≤ x
  shows − ln (EVD-cdf γ 0 1 x) = 1 − GPD-cdf γ 0 1 x
  using x supp
  by (simp add: EVD-cdf-def[of 1] GPD-cdf-def[OF zero-less-one] Let-def)

lemma GPD-cdf-eq0:
  assumes x ≤ μ
  shows GPD-cdf ξ μ σ x = 0
proof −
  have GPD-cdf ξ μ σ x = enn2real (set-nn-integral lborel {..x} (GPD-density ξ
μ σ))
    by(simp add: emeasure-density cdf-def measure-def)
  also have ... = enn2real (set-nn-integral lborel {..x} (λy. 0))
```

```
    using assms
   by(intro arg-cong[where f=enn2real] nn-integral-cong-AE AE-I'[where N={x}])
      (auto simp: GPD-density-def indicator-def)
  finally show ?thesis
    by simp
qed

lemma GPD-cdf-at-bot-0: (GPD-cdf ξ μ σ ⟶ 0) at-bot
proof −
  have ∀F x in at-bot. GPD-cdf ξ μ σ x = 0
    using eventually-le-at-bot[of μ]
    by eventually-elim (intro GPD-cdf-eq0)
  thus ?thesis
    by (rule tendsto-eventually)
qed

lemma GPD-cdf-at-top-1:
  assumes sigma:0 < σ
  shows (GPD-cdf ξ μ σ ⟶ 1) at-top
proof (cases ξ = 0)
  case True
  have ev: ∀F x in at-top. 1 − exp (− (x − μ) / σ) = GPD-cdf ξ μ σ x
    using eventually-ge-at-top[of μ]
    by eventually-elim
       (use True in simp add: GPD-cdf-def[OF sigma] Let-def field-simps mi-
nus-divide-left)
  have lim: ((λx. 1 − exp (− (x − μ) / σ)) ⟶ 1) at-top
  proof −
    have ((λx. exp (− (x − μ) / σ)) ⟶ 0) at-top
      using sigma by real-asymp
    from tendsto-diff[OF tendsto-const this]
    show ?thesis
      by simp
  qed
  show ?thesis
    by (rule Lim-transform-eventually[OF lim ev])
next
  case False
  show ?thesis
  proof(cases 0 < ξ)
    case True
    have ev: ∀F x in at-top. 1 − (1 + ξ ∗ ((x − μ) / σ)) powr (− 1 / ξ)
                                  = GPD-cdf ξ μ σ x
      using eventually-ge-at-top[of μ]
    proof eventually-elim
      case (elim x)
      hence y0: 0 ≤ (x − μ) / σ
        using sigma by (intro divide-nonneg-pos) auto
      have − 1 ≤ ξ ∗ ((x − μ) / σ)
```

```
      using mult-nonneg-nonneg[OF less-imp-le[OF True] y0] by linarith
    thus ?case
      using True elim by (simp add: GPD-cdf-def[OF sigma] Let-def)
  qed
  have lim: ((λx. 1 − (1 + ξ ∗ ((x − μ) / σ)) powr (− 1 / ξ)) ⟶ 1) at-top
  proof −
    have 1: LIM x at-top. 1 + ξ ∗ ((x − μ) / σ) :> at-top
    proof −
      have LIM x at-top. (x − μ) / σ :> at-top
        using sigma by real-asymp
      hence LIM x at-top. ξ ∗ ((x − μ) / σ) :> at-top
        by (intro filterlim-tendsto-pos-mult-at-top[OF tendsto-const True])
      thus ?thesis
        by (intro filterlim-tendsto-add-at-top[OF tendsto-const])
    qed
    have ((λx. (1 + ξ ∗ ((x − μ) / σ)) powr (− 1 / ξ)) ⟶ 0) at-top
     by (intro tendsto-neg-powr[OF - 1]) (use True in ‹auto simp: divide-neg-pos›)
    from tendsto-diff[OF tendsto-const this]
    show ?thesis
      by simp
  qed
  show ?thesis
    by (rule Lim-transform-eventually[OF lim ev])
next
  case False
  with ‹ξ ≠ 0› have xi-neg: ξ < 0
    by simp
  have ev: ∀F x in at-top. GPD-cdf ξ μ σ x = 1
    using eventually-ge-at-top[of μ − σ / ξ]
  proof eventually-elim
    case (elim x)
    note neg = divide-pos-neg[OF sigma xi-neg]
    hence mux:μ ≤ x
      using elim neg by linarith
    have eq: (− σ / ξ) / σ = − 1 / ξ
      using xi-neg sigma by (simp add: field-simps)
    have (− σ / ξ) / σ ≤ (x − μ) / σ
      using elim sigma by (intro divide-right-mono) auto
    hence zb: − 1 / ξ ≤ (x − μ) / σ
      using sigma by (simp add: eq)
    have ξ ∗ ((x − μ) / σ) ≤ ξ ∗ (− 1 / ξ)
      by (rule mult-left-mono-neg[OF zb less-imp-le[OF xi-neg]])
    hence xiy-le: ξ ∗ ((x − μ) / σ) ≤ − 1
      using xi-neg by simp
    consider ξ ∗ ((x − μ) / σ) = − 1 | ξ ∗ ((x − μ) / σ) < − 1
      using xiy-le by argo
    thus ?case
    proof cases
      case 1
```

```
    thus ?thesis
      using mux xi-neg by (simp add: GPD-cdf-def[OF sigma] Let-def)
  next
    case 2
    thus ?thesis
      using mux xi-neg by (auto simp: GPD-cdf-def[OF sigma] Let-def)
  qed
 qed
 thus ?thesis
  by (rule tendsto-eventually)
 qed
qed

lemma
 assumes sigma[arith]:0 < σ
 shows GPD-density-integrable:integrable lborel (GPD-density ξ μ σ)
  and GPD-density-integral:(∫ x. GPD-density ξ μ σ x ∂lborel) = 1
  and GPD-density-nn-integral:(∫⁺x. GPD-density ξ μ σ x ∂lborel) = 1
proof −
 have lim: ((λa. (∫ x∈{..a}. GPD-density ξ μ σ x ∂lborel)) ⟶ 1) at-top
 proof −
  have ∗: (∫ x∈{..a}. GPD-density ξ μ σ x ∂lborel) = GPD-cdf ξ μ σ a for a
    unfolding cdf-def
    by(subst measure-density-on)
      (auto simp: GPD-density-set-integrable[OF sigma])
  show ?thesis
    unfolding ∗ by(rule GPD-cdf-at-top-1[OF sigma])
 qed
 show integ: integrable lborel (GPD-density ξ μ σ)
  and eq2: (∫ x. GPD-density ξ μ σ x ∂lborel) = 1
  using lim GPD-density-set-integrable[OF sigma]
    integrable-monotone-convergence-at-top[where f=GPD-density ξ μ σ and
x=1,
     OF - AE-I2[OF GPD-density-nonneg[OF less-imp-le[OF sigma]]]]
   integral-monotone-convergence-at-top[where f=GPD-density ξ μ σ and x=1,
     OF - AE-I2[OF GPD-density-nonneg[OF less-imp-le[OF sigma]]]]
  by(auto simp: set-integrable-def set-lebesgue-integral-def mult.commute)
 thus (∫⁺x. GPD-density ξ μ σ x ∂lborel) = 1
  by(subst nn-integral-eq-integral) auto
qed

lemma prob-space-GPD:
 assumes sigma[arith]:0 < σ
 shows prob-space (density lborel (GPD-density ξ μ σ))
 by(auto intro!: prob-spaceI simp: emeasure-density GPD-density-nn-integral[OF
sigma])

corollary real-distribution-GPD:
 0 < σ ⟹ real-distribution (density lborel (GPD-density ξ μ σ))
```

by(*simp add*: *real-distribution-def real-distribution-axioms-def prob-space-GPD*)

lemma *isCont-GPD-cdf*:
assumes *sigma*[*arith*]: $0 < \sigma$
shows *isCont* (*GPD-cdf* ξ μ σ) x
using *continuous-on-LBINT-com*[*OF GPD-density-integrable*[*OF sigma*]]
GPD-cdf-set-integral[*OF sigma*]
by *simp*

lemma *right-continuous-mono-GPD-cdf*:
assumes *sigma*[*arith*]: $0 < \sigma$
shows *right-continuous-mono* (*GPD-cdf* ξ μ σ) *0 1*
by(*auto intro*!: *right-continuous-mono-cdf real-distribution-GPD*)

corollary *GPD-cdf-nonneg*:$0 < \sigma \Longrightarrow 0 \leq$ *GPD-cdf* ξ μ σ x
and *GPD-cdf-le-1*:$0 < \sigma \Longrightarrow$ *GPD-cdf* ξ μ σ $x \leq 1$
using *right-continuous-mono-GPD-cdf right-continuous-mono.a-le-f*[*of - 0 1*]
right-continuous-mono.f-le-b[*of - 0 1*]
by *fastforce+*

lemma *GPD-normalization*:
fixes γ μ σ x :: *real*
assumes *sigma*[*arith*]: $0 < \sigma$
defines $a \equiv (\lambda n.\ \sigma * (\textit{if } \gamma = 0 \textit{ then } 1 \textit{ else real } n \textit{ powr } \gamma))$
defines $b \equiv (\lambda n.\ \mu + \sigma * (\textit{if } \gamma = 0 \textit{ then ln } (\textit{real } n) \textit{ else } (\textit{real } n \textit{ powr } \gamma - 1)\ /$
$\gamma))$
shows $(\lambda n.$ *GPD-cdf* γ μ σ $(a\ n * x + b\ n)\ \hat{}\ n) \longrightarrow$ *EVD-cdf* γ *0 1* x
proof(*cases* $\gamma = 0$)
assume *0*:$\gamma = 0$
have $\forall_F\ n$ *in sequentially*. $|x| <$ *ln* (*real n*)
using *filterlim-compose*[*OF ln-at-top filterlim-real-sequentially*]
by(*auto simp*: *filterlim-at-top-dense*)
hence *pos*: $\forall_F\ n$ *in sequentially*. $\sigma * x + \sigma *$ *ln* (*real n*) > 0
by *eventually-elim* (*smt* (*verit*, *ccfv-SIG*) *sigma distrib-left mult-pos-pos*)
moreover have $\forall_F\ n$ *in sequentially*. $\mu \leq a\ n * x + b\ n$
using *pos* **by** *eventually-elim* (*simp add*: *0 a-def b-def*)
moreover have $\forall_F\ n$ *in sequentially*. $0 <$ *real n*
by *simp*
ultimately have $\forall_F\ n$ *in sequentially*. (*GPD-cdf* γ μ σ $(a\ n * x + b\ n))\hat{}n$
$= (1 - \textit{exp } (-x)\ /\ \textit{real } n)\hat{}n$
unfolding *GPD-cdf-def*[*OF sigma*]
by *eventually-elim* (*auto simp*: *0 b-def a-def add-divide-distrib exp-diff*)
moreover have $(\lambda n.\ (1 - \textit{exp } (-x)\ /\ \textit{real } n)\hat{}n) \longrightarrow \textit{exp } (-\textit{exp } (-x))$
using *tendsto-exp-limit-sequentially*[*of* $-$ *exp* $(-x)$]
by *simp*
ultimately show *?thesis*
using *tendsto-cong* **by**(*fastforce simp*: *EVD-cdf-def 0*)
next
assume *0*:$\gamma \neq 0$

```
  define y where y ≡ (λn::nat. (real n powr γ * (1 + γ * x) - 1) / γ)
  have arg: a n * x + b n = μ + σ * y n for n
    using 0 by(simp add: a-def b-def y-def field-simps)
  have gy: γ * y n = real n powr γ * (1 + γ * x) - 1 for n
    using 0 by(simp add: y-def)
  have cdf-eq: GPD-cdf γ μ σ (a n * x + b n)
                = (if y n < 0 then 0
                    else if - 1 ≤ γ * y n then 1 - (1 + γ * y n) powr (- 1 / γ)
else 1) for n
   unfolding arg using 0 by(simp add: GPD-cdf-def[OF sigma] Let-def mult-less-0-iff)
  show ?thesis
  proof(cases 0 < 1 + γ * x)
    case c-pos: True
    have ge: - 1 ≤ γ * y n for n
      using c-pos by(simp add: gy)
    have ev0: ∀F n in sequentially. 0 ≤ y n
    proof(cases 0 < γ)
      case True
      have filterlim (λn. real n powr γ) at-top sequentially
      by(rule filterlim-compose[OF real-powr-at-top[OF True] filterlim-real-sequentially])
      hence ∀F n in sequentially. 1 / (1 + γ * x) ≤ real n powr γ
        by(simp add: filterlim-at-top)
      thus ?thesis
      proof eventually-elim
        case (elim n)
        hence 1 ≤ real n powr γ * (1 + γ * x)
          using c-pos by(simp add: pos-divide-le-eq)
        thus 0 ≤ y n
          unfolding y-def using True by(intro divide-nonneg-pos) auto
      qed
    next
      case False
      with 0 have neg: γ < 0
        by simp
      have (λn. real n powr γ) ⟶ 0
        by(rule tendsto-neg-powr[OF neg filterlim-real-sequentially])
      hence ∀F n in sequentially. real n powr γ < 1 / (1 + γ * x)
        using c-pos by(intro order-tendstoD) auto
      thus ?thesis
      proof eventually-elim
        case (elim n)
        hence real n powr γ * (1 + γ * x) ≤ 1
          using c-pos by(simp add: pos-less-divide-eq)
        thus 0 ≤ y n
          unfolding y-def using neg by(intro divide-nonpos-neg) auto
      qed
    qed
    have ev: ∀F n in sequentially.
              GPD-cdf γ μ σ (a n * x + b n) = 1 - (1 + γ * x) powr (- 1 / γ)
```

```
/ real n
    proof(safe intro!: eventually-mp[OF eventually-sequentiallyI[of 1] ev0])
      fix n :: nat
      assume n: 1 ≤ n and yn: 0 ≤ y n
      have n0: 0 < real n
        using n by simp
       have (1 + γ * y n) powr (− 1 / γ) = (real n powr γ * (1 + γ * x)) powr
(− 1 / γ)
        by(simp add: gy)
      also have ... = (real n powr γ) powr (− 1 / γ) * (1 + γ * x) powr (− 1 / γ)
        using c-pos by(simp add: powr-mult)
      also have ... = (1 + γ * x) powr (− 1 / γ) / real n
        using 0 n0 by(simp add: powr-powr powr-minus-divide)
      finally show GPD-cdf γ μ σ (a n * x + b n)
                    = 1 − (1 + γ * x) powr (− 1 / γ) / real n
        using yn ge by(simp add: cdf-eq)
    qed
    have ev2: ∀F n in sequentially. GPD-cdf γ μ σ (a n * x + b n) ^ n
                  = (1 − (1 + γ * x) powr (− 1 / γ) / real n) ^ n
      using ev by eventually-elim simp
    have lim: (λn. (1 − (1 + γ * x) powr (− 1 / γ) / real n) ^ n)
                ⟶ exp (− ((1 + γ * x) powr (− 1 / γ)))
      using tendsto-exp-limit-sequentially[of − ((1 + γ * x) powr (− 1 / γ))]
      by simp
    have EVD: EVD-cdf γ 0 1 x = exp (− ((1 + γ * x) powr (− 1 / γ)))
      using c-pos 0 by(simp add: EVD-cdf-def)
    show ?thesis
      unfolding EVD by(subst tendsto-cong[OF ev2]) (rule lim)
  next
    case c-nonpos: False
    have ev: ∀F n in sequentially.
              GPD-cdf γ μ σ (a n * x + b n) ^ n = EVD-cdf γ 0 1 x
    proof(intro eventually-sequentiallyI[of 1])
      fix n :: nat
      assume n: 1 ≤ n
      hence 0 < real n powr γ
        by simp
      hence le: γ * y n ≤ − 1
        using c-nonpos by(simp add: gy mult-nonneg-nonpos)
      show GPD-cdf γ μ σ (a n * x + b n) ^ n = EVD-cdf γ 0 1 x
      proof(cases 0 < γ)
        case True
        hence y n < 0
          using le by (smt (verit, best) mult-nonneg-nonneg)
        hence GPD-cdf γ μ σ (a n * x + b n) = 0
          by(simp add: cdf-eq)
        thus ?thesis
          using n True c-nonpos by(simp add: EVD-cdf-def)
      next
```

```
      case False
      with 0 have neg: γ < 0
        by simp
      hence yn: ¬ y n < 0
        using le by (smt (verit, best) mult-less-0-iff)
      have GPD-cdf γ μ σ (a n * x + b n) = 1
      proof(cases γ * y n = − 1)
        case True
        thus ?thesis
          using yn 0 neg by(simp add: cdf-eq)
      next
        case False
        with le have γ * y n < − 1
          by simp
        thus ?thesis
          using yn by(simp add: cdf-eq)
      qed
      thus ?thesis
        using neg c-nonpos by(simp add: EVD-cdf-def)
    qed
  qed
  show ?thesis
    using ev by(simp add: tendsto-eventually)
 qed
qed

end
```

3 Domain of Attraction

```
theory Domain-of-Attraction
imports
  HOL−Probability.Weak-Convergence
  Cauchy-Functional-Equation.Cauchy-Functional-Equation
  Pareto-Distribution
begin
```

This theory file is based on Chapter 1 of the book by de Haan and Ferreira [2].

3.1 Definitions

We work with distribution functions as plain functions *real* ⇒ *real*, as does the book. The corresponding statements for i.i.d. random variables are given in Section 5.

3.1.1 Degenerate Distributions

definition *degenerate-df* :: (*real* ⇒ *real*) ⇒ *bool* **where**
degenerate-df F ⟷ (∀ *x*. *F x* = *0* ∨ *F x* = *1*)

lemma *degenerate-df-iff-indicator*:
assumes *right-continuous-mono F 0 1*
shows *degenerate-df F* ⟷ (∃ *c*. *F* = (λ*x*. *indicat-real* {*c*..} *x*))
proof −
interpret *right-continuous-mono F 0 1* **by** *fact*
show *?thesis*
proof
assume *deg*: *degenerate-df F*
hence *F01*: ⋀*x*. *F x* = *0* ∨ *F x* = *1*
by (*auto simp*: *degenerate-df-def*)
obtain *z* **where** *z*: *F z* = *1*
using *order-tendstoD*[*OF top*, *of 1/2*] *F01*
by (*fastforce dest*!: *eventually-happens'*[*OF trivial-limit-at-top-linorder*])
obtain *w* **where** *w*: *F w* = *0*
using *order-tendstoD*[*OF bot*, *of 1/2*] *F01*
by (*fastforce dest*!: *eventually-happens'*[*OF trivial-limit-at-bot-linorder*])
define *c* **where** *c* = *Inf* {*x*. *F x* = *1*}
have *ne*: {*x*. *F x* = *1*} ≠ {}
using *z* **by** *auto*
have *bdd*: *bdd-below* {*x*. *F x* = *1*}
proof (*intro bdd-belowI*[*of* - *w*])
fix *x*
assume *x* ∈ {*x*. *F x* = *1*}
thus *w* ≤ *x*
using *monoD*[*OF mono*, *of x w*] *w* **by** *force*
qed
have *F1*: *F x* = *1* **if** *c* < *x* **for** *x*
proof −
have *Inf* {*x*. *F x* = *1*} < *x*
using *that* **by** (*simp add*: *c-def*)
from *cInf-lessD*[*OF ne this*] **obtain** *x'* **where** *F x'* = *1* *x'* < *x*
by *auto*
thus *?thesis*
using *monoD*[*OF mono*, *of x' x*] *F01*[*of x*] **by** *auto*
qed
have *F0*: *F x* = *0* **if** *x* < *c* **for** *x*
using *cInf-lower*[*OF* - *bdd*] *that F01*[*of x*] **by** (*fastforce simp*: *c-def leD*)
have *Fc*: *F c* = *1*
proof (*rule tendsto-unique*[*of at-right c F*])
show (*F* ⟶ *F c*) (*at-right c*)
using *cont*[*of c*] **by** (*simp add*: *continuous-within*)
show (*F* ⟶ *1*) (*at-right c*)
by (*intro tendsto-eventually eventually-mono*[*OF eventually-at-right-less*])
(*simp add*: *F1*)
qed *auto*

```
    have F = (λx. indicat-real {c..} x)
    proof
      fix x
      consider x < c | x = c | c < x
        by argo
      thus F x = indicat-real {c..} x
        by cases (auto simp: F0 F1 Fc indicator-def)
    qed
    thus ∃ c. F = (λx. indicat-real {c..} x) ..
  qed(auto simp: degenerate-df-def indicator-def)
qed

lemma EVD-cdf-nondegenerate:
  assumes [arith]:0 < σ
  shows ¬ degenerate-df (EVD-cdf ξ μ σ)
  by(auto simp: degenerate-df-def EVD-cdf-def Let-def intro!: exI[where x=μ])
```

3.1.2 Domain of Attraction

```
definition domain-of-attraction :: (real ⇒ real) ⇒ (real ⇒ real) ⇒ bool
  (infix ‹(in'-domain'-attraction'-of)› 50) where
domain-of-attraction F G ⟷
   right-continuous-mono F 0 1 ∧ right-continuous-mono G 0 1 ∧
   (∃ a b. (∀ n::nat. 0 < a n) ∧ weak-conv (λn x. F (a n * x + b n) ^ n) G)

lemma domain-of-attractionD:
  assumes domain-of-attraction F G
  shows right-continuous-mono F 0 1 right-continuous-mono G 0 1
    and ∃ a b. (∀ n::nat. 0 < a n) ∧ weak-conv (λn x. F (a n * x + b n) ^ n) G
  using assms by(auto simp: domain-of-attraction-def)

lemma domain-of-attractionI-eventually-pos:
  assumes F:right-continuous-mono F 0 1
    and G:right-continuous-mono G 0 1
    and a-pos:∀F n in at-top. a n > 0
    and wc:weak-conv (λn x. F (a n * x + b n) ^ n) G
  shows domain-of-attraction F G
proof −
  obtain k where a:⋀n. k ≤ n ⟹ a n > 0
    using a-pos eventually-sequentially by auto
  define a' where a' ≡ λn. if n < k then 1 else a n
  have a'-pos:a' n > 0 for n
    using a by(cases n < k) (auto simp: a'-def)
  have ev:∀F n in sequentially.
        F (a n * x + b n) ^ n = F (a' n * x + b n) ^ n for x
    by(auto intro!: eventually-sequentiallyI[where c=k] simp: a'-def)
  hence wc':weak-conv (λn x. F (a' n * x + b n) ^ n) G
    using wc by(auto simp: weak-conv-cong[OF ev])
  thus ?thesis
```

```
    unfolding domain-of-attraction-def
    using F G a'-pos
    by(intro conjI exI[where x=a',OF exI[where x=b]]) auto
qed

lemma domain-of-attraction-EVD-cdf-rescale:
  fixes γ μ1 μ2 σ1 σ2 :: real
  assumes sigma1[arith]: 0 < σ1 and sigma2[arith]: 0 < σ2
    and dom: domain-of-attraction F (EVD-cdf γ μ1 σ1)
  shows domain-of-attraction F (EVD-cdf γ μ2 σ2)
proof −
  obtain a b where a: ⋀n. 0 < a n
    and wc: weak-conv (λn x. F (a n * x + b n) ^ n) (EVD-cdf γ μ1 σ1)
    and F: right-continuous-mono F 0 1
    using dom unfolding domain-of-attraction-def by blast
  define a' where a' = (λn. a n * σ1 / σ2)
  define b' where b' = (λn. a n * μ1 − a n * σ1 * μ2 / σ2 + b n)
  have a'-pos: 0 < a' n for n
    using a by (simp add: a'-def)
  have arg-eq: a' n * y + b' n = a n * (σ1 / σ2 * (y − μ2) + μ1) + b n for n y
    by (simp add: a'-def b'-def field-simps)
  have fun-eq: (λn. F (a' n * y + b' n) ^ n)
                = (λn. F (a n * (σ1 / σ2 * (y − μ2) + μ1) + b n) ^ n) for y
    by (simp add: arg-eq)
  have EVD-eq: EVD-cdf γ μ1 σ1 (σ1 / σ2 * (y − μ2) + μ1) = EVD-cdf γ μ2
σ2 y for y
  proof −
    have EVD-cdf γ μ1 σ1 (σ1 / σ2 * (y − μ2) + μ1)
          = EVD-cdf γ 0 1 ((σ1 / σ2 * (y − μ2) + μ1 − μ1) / σ1)
      by (subst EVD-cdf-location-scale) simp-all
    also have ... = EVD-cdf γ 0 1 ((y − μ2) / σ2)
      by simp
    also have ...  = EVD-cdf γ μ2 σ2 y
      by (simp add: EVD-cdf-location-scale)
    finally show ?thesis .
  qed
  have wc': weak-conv (λn y. F (a' n * y + b' n) ^ n) (EVD-cdf γ μ2 σ2)
    using wc isCont-EVD-cdf[OF sigma1]
    unfolding weak-conv-def EVD-eq[symmetric] fun-eq
    by simp
  show ?thesis
    unfolding domain-of-attraction-def
    using F a'-pos right-continuous-mono-EVD-cdf[OF sigma2] wc' by blast
qed

theorem domain-of-attraction-EVD-cdf-iff:
  fixes γ μ σ :: real
  assumes sigma[arith]: 0 < σ
   shows domain-of-attraction F (EVD-cdf γ μ σ) ⟷ domain-of-attraction F
```

```
(EVD-cdf γ 0 1)
  using domain-of-attraction-EVD-cdf-rescale[OF sigma,of 1]
        domain-of-attraction-EVD-cdf-rescale[OF zero-less-one sigma] by auto

lemma EVD-cdf-MDA:
  assumes sigma[arith]:0 < σ
  shows domain-of-attraction (EVD-cdf ξ μ σ) (EVD-cdf ξ μ σ)
proof −
  interpret real-distribution density lborel (EVD-density ξ μ σ)
    by(rule real-distribution-EVD[OF sigma])
  define a where a = (λn::nat. if n = 0 ∨ ξ = 0 then 1 else real n powr ξ)
  define b where b = (λn::nat. if n = 0 then 0 else if ξ = 0 then σ * ln (real n)
                                 else (μ − σ / ξ) * (1 − real n powr ξ))
  have a-pos: 0 < a n for n
    by (auto simp: a-def)
  have eq: EVD-cdf ξ μ σ (a n * x + b n) ^ n = EVD-cdf ξ μ σ x if n: 0 < n
for n x
  proof −
    have npos: (0::real) < real n
      using n by simp
    have EVD-cdf ξ μ σ (a n * x + b n) ^ n = EVD-cdf ξ 0 1 ((a n * x + b n
− μ) / σ) ^ n
      by(simp add: EVD-cdf-location-scale)
    also have ... = EVD-cdf ξ 0 1 (if ξ = 0 then (a n * x + b n − μ) / σ − ln
(real n)
                    else real n powr (− ξ) * ((a n * x + b n − μ) / σ)
                        + (real n powr (− ξ) − 1) / ξ)
      by (rule EVD-cdf-max-stable[OF n])
    also have (if ξ = 0 then (a n * x + b n − μ) / σ − ln (real n)
                    else real n powr (− ξ) * ((a n * x + b n − μ) / σ)
                        + (real n powr (− ξ) − 1) / ξ) = (x − μ) / σ
      by (cases ξ = 0)
         (use npos in auto simp: a-def b-def field-simps powr-minus-divide)
    also have EVD-cdf ξ 0 1 ((x − μ) / σ) = EVD-cdf ξ μ σ x
      by(simp add: EVD-cdf-location-scale)
    finally show ?thesis .
  qed
  have wc: weak-conv (λn x. EVD-cdf ξ μ σ (a n * x + b n) ^ n) (EVD-cdf ξ μ
σ)
    unfolding weak-conv-def
   by (intro allI impI tendsto-eventually eventually-mono[OF eventually-gt-at-top[of
0]])
       (simp add: eq)
  show ?thesis
    unfolding domain-of-attraction-def
    using right-continuous-mono-axioms a-pos wc by blast
qed

lemma GPD-cdf-MDA:
```

```
  fixes γ μ σ :: real
  assumes 0 < σ
  shows domain-of-attraction (GPD-cdf γ μ σ) (EVD-cdf γ 0 1)
  using assms
  by(intro domain-of-attractionI-eventually-pos[OF - -  eventually-sequentiallyI[of
1],
       where a=λn. σ * (if γ = 0 then 1 else real n powr γ)
        and b=λn. μ + σ * (if γ = 0 then ln (real n) else (real n powr γ - 1) /
γ)]
      right-continuous-mono-GPD-cdf right-continuous-mono-EVD-cdf
      LIMSEQ-imp-weak-conv GPD-normalization) auto

lemma domain-of-attraction-affine:
  fixes c d :: real
  assumes c[arith]:0 < c and dm:domain-of-attraction F G
  shows domain-of-attraction (λx. F ((x - d) / c)) G
proof -
  have F:right-continuous-mono F 0 1
    and G:right-continuous-mono G 0 1
  and ∃ a b. (∀ n. 0 < a n) ∧ weak-conv (λn x. F (a n * x + b n) ^ n) G
    using domain-of-attractionD[OF dm] by auto
  then obtain a b where a:⋀n. 0 < a n and wc:weak-conv (λn x. F (a n * x +
b n) ^ n) G
    by blast
  interpret F: right-continuous-mono F 0 1
    by fact
  have lim1:LIM x at-top. (x - d) / c :> at-top
    using filterlim-tendsto-pos-mult-at-top[OF tendsto-const[of 1 / c] -
      filterlim-tendsto-add-at-top[OF tendsto-const[of - d at-top],of λx. x]]
    by(simp add: filterlim-ident)
  have lim2:LIM x at-bot. (x - d) / c :> at-bot
    using filterlim-tendsto-pos-mult-at-bot[OF tendsto-const[of 1 / c] -
      filterlim-tendsto-add-at-bot-iff
      [THEN iffD2,OF tendsto-const[of - d at-bot],of λx. x]]
    by(simp add: filterlim-ident)
  have 1:right-continuous-mono (λx. F ((x - d) / c)) 0 1
  proof
    fix x :: real
    have *:(λx. (x - d) / c) ` {x<..} = {(x - d) / c<..}
      by(intro image-mono-on-continuous-on-greaterThan)
        (auto intro!: strict-mono-onI continuous-on-divide continuous-on-diff
          simp: divide-strict-right-mono lim1)
    show continuous (at-right x) (λx. F ((x - d) / c))
      by(rule continuous-within-compose2[where g=F])
        (auto simp: continuous-cdivide-iff continuous-diff * F.cont)
    show  ((λx. F ((x - d) / c)) ⟶ 0) at-bot
      ((λx. F ((x - d) / c)) ⟶ 1) at-top
      by(auto intro!: filterlim-compose[where g=F,OF - lim1] F.bot F.top
          filterlim-compose[where g=F,OF - lim2])
```

```
    show mono (λx. F ((x − d) / c))
      by(auto intro!: monoD[OF F.mono] monoI simp: divide-right-mono)
  qed
  show ?thesis
    by(auto intro!: exI[where x=λn. c ∗ a n,OF conjI[OF - exI[where x=λn. c
∗ b n + d]]]
        simp: domain-of-attraction-def 1 G a distrib-left[symmetric] mult.assoc wc)
qed
```

3.2 Fisher–Tippett–Gnedenko Theorem

The following two definitions correspond to the functions U and D in equation (1.1.6) in the book.

```
abbreviation tail-quantile :: (real ⇒ real) ⇒ real ⇒ real where
tail-quantile F t ≡ ginv F (1 − 1 / t)
```

tail-quantile is exactly the *return level* associated with *return period t*.

```
abbreviation limit-quantile :: (real ⇒ real) ⇒ real ⇒ real where
limit-quantile G y ≡ ginv G (exp (− (1 / y)))

lemma lt-tail-quantile-iff:
  fixes F :: real ⇒ real
  assumes F: right-continuous-mono F 0 1 and s: 1 < s
  shows w < tail-quantile F s ⟷ 1 / s < 1 − F w
proof −
  have p: 0 < 1 − 1 / s 1 − 1 / s < 1
    using s by simp-all
  show ?thesis
    using right-continuous-mono.ginv-le-iff[OF F p, of w] by linarith
qed

lemma tail-quantile-le-of-lt-inv:
  assumes F: right-continuous-mono F 0 1
    and Ft[arith]: F t < 1
    and u1[arith]: 1 < u and u2[arith]: u < 1 / (1 − F t)
  shows tail-quantile F u ≤ t
  by(subst right-continuous-mono.ginv-le-iff[OF F])
    (use pos-less-divide-eq[THEN iffD1,OF - u2] in simp-all add: field-simps)

lemma t-lt-tail-quantile-of-gt-inv:
  fixes F :: real ⇒ real and t u :: real
  assumes F: right-continuous-mono F 0 1
    and Ft: F t < 1
    and u: 1 / (1 − F t) < u
  shows t < tail-quantile F u
 by (smt (verit, best) assms ln-divide-pos ln-less-cancel-iff right-continuous-mono.a-le-f
      right-continuous-mono.ginv-le-iff zero-less-divide-1-iff)
```

```
lemma tail-quantile-at-top:
  fixes F :: real ⇒ real
  assumes F: right-continuous-mono F 0 1
    and unbdd: ⋀x. F x < 1
  shows filterlim (tail-quantile F) at-top at-top
  unfolding filterlim-at-top
proof
  interpret right-continuous-mono F 0 1
    by fact
  fix M :: real
  show ∀F t in at-top. M ≤ tail-quantile F t
    using eventually-gt-at-top[of 1 / (1 − F M)] eventually-gt-at-top[of 1::real]
    by eventually-elim
       (use assms less-eq-real-def F t-lt-tail-quantile-of-gt-inv in presburger)
qed

lemma limit-quantile-left-cont:
  assumes G: right-continuous-mono G 0 1 and p: 0 < p
  shows (limit-quantile G ⟶ limit-quantile G p) (at-left p)
proof −
  interpret right-continuous-mono G 0 1
    by fact
  define g where g = (λy::real. exp (− 1 / y))
  have g-tendsto: (g ⟶ g p) (at-left p)
    unfolding g-def using p by (intro tendsto-intros) simp
  have g-ev: ∀F y in at-left p. 0 < y ∧ y < p
    using eventually-at-left-field[where x = p] p by auto
  have g-lt: ∀F y in at-left p. g y < g p
    using p by(auto intro!: eventually-mono[OF g-ev] simp: g-def field-simps)
  note g-at-left = tendsto-imp-filterlim-at-left[OF g-tendsto g-lt]
  have g01: 0 < g p ∧ g p < 1
    unfolding g-def using p by auto
  have filterlim (ginv G) (nhds (ginv G (g p))) (at-left (g p))
    using ginv-left-cont[OF g01] by simp
  from filterlim-compose[OF this g-at-left]
  show ?thesis
    by(simp add: g-def)
qed

lemma limit-quantile-left-cont-scale:
  assumes G: right-continuous-mono G 0 1 and y: 0 < y and p: 0 < p
  shows ((λx. limit-quantile G (x ∗ y)) ⟶ limit-quantile G (p ∗ y)) (at-left p)
proof −
  have mult-tendsto: ((λx. x ∗ y) ⟶ p ∗ y) (at-left p)
    by (intro tendsto-intros)
  have mult-ev: ∀F x in at-left p. 0 < x ∧ x < p
    using eventually-at-left-field[where x = p] p by auto
  have mult-lt: ∀F x in at-left p. x ∗ y < p ∗ y
   using mult-ev by eventually-elim (use y in ‹auto intro!: mult-strict-right-mono›)
```

```
  note mult-at-left = tendsto-imp-filterlim-at-left[OF mult-tendsto mult-lt]
  have py: 0 < p * y
    using y p by simp
  have filterlim (limit-quantile G) (nhds (limit-quantile G (p * y))) (at-left (p *
y))
    using limit-quantile-left-cont[OF G py] by simp
  from filterlim-compose[OF this mult-at-left]
  show ?thesis
    by simp
qed

lemma tail-quantile-mono:
  assumes F: right-continuous-mono F 0 1 and s: 1 < s and st: s ≤ t
  shows tail-quantile F s ≤ tail-quantile F t
proof −
  interpret right-continuous-mono F 0 1
    by fact
  show ?thesis
    using s st ex-ge-b ex-less-a
      by(intro ginv-compose-mono[where A={0<..<1},THEN mono-onD] mono
mono-onI)
       (auto simp: divide-mono)
qed

lemma limit-quantile-mono:
  assumes G: right-continuous-mono G 0 1 and y1: 0 < y1 and y12: y1 ≤ y2
  shows limit-quantile G y1 ≤ limit-quantile G y2
proof −
  interpret right-continuous-mono G 0 1 by fact
  show ?thesis
    using y1 y12 ex-ge-b ex-less-a
      by(intro ginv-compose-mono[where A={0<..<1},THEN mono-onD] mono
mono-onI)
       (auto simp: divide-mono)
qed

lemma limit-quantile-EVD-cdf:
  assumes sigma[arith]:0 < σ
    and ypos[arith]: 0 < y
  shows limit-quantile (EVD-cdf ξ μ σ) y
        = (if ξ = 0 then σ * ln y + μ else σ * (y powr ξ − 1) / ξ + μ)
    (is ginv - ?p = ?xs)
proof −
  have yppos: 0 < y powr ξ
    using ypos by simp
  have iff: ?p ≤ EVD-cdf ξ μ σ x ⟷ ?xs ≤ x for x
  proof −
    consider ξ = 0 | 0 < ξ | ξ < 0 by argo
    thus ?thesis
```

proof *cases*
case *1*
have *?p* $\leq$ *EVD-cdf* ξ μ σ *x* $\longleftrightarrow$ *exp* $(-$ $((x - \mu)$ / $\sigma)) \leq$ *1* / *y*
by (*simp add*: *EVD-cdf-def Let-def 1*)
also have ... $\longleftrightarrow$ *exp* $(-$ $((x - \mu)$ / $\sigma)) \leq$ *exp* $(-$ *ln y*)
by (*simp add*: *exp-minus inverse-eq-divide*)
also have ... $\longleftrightarrow$ *ln y* $\leq (x - \mu)$ / σ
by *simp*
also have ... $\longleftrightarrow$ σ $*$ *ln y* $+$ $\mu \leq x$
by (*auto simp*: *field-simps*)
finally show *?thesis*
by (*simp add*: *1*)
next
case *2*
define *u* **where** *u* = *1* $+$ ξ $*$ $((x - \mu)$ / $\sigma)$
show *?thesis*
proof (*cases 0* $<$ *u*)
assume *u*[*arith*]:*0* $<$ *u*
have *?p* $\leq$ *EVD-cdf* ξ μ σ *x* $\longleftrightarrow$ *u powr* $(-$ *1* / $\xi) \leq$ *1* / *y*
using *2 u* **by** (*auto simp add*: *EVD-cdf-def Let-def u-def*)
also have ... $\longleftrightarrow$ *ln* $(u$ *powr* $(-$ *1* / $\xi)) \leq$ *ln* $(1$ / *y*)
by (*rule ln-le-cancel-iff*[*symmetric*]) *auto*
also have ... $\longleftrightarrow$ $(-$ *1* / $\xi)$ $*$ *ln u* $\leq -$ *ln y*
by (*simp add*: *ln-div*)
also have ... $\longleftrightarrow$ ξ $*$ *ln y* $\leq$ *ln u*
using *2* **by** (*auto simp*: *field-simps*)
also have ... $\longleftrightarrow$ *ln* $(y$ *powr* $\xi) \leq$ *ln u*
by *simp*
also have ... $\longleftrightarrow$ *y powr* $\xi \leq u$
by (*rule ln-le-cancel-iff*) *auto*
also have ... $\longleftrightarrow$ *y powr* $\xi - 1 \leq \xi$ $*$ $((x - \mu)$ / $\sigma)$
by (*simp add*: *u-def*) *linarith*
also have ... $\longleftrightarrow$ *?xs* $\leq x$
using *2* **by** (*simp add*: *field-simps*)
finally show *?thesis* **.**
next
assume *u*:$\neg$ *0* $<$ *u*
have *y*:*0* $<$ σ $*$ *y powr* ξ
using *yppos* **by** *simp*
show *?thesis*
using *2 u add-strict-increasing*[*OF y*]
by(*auto simp*: *EVD-cdf-def Let-def u-def not-less not-le field-simps*)
qed
next
case *3*
define *u* **where** *u* = *1* $+$ ξ $*$ $((x - \mu)$ / $\sigma)$
show *?thesis*
proof (*cases 0* $<$ *u*)
case *True*

```
      have ?p ≤ EVD-cdf ξ μ σ x ⟷ u powr (− 1 / ξ) ≤ 1 / y
        using 3 True by (auto simp add: EVD-cdf-def Let-def u-def)
      also have ... ⟷ ln (u powr (− 1 / ξ)) ≤ ln (1 / y)
        by (rule ln-le-cancel-iff[symmetric]) (use True in auto)
      also have ... ⟷ (− 1 / ξ) * ln u ≤ − ln y
        by (simp add: ln-div)
      also have ... ⟷ ln u ≤ ξ * ln y
        using 3 by (auto simp: field-simps)
      also have ... ⟷ ln u ≤ ln (y powr ξ)
        by simp
      also have ... ⟷ u ≤ y powr ξ
        by (rule ln-le-cancel-iff) (auto simp: True)
      also have ... ⟷ ξ * ((x − μ) / σ) ≤ y powr ξ − 1
        unfolding u-def by auto
      also have ... ⟷ ?xs ≤ x
        using 3 by (simp add: field-simps)
      finally show ?thesis .
    next
      assume u:¬ 0 < u
      have Gx: EVD-cdf ξ μ σ x = 1
        using 3 u by (auto simp: EVD-cdf-def Let-def u-def not-less)
      have ξ * ((x − μ) / σ) < y powr ξ − 1
        using u yppos unfolding u-def by linarith
      hence σ * (y powr ξ − 1) / ξ + μ ≤ x
        using 3 by (simp add: field-simps)
      hence ?xs ≤ x
        using 3 by simp
      moreover have ?p ≤ 1
        by (auto intro!: less-imp-le)
      ultimately show ?thesis
        by (simp add: Gx)
    qed
  qed
  have set-eq: {x. ?p ≤ EVD-cdf ξ μ σ x} = {?xs..}
    using iff by auto
  show ?thesis
    unfolding ginv-def set-eq by simp
qed

corollary isCont-limit-quantile-EVD-cdf:
  assumes sigma[arith]:0 < σ
    and ypos[arith]: 0 < y
  shows isCont (limit-quantile (EVD-cdf ξ μ σ)) y
proof(subst isCont-cong)
  show ∀F x in nhds y.
      limit-quantile (EVD-cdf ξ μ σ) x
      = (if ξ = 0 then σ * ln x + μ else σ * (x powr ξ − 1) / ξ + μ)
    unfolding eventually-nhds
```

```
    by(auto intro!: exI[where x={y/2<..<2*y}] simp: limit-quantile-EVD-cdf)
next
  show isCont (λx. if ξ = 0 then σ * ln x + μ else σ * (x powr ξ - 1) / ξ + μ) y
    by(cases ξ = 0)
      (auto intro!: isCont-add isCont-mult isCont-ln isCont-divide isCont-diff
        simp: continuous-at-within-powr)
qed

lemma dist-function-unique-limit-quantile:
  assumes G:right-continuous-mono G 0 1 and G':right-continuous-mono G' 0 1
    and eq:⋀y. 0 < y ⟹ limit-quantile G y = limit-quantile G' y
  shows G = G'
proof -
  have ginv-eq: ginv G p = ginv G' p if p: 0 < p p < 1 for p
    using eq[of - 1 / ln p] using p by simp
  have le: H' x ≤ H x
    if H: right-continuous-mono H 0 1 and H': right-continuous-mono H' 0 1
      and geq: ⋀p. 0 < p ⟹ p < 1 ⟹ ginv H p = ginv H' p
    for H H' :: real ⇒ real and x
  proof (rule ccontr)
    interpret H: right-continuous-mono H 0 1
      by fact
    interpret H':right-continuous-mono H' 0 1
      by fact
    assume ¬ H' x ≤ H x
    hence lt: H x < H' x
      by simp
    define p where p = (H x + H' x) / 2
    have p1: H x < p p < H' x
      using lt by (simp-all add: p-def)
    have p0: 0 < p p < 1
      using H.a-le-f[of x] p1 H'.f-le-b[of x]
      by auto
    have ginv H' p ≤ x
      using H'.ginv-le-iff p0 p1 by auto
    hence ginv H p ≤ x
      by (simp add: geq p0)
    with p1 show False
      by (simp add: H.ginv-le-iff p0(1,2))
  qed
  show ?thesis
    using le[OF G G' ginv-eq] le[OF G' G ginv-eq[symmetric]]
    by(auto dest: antisym)
qed

lemma isCont-ginv-iff-isCont-limit-quantile:
  assumes G: right-continuous-mono G 0 1
    and y: 0 < y
  shows isCont (ginv G) (exp (- (1 / y))) ⟷ isCont (limit-quantile G) y
```

proof
assume *cont*:*isCont* (*limit-quantile G*) *y*
have *1*: *isCont* (λ*x*. − *1* / *ln x*) (*exp* (− *1* / *y*))
using *y* **by** (*intro continuous-intros*) *simp-all*
have *2*: *isCont* (*limit-quantile G*) (− *1* / *ln* (*exp* (− *1* / *y*)))
using *y* **by**(*simp add*: *cont*)
have *3*: ∀$_F$ *p*′ *in nhds* (*exp* (− (*1* / *y*))). *limit-quantile G* (− *1* / *ln p*′) = *ginv G p*′
using *eventually-nhds-in-open*[*of* {*0*<..<*1*} *exp* (− (*1* / *y*))] *y*
by (*auto elim*!: *eventually-mono*)
show *isCont* (*ginv G*) (*exp* (− (*1* / *y*)))
using *continuous-at-compose*[*OF 1 2*] *isCont-cong*[*OF 3*]
by(*auto simp*: *comp-def*)
next
assume *cont*:*isCont* (*ginv G*) (*exp* (− (*1* / *y*)))
show *isCont* (*limit-quantile G*) *y*
using *continuous-at-compose*[**where** *f*=λ*x*. *exp* (− (*1* / *x*)),*OF - cont*] *y*
by(*simp add*: *comp-def*)
qed

lemma *seq-exp-quantile-tendsto*:
fixes *F G* :: *real* ⇒ *real* **and** *a b* :: *nat* ⇒ *real* **and** *y* :: *real*
assumes *F*: *right-continuous-mono F 0 1* **and** *G*: *right-continuous-mono G 0 1*
and *a*: ⋀*n*. *0* < *a n*
and *wc*: *weak-conv* (λ*n x*. *F* (*a n* ∗ *x* + *b n*) ^ *n*) *G*
and *y*: *0* < *y* **and** *cont*: *isCont* (*limit-quantile G*) *y*
shows (λ*n*. (*limit-quantile F* (*real n* ∗ *y*) − *b n*) / *a n*) ⟶ *limit-quantile G y*
proof −
interpret *G*: *right-continuous-mono G 0 1* **by** *fact*
interpret *F*: *right-continuous-mono F 0 1* **by** *fact*
define *p* **where** *p* = *exp* (− (*1* / *y*))
have *p01*: *0* < *p p* < *1*
using *y* **by** (*auto simp*: *p-def*)
have *cont*′: *isCont* (*ginv G*) *p*
unfolding *p-def* **using** *cont* **by**(*simp add*: *isCont-ginv-iff-isCont-limit-quantile*[*OF G y*])
obtain *z* **where** *z*: *p* < *G z*
using *order-tendstoD*(*1*)[*OF G.top p01*(*2*)]
by (*auto dest*!: *eventually-happens*′[*OF trivial-limit-at-top-linorder*])
obtain *w* **where** *w*: *G w* < *p*
using *order-tendstoD*(*2*)[*OF G.bot p01*(*1*)]
by (*auto dest*!: *eventually-happens*′[*OF trivial-limit-at-bot-linorder*])
have *monoFn*: *mono* (λ*x*. *F* (*a n* ∗ *x* + *b n*) ^ *n*) **for** *n*
using *a* **by**(*auto intro*!: *monoI power-mono monoD*[*OF F.mono*] *F.a-le-f*)
have *main*: (λ*n*. *ginv* (λ*x*. *F* (*a n* ∗ *x* + *b n*) ^ *n*) *p*) ⟶ *ginv G p*
by (*rule tendsto-ginv*[*OF monoFn G.mono - cont*′ *z w*])
(*use wc* **in** ‹*auto simp*: *weak-conv-def*›)
have *eq*: *ginv* (λ*x*. *F* (*a n* ∗ *x* + *b n*) ^ *n*) *p*

```
        = (ginv F (exp (− 1 / (real n * y))) − b n) / a n if n: 0 < n for n
  proof −
    define qn where qn = exp (− 1 / (real n * y))
    have qn01: 0 < qn qn < 1
      using n y by (auto simp: qn-def)
    have qn-pow: qn ^ n = p
      using n y by (simp add: qn-def p-def exp-of-nat-mult[symmetric])
    have iff1: p ≤ F (a n * x + b n) ^ n ⟷ qn ≤ F (a n * x + b n) for x
    proof
      assume qn ≤ F (a n * x + b n)
      hence qn ^ n ≤ F (a n * x + b n) ^ n
        using qn01(1) by (intro power-mono) auto
      thus p ≤ F (a n * x + b n) ^ n
        by (simp add: qn-pow)
    next
      assume *: p ≤ F (a n * x + b n) ^ n
      show qn ≤ F (a n * x + b n)
      proof (rule ccontr)
        assume ¬ qn ≤ F (a n * x + b n)
        hence F (a n * x + b n) ^ n < qn ^ n
          using F.a-le-f n by (intro power-strict-mono) auto
        with * show False
          by (simp add: qn-pow)
      qed
    qed
    have iff2: qn ≤ F v ⟷ ginv F qn ≤ v for v
      using F.ginv-le-iff[OF qn01] by simp
    have iff3: ginv F qn ≤ a n * x + b n ⟷ (ginv F qn − b n) / a n ≤ x for x
      using a[of n] by (auto simp: field-simps)
    have {x. p ≤ F (a n * x + b n) ^ n} = {(ginv F qn − b n) / a n..}
      by (auto simp: iff1 iff2 iff3)
    thus ?thesis
      by (simp add: ginv-def qn-def)
  qed
  have (λn. ginv (λx. F (a n * x + b n) ^ n) p) ⟶ limit-quantile G y
    using main by (simp add: p-def)
  moreover have ∀F n in sequentially. ginv (λx. F (a n * x + b n) ^ n) p
                          = (limit-quantile F (real n * y) − b n) / a n
    using eventually-gt-at-top[of 0] by eventually-elim (simp add: eq)
  ultimately show ?thesis
    by (rule Lim-transform-eventually)
qed

lemma seq-tail-quantile-tendsto:
  fixes F G :: real ⇒ real and a b :: nat ⇒ real and y :: real
  assumes F:right-continuous-mono F 0 1 and G:right-continuous-mono G 0 1
    and dg: ¬ degenerate-df G
    and a:⋀n. 0 < a n
    and wc:weak-conv (λn x. F (a n * x + b n) ^ n) G
```

```
    and y:0 < y and cont:isCont (limit-quantile G) y
  shows (λn. (tail-quantile F (real n * y) - b n) / a n) ⟶ limit-quantile G y
proof (rule metric-LIMSEQ-I)
  interpret F: right-continuous-mono F 0 1 by fact
  interpret G: right-continuous-mono G 0 1 by fact
  fix ε :: real
  assume eps: 0 < ε
  obtain δ where δ: 0 < δ
    ⋀y''. y'' ≠ y ∧ |y'' - y| < δ ⟹ |limit-quantile G y'' - limit-quantile G y| <
ε / 2
    using LIM-D[of - - - ε / 2] eps cont by (fastforce simp: isCont-def)
  have cd: countable {s ∈ {0<..}. ¬ isCont (limit-quantile G) s}
    by (intro mono-on-ctble-discont-open)
       (fastforce intro!: mono-onI limit-quantile-mono[OF G])+
  obtain y' where y': y' ∈ {max (y - δ) (y / 2)<..<y}
    y' ∉ {s ∈ {0<..}. ¬ isCont (limit-quantile G) s}
    using real-interval-avoid-countable-set[OF - cd, of max (y - δ) (y / 2) y] δ(1)
y
    by auto
  have y'1: 0 < y' y' < y |y' - y| < δ
    using y' y by auto
  have y'2: isCont (limit-quantile G) y'
    using y' y'1(1) by auto
  have y'3: limit-quantile G y - ε / 2 < limit-quantile G y'
    using δ(2)[of y'] y'1(2,3) by argo
  note L = seq-exp-quantile-tendsto[OF F G a wc y cont]
  note L' = seq-exp-quantile-tendsto[OF F G a wc y'1(1) y'2]
  have up: ∀F n in sequentially. tail-quantile F (real n * y)
                          ≤ ginv F (exp (- 1 / (real n * y)))
    using eventually-gt-at-top[of nat ⌈1 / y⌉]
  proof eventually-elim
    case (elim n)
    have n1: 1 < real n * y
    proof -
      have real (nat ⌈1 / y⌉) < real n
        using elim by simp
      moreover have 1 / y ≤ real (nat ⌈1 / y⌉)
        by (rule real-nat-ceiling-ge)
      ultimately have 1 / y < real n
        by simp
      thus ?thesis
        using y by (simp add: pos-divide-less-eq)
    qed
    have 1 - 1 / (real n * y) ≤ exp (- 1 / (real n * y))
      using exp-ge-add-one-self[of - 1 / (real n * y)] by simp
    thus ?case
      using n1
      by (intro ginv-mono[OF F.mono] F.ex-less-a F.ex-ge-b) auto
  qed
```

```
have low: ∀F n in sequentially. ginv F (exp (− 1 / (real n ∗ y′)))
                    ≤ tail-quantile F (real n ∗ y)
proof −
  have ev1: ∀F n in sequentially. y′ / y < exp (− 1 / (real n ∗ y′))
  proof (rule order-tendstoD(1))
    have (λn. − 1 / (real n ∗ y′)) ⟶ 0
      using y′1(1) tendsto-mult[OF tendsto-const[of − 1 / y′] lim-1-over-n]
      by (auto simp: field-simps intro: Lim-transform-eventually)
    thus (λn. exp (− 1 / (real n ∗ y′))) ⟶ 1
      using tendsto-exp[of - 0] by fastforce
    show y′ / y < 1
      using y′1(1,2) y by simp
  qed
  show ?thesis
    using ev1 eventually-gt-at-top[of 0]
  proof eventually-elim
    case (elim n)
    define s where s = 1 / (real n ∗ y′)
    have s-pos: 0 < s
      using elim y′1(1) by (auto simp: s-def)
    have ny-pos: 0 < real n ∗ y
      using elim y by auto
    have exp s ≥ 1 + s
      by (rule exp-ge-add-one-self)
    hence 1 ≥ (1 + s) ∗ exp (− s)
      using mult-right-mono[of 1 + s exp s exp (− s)]
      by (simp add: exp-minus field-simps)
    hence ∗: s ∗ exp (− s) ≤ 1 − exp (− s)
      by (simp add: algebra-simps)
    have 1 / (real n ∗ y) = (y′ / y) ∗ s
      using elim y′1(1) by (simp add: s-def field-simps)
    also have ... ≤ exp (− s) ∗ s
      using elim s-pos by (fastforce intro!: mult-right-mono simp: s-def)
    also have ... ≤ 1 − exp (− s)
      using ∗ by (simp add: mult.commute)
    finally have exp (− 1 / (real n ∗ y′)) ≤ 1 − 1 / (real n ∗ y)
      by (simp add: s-def)
    moreover have 0 < exp (− 1 / (real n ∗ y′))
      by simp
    moreover have 1 − 1 / (real n ∗ y) < 1
      using ny-pos by simp
    ultimately show ?case
      by(intro ginv-mono[OF F.mono] F.ex-ge-b F.ex-less-a) auto
  qed
qed

have evlow: ∀F n in sequentially. limit-quantile G y − ε
                        < (tail-quantile F (real n ∗ y) − b n) / a n
proof −
```

```
    have *:∀F n in sequentially. limit-quantile G y − ε
                                  < ((limit-quantile F (real n * y')) − b n) / a n
      using y'3 eps by (intro order-tendstoD(1)[OF L']) auto
    show ?thesis
      using a
      by(intro eventually-mono[OF eventually-conj[OF * low]])
        (smt (verit, ccfv-SIG) divide-minus-left divide-right-mono)
  qed
  have evup: ∀F n in sequentially. (tail-quantile F (real n * y) − b n) / a n
                           < limit-quantile G y + ε
  proof −
    have *:∀F n in sequentially. ((limit-quantile F (real n * y)) − b n) / a n
                         < limit-quantile G y + ε
      using eps by (intro order-tendstoD(2)[OF L]) auto
    show ?thesis
      using a
      by(intro eventually-mono[OF eventually-conj[OF * up]])
      (smt (verit, ccfv-SIG) divide-right-mono-neg minus-divide-divide minus-divide-left)
  qed
  from evlow evup
  have ∀F n in sequentially. dist ((tail-quantile F (real n * y) − b n) / a n)
                      (limit-quantile G y) < ε
    by eventually-elim (simp add: dist-real-def abs-diff-less-iff)
  thus ∃no. ∀n≥no. dist ((tail-quantile F (real n * y) − b n) / a n)
             (limit-quantile G y) < ε
    unfolding eventually-sequentially by blast
qed

lemma nondegenerate-growth-points:
  assumes G: right-continuous-mono G 0 1 and deg: ¬ degenerate-df G
    and N: countable N
  obtains y1 y2 where 0 < y1 y1 < y2 y1 ∉ N y2 ∉ N
    isCont (limit-quantile G) y1 isCont (limit-quantile G) y2
    limit-quantile G y1 < limit-quantile G y2
proof −
  interpret G: right-continuous-mono G 0 1
    by fact
  obtain x0 where x0: 0 < G x0 G x0 < 1
  proof −
    from deg obtain x0 where G x0 ≠ 0 G x0 ≠ 1
      by (auto simp: degenerate-df-def)
    thus thesis
      using G.a-le-f[of x0] G.f-le-b[of x0]
      by (intro that[of x0]) auto
  qed
  define p1 where p1 = G x0
  define p2 where p2 = (G x0 + 1) / 2
  have p12: 0 < p1 p1 < p2 p2 < 1
    using x0 by (auto simp: p1-def p2-def)
```

define *v1* **where** *v1 = − 1 / ln p1*
define *v2* **where** *v2 = − 1 / ln p2*
have *ln1*: *ln p1 < 0* **and** *ln2*: *ln p2 < 0*
using *p12* **by** *auto*
have *v1pos*: *0 < v1*
unfolding *v1-def* **using** *ln1* **by** *simp*
have *v2pos*: *0 < v2*
unfolding *v2-def* **using** *ln2* **by** *simp*
have *exp1*: *exp (− 1 / v1) = p1*
using *ln1 p12*(*1*) **by** (*simp add*: *v1-def*)
have *exp2*: *exp (− 1 / v2) = p2*
using *ln2 p12* **by** (*simp add*: *v2-def*)
have *v12*: *v1 < v2*
proof −
have ∗: *ln p1 < ln p2*
using *p12* **by** *simp*
have *v1 = 1 / (− ln p1) v2 = 1 / (− ln p2)*
by (*simp-all add*: *v1-def v2-def*)
moreover have *1 / (− ln p1) < 1 / (− ln p2)*
using ∗ *ln1 ln2* **by** (*intro divide-strict-left-mono*)
(*auto simp*: *zero-less-mult-iff*)
ultimately show *?thesis*
by *simp*
qed
have *Dv1*: *limit-quantile G v1 ≤ x0*
using *G.ginv-le-iff*[*of p1 x0*] *p12 exp1*
by (*simp add*: *p1-def*)
have *Dv2*: *x0 < limit-quantile G v2*
proof −
have ¬ *ginv G p2 ≤ x0*
using *G.ginv-le-iff*[*of p2 x0*] *p12*
by (*simp add*: *p1-def*[*symmetric*])
thus *?thesis*
using *exp2* **by** *simp*
qed
have *cd*: *countable* (*N ∪* {*s ∈* {*0*<..}. ¬ *isCont* (*limit-quantile G*) *s*})
using *N*
by (*intro countable-Un mono-on-ctble-discont-open*)
(*fastforce intro*!: *mono-onI limit-quantile-mono*[*OF G*])+
obtain *y1* **where** *y1*: *y1 ∈* {*v1 / 2*<..<*v1*}
y1 ∉ N ∪ {*s ∈* {*0*<..}. ¬ *isCont* (*limit-quantile G*) *s*}
using *real-interval-avoid-countable-set*[*OF - cd*, *of v1 / 2 v1*] *v1pos* **by** *auto*
obtain *y2* **where** *y2*: *y2 ∈* {*v2*<..<*v2 + 1*}
y2 ∉ N ∪ {*s ∈* {*0*<..}. ¬ *isCont* (*limit-quantile G*) *s*}
using *real-interval-avoid-countable-set*[*OF - cd*, *of v2 v2 + 1*] **by** *auto*
have *y1f*: *0 < y1 y1 < v1*
using *y1*(*1*) *v1pos* **by** *auto*
have *y2f*: *0 < y2 v2 < y2*
using *y2*(*1*) *v2pos* **by** *auto*

```
  show ?thesis
  proof (rule that[of y1 y2])
    show 0 < y1
      by fact
    show y1 < y2
      using y1f y2f v12 by simp
    show y1 ∉ N y2 ∉ N
      using y1(2) y2(2) by auto
    show isCont (limit-quantile G) y1 isCont (limit-quantile G) y2
      using y1 y2 y1f(1) y2f(1) by auto
    have limit-quantile G y1 ≤ limit-quantile G v1
      using y1f by (intro limit-quantile-mono[OF G]) auto
    moreover have limit-quantile G v2 ≤ limit-quantile G y2
      using y2f v2pos by (intro limit-quantile-mono[OF G]) auto
    ultimately show limit-quantile G y1 < limit-quantile G y2
      using Dv1 Dv2 by simp
  qed
qed

lemma
  fixes F G :: real ⇒ real and a b :: nat ⇒ real
  assumes F:right-continuous-mono F 0 1 and G:right-continuous-mono G 0 1
    and deg:¬ degenerate-df G
    and a:⋀n. 0 < a n
    and wc:weak-conv (λn x. F (a n * x + b n) ^ n) G
  shows normalizing-ratio1:(λn. a (Suc n) / a n) ⟶ 1
    and normalizing-ratio2:(λn. (b (Suc n) - b n) / a n) ⟶ 0
proof -
  have er:⋀c. ∀F n in sequentially. c ≤ real n
    using filterlim-real-sequentially unfolding filterlim-at-top by blast
  obtain y1 y2 where y12: 0 < y1 y1 < y2
    and cont12: isCont (limit-quantile G) y1 isCont (limit-quantile G) y2
    and D12: limit-quantile G y1 < limit-quantile G y2
    by (rule nondegenerate-growth-points[OF G deg countable-empty]) auto
  have y2pos: 0 < y2
    using y12 by simp
  have base: (λn. (tail-quantile F (real n * z) - b n) / a n) ⟶ limit-quantile
G z
    if z: 0 < z isCont (limit-quantile G) z for z
    by (rule seq-tail-quantile-tendsto[OF F G deg a wc z])
  have shifted: (λn. (tail-quantile F (real n * z) - b (Suc n)) / a (Suc n))
                ⟶ limit-quantile G z
    if z: 0 < z isCont (limit-quantile G) z for z
  proof (rule squeeze-tendsto-cont-points-pos[where
      gr = λy' n. (tail-quantile F (real (Suc n) * y') - b (Suc n)) / a (Suc n)
      and f = limit-quantile G, OF - z])
    show mono-on {0<..} (limit-quantile G)
      by (auto intro!: mono-onI limit-quantile-mono[OF G])
  next
```

fix y' :: *real*
assume y': $0 < y'$ *isCont* (*limit-quantile G*) y'
show (λn. (*tail-quantile F* (*real* (*Suc n*) $*$ y') $-$ *b* (*Suc n*)) / *a* (*Suc n*))
$\longrightarrow$ *limit-quantile G* y'
by (*intro LIMSEQ-Suc base* y')
next
fix y' :: *real*
assume y': $0 < y'$ $y' < z$
show $\forall_F$ *n in sequentially*. (*tail-quantile F* (*real* (*Suc n*) $*$ y') $-$ *b* (*Suc n*)) / *a* (*Suc n*)
$\leq$ (*tail-quantile F* (*real n* $*$ *z*) $-$ *b* (*Suc n*)) / *a* (*Suc n*)
using *er*[*of max* (y' / ($z - y'$)) (*1* / y')]
proof *eventually-elim*
case *h*:(*elim n*)
have *1*: *1* $<$ *real* (*Suc n*) $*$ y'
proof $-$
have *1* = (*1* / y') $*$ y'
using y'(*1*) **by** *simp*
also have ... $\leq$ *real n* $*$ y'
using *h* y'(*1*) **by** (*intro mult-right-mono*) *auto*
also have ... $<$ *real* (*Suc n*) $*$ y'
using y'(*1*) **by** *simp*
finally show *?thesis* **.**
qed
have *2*: *real* (*Suc n*) $*$ $y' \leq$ *real n* $*$ *z*
proof $-$
have y' / ($z - y'$) $\leq$ *real n*
using *h* **by** *simp*
hence $y' \leq$ *real n* $*$ ($z - y'$)
using y' **by** (*simp add*: *field-simps*)
thus *?thesis*
by (*simp add*: *algebra-simps*)
qed
have *tail-quantile F* (*real* (*Suc n*) $*$ y') $\leq$ *tail-quantile F* (*real n* $*$ *z*)
by (*rule tail-quantile-mono*[*OF F 1 2*])
thus *?case*
using *a*[*of Suc n*] **by** (*auto intro*!: *divide-right-mono*)
qed
next
fix y' :: *real*
assume y': $z < y'$
show $\forall_F$ *n in sequentially*. (*tail-quantile F* (*real n* $*$ *z*) $-$ *b* (*Suc n*)) / *a* (*Suc n*)
$\leq$ (*tail-quantile F* (*real* (*Suc n*) $*$ y') $-$ *b* (*Suc n*)) / *a* (*Suc n*)
using *er*[*of 1* / *z* + *1*]
proof *eventually-elim*
case (*elim n*)
have *zpos*: *0* $<$ *z*
using *z* **by** *simp*

```
    have 1: 1 < real n * z
    proof −
      have 1 < (1 / z + 1) * z
        using zpos by (simp add: field-simps)
      also have ... ≤ real n * z
        using elim zpos by (intro mult-right-mono) auto
      finally show ?thesis .
    qed
    have 2: real n * z ≤ real (Suc n) * y'
      using y' zpos by (intro mult-mono) auto
    have tail-quantile F (real n * z) ≤ tail-quantile F (real (Suc n) * y')
      by (rule tail-quantile-mono[OF F 1 2])
    thus ?case
      using a[of Suc n] by (auto intro!: divide-right-mono)
  qed
 qed
 note B1 = base[OF y12(1) cont12(1)] and B2 = base[OF y2pos cont12(2)]
 note S1 = shifted[OF y12(1) cont12(1)] and S2 = shifted[OF y2pos cont12(2)]
 define D where D = limit-quantile G y2 − limit-quantile G y1
 have D-pos: 0 < D
   using D12 by (simp add: D-def)
 have d1: (λn. (tail-quantile F (real n * y2) − tail-quantile F (real n * y1)) / a
n)
          ⟶ D
   using tendsto-diff[OF B2 B1]
   by (simp add: diff-divide-distrib D-def)
 have d2: (λn. (tail-quantile F (real n * y2) − tail-quantile F (real n * y1)) / a
(Suc n))
          ⟶ D
   using tendsto-diff[OF S2 S1]
   by (simp add: diff-divide-distrib D-def)
 have ev-pos: ∀F n in sequentially. 0 < (tail-quantile F (real n * y2)
                        − tail-quantile F (real n * y1)) / a n
   using order-tendstoD(1)[OF d1 D-pos] .
 show r1: (λn. a (Suc n) / a n) ⟶ 1
 proof −
   have (λn. ((tail-quantile F (real n * y2) − tail-quantile F (real n * y1)) / a n)
         / ((tail-quantile F (real n * y2) − tail-quantile F (real n * y1)) / a (Suc
n)))
         ⟶ D / D
     using D-pos by (intro tendsto-divide d1 d2) auto
   hence ∗: (λn. ((tail-quantile F (real n * y2) − tail-quantile F (real n * y1)) /
a n)
         / ((tail-quantile F (real n * y2) − tail-quantile F (real n * y1)) / a (Suc
n)))
         ⟶ 1
     using D-pos by simp
   have ∀F n in sequentially. ((tail-quantile F (real n * y2)
                     − tail-quantile F (real n * y1)) / a n)
```

```
                    / ((tail-quantile F (real n * y2)
                        - tail-quantile F (real n * y1)) / a (Suc n))
                    = a (Suc n) / a n
      using ev-pos
    proof eventually-elim
      case (elim n)
      have 0 < tail-quantile F (real n * y2) - tail-quantile F (real n * y1)
        using elim a[of n] by (simp add: zero-less-divide-iff)
      thus ?case
        using a[of n] a[of Suc n] by (auto simp: field-simps)
    qed
    from Lim-transform-eventually[OF * this]
    show ?thesis .
  qed
  have eq: (tail-quantile F (real n * y1) - b n) / a n
          - ((tail-quantile F (real n * y1) - b (Suc n)) / a (Suc n)) * (a (Suc n)
/ a n)
          = (b (Suc n) - b n) / a n for n
    using  a[of n] a[of Suc n] by (simp add: field-simps)
  have (λn. (tail-quantile F (real n * y1) - b n) / a n
          - ((tail-quantile F (real n * y1) - b (Suc n)) / a (Suc n)) * (a (Suc n)
/ a n))
          ⟶ limit-quantile G y1 - limit-quantile G y1 * 1
    by (intro tendsto-diff tendsto-mult B1 S1 r1)
  thus (λn. (b (Suc n) - b n) / a n) ⟶ 0
    unfolding eq by simp
qed

lemma real-tail-quantile-tendsto:
  fixes F G :: real ⇒ real and a b :: nat ⇒ real and y :: real
  assumes F:right-continuous-mono F 0 1 and G:right-continuous-mono G 0 1
    and deg:¬ degenerate-df G
    and a:⋀n. 0 < a n
    and wc:weak-conv (λn x. F (a n * x + b n) ^ n) G
    and y:0 < y and cont:isCont (limit-quantile G) y
  shows ((λt. (tail-quantile F (t * y) - b (nat ⌊t⌋)) / a (nat ⌊t⌋))
         ⟶ limit-quantile G y) at-top
proof -
  note nf = filterlim-compose[OF filterlim-nat-sequentially filterlim-floor-sequentially]
  note nf-real = filterlim-compose[OF filterlim-real-sequentially nf]
  have ev-nf: ∀F t::real in at-top. c ≤ real (nat ⌊t⌋) for c
    using nf-real unfolding filterlim-at-top by blast
  show ?thesis
  proof (rule squeeze-tendsto-cont-points-pos[where
      gr = λy' t. (tail-quantile F (real (nat ⌊t⌋) * y') - b (nat ⌊t⌋)) / a (nat ⌊t⌋)
      and f = limit-quantile G, OF - y cont])
    show mono-on {0<..} (limit-quantile G)
      by (auto intro!: mono-onI limit-quantile-mono[OF G])
  next
```

```
    fix y' :: real
    assume y': 0 < y' isCont (limit-quantile G) y'
    show ((λt. (tail-quantile F (real (nat ⌊t⌋) * y') - b (nat ⌊t⌋)) / a (nat ⌊t⌋))
          ⟶ limit-quantile G y') at-top
      using filterlim-compose[OF seq-tail-quantile-tendsto[OF F G deg a wc y'] nf]
      by simp
  next
    fix y' :: real
    assume y'1: 0 < y' and y'2: y' < y
    show ∀F t in at-top. (tail-quantile F (real (nat ⌊t⌋) * y') - b (nat ⌊t⌋)) / a
(nat ⌊t⌋)
                ≤ (tail-quantile F (t * y) - b (nat ⌊t⌋)) / a (nat ⌊t⌋)
      using ev-nf[of 1 / y' + 1] eventually-ge-at-top[of 0]
    proof eventually-elim
      case (elim t)
      have 1: 1 < real (nat ⌊t⌋) * y'
      proof -
        have 1 < (1 / y' + 1) * y'
          using y'1 by (simp add: field-simps)
        also have ... ≤ real (nat ⌊t⌋) * y'
          using elim y'1 by (intro mult-right-mono) auto
        finally show ?thesis .
      qed
      have real (nat ⌊t⌋) ≤ t
        using elim by linarith
      hence 2: real (nat ⌊t⌋) * y' ≤ t * y
        using y'1 y'2 y elim by (intro mult-mono) auto
      have tail-quantile F (real (nat ⌊t⌋) * y') ≤ tail-quantile F (t * y)
        by (rule tail-quantile-mono[OF F 1 2])
      thus ?case
        using a[of nat ⌊t⌋] by (auto intro!: divide-right-mono)
    qed
  next
    fix y' :: real
    assume y': y < y'
    show ∀F t in at-top. (tail-quantile F (t * y) - b (nat ⌊t⌋)) / a (nat ⌊t⌋)
                ≤ (tail-quantile F (real (nat ⌊t⌋) * y') - b (nat ⌊t⌋)) / a (nat ⌊t⌋)
      using ev-nf[of y / (y' - y)] eventually-gt-at-top[of 1 / y]
    proof eventually-elim
      case (elim t)
      have t-pos: 0 < t
        using elim(2) y zero-less-divide-1-iff[of y] by linarith
      have 1: 1 < t * y
        using elim(2) y by (simp add: field-simps)
      have 2: t * y ≤ real (nat ⌊t⌋) * y'
      proof -
        have t ≤ real (nat ⌊t⌋) + 1
          using t-pos by linarith
        hence t * y ≤ (real (nat ⌊t⌋) + 1) * y
```

```
      using y by (intro mult-right-mono) auto
    also have ... ≤ real (nat ⌊t⌋) * y'
      using elim(1) y' y by (simp add: field-simps)
    finally show ?thesis .
  qed
  show ?case
    using a[of nat ⌊t⌋] tail-quantile-mono[OF F 1 2]
    by (auto intro!: divide-right-mono)
 qed
 qed
qed

lemma scaling-limits-ex:
 fixes F G :: real ⇒ real and a b :: nat ⇒ real
 assumes F:right-continuous-mono F 0 1 and G:right-continuous-mono G 0 1
  and deg: ¬ degenerate-df G
  and a:⋀n. 0 < a n
  and wc:weak-conv (λn x. F (a n * x + b n) ^ n) G
 obtains A B where
  ⋀y. 0 < y ⟹ 0 < A y
  and ⋀y. 0 < y ⟹ ((λt. a (nat ⌊t * y⌋) / a (nat ⌊t⌋)) ⟶ A y) at-top
  and ⋀y. 0 < y ⟹
       ((λt. (b (nat ⌊t * y⌋) - b (nat ⌊t⌋)) / a (nat ⌊t⌋)) ⟶ B y) at-top
  and A ∈ borel-measurable (restrict-space borel {0<..})
  and ⋀x y. 0 < x ⟹ 0 < y ⟹ A (x * y) = A x * A y
  and ⋀x y. 0 < x ⟹ 0 < y ⟹ isCont (limit-quantile G) x
       ⟹ isCont (limit-quantile G) (x * y)
       ⟹ limit-quantile G (x * y) = limit-quantile G x * A y + B y
proof -
 have RT: ((λt. (tail-quantile F (t * s) - b (nat ⌊t⌋)) / a (nat ⌊t⌋))
          ⟶ limit-quantile G s) at-top
   if 0 < s isCont (limit-quantile G) s for s
   by (rule real-tail-quantile-tendsto[OF F G deg a wc that])
 have mult-top: filterlim (λt::real. t * y) at-top at-top if 0 < y for y :: real
   using filterlim-tendsto-pos-mult-at-top[OF tendsto-const that filterlim-ident]
   by (simp add: mult.commute)
 have RTy: ((λt. (tail-quantile F (t * y * s) - b (nat ⌊t * y⌋)) / a (nat ⌊t * y⌋))
          ⟶ limit-quantile G s) at-top
   if y: 0 < y and s: 0 < s isCont (limit-quantile G) s for y s
   by (rule filterlim-compose[OF RT[OF s] mult-top[OF y]])
 have ex: ∃L M. 0 ≤ L ∧
          ((λt. a (nat ⌊t * y⌋) / a (nat ⌊t⌋)) ⟶ L) at-top ∧
          ((λt. (b (nat ⌊t * y⌋) - b (nat ⌊t⌋)) / a (nat ⌊t⌋)) ⟶ M) at-top
   if y: 0 < y for y :: real
 proof -
   have cN: countable {s ∈ {0<..}. ¬ isCont (limit-quantile G) (y * s)}
   proof -
     have *: countable {u ∈ {0<..}. ¬ isCont (limit-quantile G) u}
       by (intro mono-on-ctble-discont-open)
```

```
      (fastforce intro!: mono-onI limit-quantile-mono[OF G])+
    have sub: {s ∈ {0<..}. ¬ isCont (limit-quantile G) (y * s)}
          ⊆ (λu. u / y) ` {u ∈ {0<..}. ¬ isCont (limit-quantile G) u}
    proof
      fix s
      assume s: s ∈ {s ∈ {0<..}. ¬ isCont (limit-quantile G) (y * s)}
      hence y * s ∈ {u ∈ {0<..}. ¬ isCont (limit-quantile G) u}
        using y by auto
      moreover have s = (y * s) / y
        using y by simp
      ultimately show s ∈ (λu. u / y) ` {u ∈ {0<..}. ¬ isCont (limit-quantile
G) u}
        by blast
    qed
    show ?thesis
      by (rule countable-subset[OF sub countable-image[OF *]])
  qed
  obtain s1 s2 where s12: 0 < s1 s1 < s2
    and nn: s1 ∉ {s ∈ {0<..}. ¬ isCont (limit-quantile G) (y * s)}
      s2 ∉ {s ∈ {0<..}. ¬ isCont (limit-quantile G) (y * s)}
    and cs: isCont (limit-quantile G) s1 isCont (limit-quantile G) s2
    and Ds: limit-quantile G s1 < limit-quantile G s2
    by (rule nondegenerate-growth-points[OF G deg cN])
  have s2pos: 0 < s2
    using s12 by simp
  have cys: isCont (limit-quantile G) (y * s1) isCont (limit-quantile G) (y * s2)
    using nn s12 s2pos by auto
  have ys-pos: 0 < y * s1 0 < y * s2
    using y s12 s2pos by auto
  have P1: ((λt. (tail-quantile F (t * y * s1) − b (nat ⌊t⌋)) / a (nat ⌊t⌋))
          ⟶ limit-quantile G (y * s1)) at-top
    using RT[OF ys-pos(1) cys(1)] by (simp add: mult.assoc)
  have P2: ((λt. (tail-quantile F (t * y * s2) − b (nat ⌊t⌋)) / a (nat ⌊t⌋))
          ⟶ limit-quantile G (y * s2)) at-top
    using RT[OF ys-pos(2) cys(2)] by (simp add: mult.assoc)
  note Q1 = RTy[OF y s12(1) cs(1)] and Q2 = RTy[OF y s2pos cs(2)]
  define d where d = limit-quantile G s2 − limit-quantile G s1
  have d-pos: 0 < d
    using Ds by (simp add: d-def)
  have Qdiff: ((λt. (tail-quantile F (t * y * s2) − tail-quantile F (t * y * s1))
          / a (nat ⌊t * y⌋)) ⟶ d) at-top
    using tendsto-diff[OF Q2 Q1] by (simp add: diff-divide-distrib d-def)
  have Pdiff: ((λt. (tail-quantile F (t * y * s2) − tail-quantile F (t * y * s1))
          / a (nat ⌊t⌋))
        ⟶ limit-quantile G (y * s2) − limit-quantile G (y * s1)) at-top
    using tendsto-diff[OF P2 P1] by (simp add: diff-divide-distrib)
  have ev-pos: ∀F t in at-top. 0 < (tail-quantile F (t * y * s2)
                − tail-quantile F (t * y * s1)) / a (nat ⌊t * y⌋)
    using order-tendstoD(1)[OF Qdiff d-pos] .
```

```
  define L where L = (limit-quantile G (y * s2) - limit-quantile G (y * s1)) /
d
  have Alim: ((λt. a (nat ⌊t * y⌋) / a (nat ⌊t⌋)) ⟶ L) at-top
  proof -
    have *: ((λt. ((tail-quantile F (t * y * s2) - tail-quantile F (t * y * s1))
                / a (nat ⌊t⌋))
              / ((tail-quantile F (t * y * s2) - tail-quantile F (t * y * s1))
                / a (nat ⌊t * y⌋))) ⟶ L) at-top
      unfolding L-def using d-pos by (intro tendsto-divide Pdiff Qdiff) auto
    have ∀F t in at-top. ((tail-quantile F (t * y * s2) - tail-quantile F (t * y *
s1))
                        / a (nat ⌊t⌋))
                      / ((tail-quantile F (t * y * s2) - tail-quantile F (t * y * s1))
                        / a (nat ⌊t * y⌋)) = a (nat ⌊t * y⌋) / a (nat ⌊t⌋)
      using ev-pos
    proof eventually-elim
      case (elim t)
      have 0 < tail-quantile F (t * y * s2) - tail-quantile F (t * y * s1)
        using elim a[of nat ⌊t * y⌋] by (simp add: zero-less-divide-iff)
      thus ?case
        using a[of nat ⌊t⌋] a[of nat ⌊t * y⌋] by (auto simp: field-simps)
    qed
    from Lim-transform-eventually[OF * this]
    show ?thesis .
  qed
  have L-nonneg: 0 ≤ L
    using a
    by(auto intro!: tendsto-lowerbound[OF Alim] always-eventually less-imp-le[of
- - / -])
  have Blim: ((λt. (b (nat ⌊t * y⌋) - b (nat ⌊t⌋)) / a (nat ⌊t⌋))
              ⟶ limit-quantile G (y * s1) - limit-quantile G s1 * L) at-top
  proof -
    have eq: (tail-quantile F (t * y * s1) - b (nat ⌊t⌋)) / a (nat ⌊t⌋)
        - ((tail-quantile F (t * y * s1) - b (nat ⌊t * y⌋)) / a (nat ⌊t * y⌋))
          * (a (nat ⌊t * y⌋) / a (nat ⌊t⌋))
        = (b (nat ⌊t * y⌋) - b (nat ⌊t⌋)) / a (nat ⌊t⌋) for t
      using a[of nat ⌊t⌋] a[of nat ⌊t * y⌋] by (simp add: field-simps)
    have ((λt. (tail-quantile F (t * y * s1) - b (nat ⌊t⌋)) / a (nat ⌊t⌋)
          - ((tail-quantile F (t * y * s1) - b (nat ⌊t * y⌋)) / a (nat ⌊t * y⌋))
            * (a (nat ⌊t * y⌋) / a (nat ⌊t⌋)))
          ⟶ limit-quantile G (y * s1) - limit-quantile G s1 * L) at-top
      by (intro tendsto-diff tendsto-mult P1 Q1 Alim)
    thus ?thesis
      unfolding eq .
  qed
  show ?thesis
    using L-nonneg Alim Blim by blast
qed
define A where A = (λy::real. Lim at-top (λt. a (nat ⌊t * y⌋) / a (nat ⌊t⌋)))
```

```
  define B where B = (λy::real. Lim at-top (λt. (b (nat ⌊t * y⌋) − b (nat ⌊t⌋))
/ a (nat ⌊t⌋)))
  have AB: 0 ≤ A y ∧
      ((λt. a (nat ⌊t * y⌋) / a (nat ⌊t⌋)) ⟶ A y) at-top ∧
      ((λt. (b (nat ⌊t * y⌋) − b (nat ⌊t⌋)) / a (nat ⌊t⌋)) ⟶ B y) at-top
    if y: 0 < y for y
  proof −
    obtain L M where LM: 0 ≤ L
      ((λt. a (nat ⌊t * y⌋) / a (nat ⌊t⌋)) ⟶ L) at-top
      ((λt. (b (nat ⌊t * y⌋) − b (nat ⌊t⌋)) / a (nat ⌊t⌋)) ⟶ M) at-top
      using ex[OF y] by blast
    have A y = L
      unfolding A-def by (intro tendsto-Lim LM(2)) simp
    moreover have B y = M
      unfolding B-def by (intro tendsto-Lim LM(3)) simp
    ultimately show ?thesis
      using LM by simp
  qed
  have Anonneg: 0 ≤ A y if 0 < y for y
    using AB[OF that] by blast
  have Atend: ((λt. a (nat ⌊t * y⌋) / a (nat ⌊t⌋)) ⟶ A y) at-top if 0 < y for y
    using AB[OF that] by blast
  have Btend: ((λt. (b (nat ⌊t * y⌋) − b (nat ⌊t⌋)) / a (nat ⌊t⌋)) ⟶ B y) at-top
    if 0 < y for y
    using AB[OF that] by blast
  have Amul: A (x * y) = A x * A y if x: 0 < x and y: 0 < y for x y
  proof −
    have 1: ((λt. a (nat ⌊t * (x * y)⌋) / a (nat ⌊t⌋)) ⟶ A (x * y)) at-top
      using x y by (intro Atend) auto
    have ((λt. (a (nat ⌊t * x * y⌋) / a (nat ⌊t * x⌋))
          * (a (nat ⌊t * x⌋) / a (nat ⌊t⌋))) ⟶ A y * A x) at-top
     by (intro tendsto-mult Atend[OF x] filterlim-compose[OF Atend[OF y] mult-top[OF
x]])
    moreover have (a (nat ⌊t * x * y⌋) / a (nat ⌊t * x⌋))
          * (a (nat ⌊t * x⌋) / a (nat ⌊t⌋)) = a (nat ⌊t * x * y⌋) / a (nat ⌊t⌋)
      for t
      using a[of nat ⌊t * x⌋] by (auto simp: field-simps)
    ultimately have ((λt::real. a (nat ⌊t * x * y⌋) / a (nat ⌊t⌋)) ⟶ A y * A
x) at-top
      by (metis (no-types, lifting) ext)
    hence ((λt. a (nat ⌊t * (x * y)⌋) / a (nat ⌊t⌋)) ⟶ A y * A x) at-top
      by (simp add: mult.assoc)
    from tendsto-unique[OF - 1 this]
    show ?thesis
      by (simp add: mult.commute)
  qed
  have A1: A 1 = 1
  proof −
    have (λt::real. a (nat ⌊t * 1⌋) / a (nat ⌊t⌋)) = (λt. 1)
```

```
    using a
    by (metis divide-self-if less-irrefl mult.comm-neutral)
  hence ((λt::real. a (nat ⌊t * 1⌋) / a (nat ⌊t⌋)) ⟶ 1) at-top
    by simp
  from tendsto-unique[OF - Atend[of 1] this]
  show ?thesis
    by simp
qed
have Apos: 0 < A y if y: 0 < y for y
proof −
  have A (y * (1 / y)) = A y * A (1 / y)
    using y by (intro Amul) auto
  moreover have y * (1 / y) = 1
    using y by simp
  ultimately have A y * A (1 / y) = 1
    using A1 by simp
  hence A y ≠ 0
    by auto
  with Anonneg[OF y] show ?thesis
    by simp
qed
have Deq: limit-quantile G (x * y) = limit-quantile G x * A y + B y
  if x: 0 < x and y: 0 < y
    and cx: isCont (limit-quantile G) x and cxy: isCont (limit-quantile G) (x *
y)
  for x y
proof −
  have xy: 0 < x * y
    using x y by simp
  have eq: ((tail-quantile F (t * y * x) − b (nat ⌊t * y⌋)) / a (nat ⌊t * y⌋))
        * (a (nat ⌊t * y⌋) / a (nat ⌊t⌋))
        + (b (nat ⌊t * y⌋) − b (nat ⌊t⌋)) / a (nat ⌊t⌋)
        = (tail-quantile F (t * y * x) − b (nat ⌊t⌋)) / a (nat ⌊t⌋) for t
    using a[of nat ⌊t⌋] a[of nat ⌊t * y⌋] by (simp add: field-simps)
  have ((λt. ((tail-quantile F (t * y * x) − b (nat ⌊t * y⌋)) / a (nat ⌊t * y⌋))
         * (a (nat ⌊t * y⌋) / a (nat ⌊t⌋))
         + (b (nat ⌊t * y⌋) − b (nat ⌊t⌋)) / a (nat ⌊t⌋))
         ⟶ limit-quantile G x * A y + B y) at-top
   by (intro tendsto-add tendsto-mult RTy[OF y x cx] Atend[OF y] Btend[OF y])
  hence ((λt. (tail-quantile F (t * (x * y)) − b (nat ⌊t⌋)) / a (nat ⌊t⌋))
        ⟶ limit-quantile G x * A y + B y) at-top
    unfolding eq by (simp add: ac-simps)
  from tendsto-unique[OF - RT[OF xy cxy] this]
  show ?thesis
    by simp
qed
have Ameas: A ∈ borel-measurable (restrict-space borel {0<..})
proof (rule borel-measurable-LIMSEQ-real)
  fix i :: nat
```

show (λy::*real*. *a* (*nat* $\lfloor$*real i* $*$ *y*$\rfloor$) / *a* (*nat* $\lfloor$*real i*$\rfloor$))
∈ *borel-measurable* (*restrict-space borel* {*0*<..})
by(*auto intro*!: *measurable-restrict-space1*)
next
fix *y*::*real*
assume *y* ∈ *space* (*restrict-space borel* {*0*<..})
hence *y*: *0* < *y*
by (*simp add*: *space-restrict-space*)
have ((λ*i*. *a* (*nat* $\lfloor$*real i* $*$ *y*$\rfloor$) / *a* (*nat* $\lfloor$*real i*$\rfloor$)) ⟶ *A y*) *sequentially*
by (*rule filterlim-compose*[*OF Atend*[*OF y*] *filterlim-real-sequentially*])
thus (λ*i*. *a* (*nat* $\lfloor$*real i* $*$ *y*$\rfloor$) / *a* (*nat* $\lfloor$*real i*$\rfloor$)) ⟶ *A y* **.**
qed
show *?thesis*
by (*rule that*[*of A B*]) (*fact Apos Atend Btend Ameas Amul Deq*)+
qed

theorem *limit-quantile-form*:
fixes *F G* :: *real* ⇒ *real* **and** *a b* :: *nat* ⇒ *real*
assumes *F*: *right-continuous-mono F 0 1* **and** *G*: *right-continuous-mono G 0 1*
and *deg*: ¬ *degenerate-df G*
and *a*: ⋀*n*. *0* < *a n*
and *wc*: *weak-conv* (λ*n x*. *F* (*a n* $*$ *x* + *b n*) ^ *n*) *G*
obtains γ *c d* **where** *0* < *c*
and ⋀*y*. *0* < *y* ⟹ *limit-quantile G y*
= (*if* γ = *0 then c* $*$ *ln y* + *d else c* $*$ (*y powr* γ − *1*) / γ + *d*)
proof −
obtain *A B* **where** *Apos*: ⋀*y*. *0* < *y* ⟹ *0* < *A y*
and *Atend*: ⋀*y*. *0* < *y* ⟹ ((λ*t*. *a* (*nat* $\lfloor$*t* $*$ *y*$\rfloor$) / *a* (*nat* $\lfloor$*t*$\rfloor$)) ⟶ *A y*) *at-top*
and *Btend*: ⋀*y*. *0* < *y* ⟹
((λ*t*. (*b* (*nat* $\lfloor$*t* $*$ *y*$\rfloor$) − *b* (*nat* $\lfloor$*t*$\rfloor$)) / *a* (*nat* $\lfloor$*t*$\rfloor$)) ⟶ *B y*) *at-top*
and *Ameas*: *A* ∈ *borel-measurable* (*restrict-space borel* {*0*<..})
and *Amul*: ⋀*x y*. *0* < *x* ⟹ *0* < *y* ⟹ *A* (*x* $*$ *y*) = *A x* $*$ *A y*
and *Deq*: ⋀*x y*. *0* < *x* ⟹ *0* < *y* ⟹ *isCont* (*limit-quantile G*) *x*
⟹ *isCont* (*limit-quantile G*) (*x* $*$ *y*)
⟹ *limit-quantile G* (*x* $*$ *y*) = *limit-quantile G x* $*$ *A y* + *B y*
by (*rule scaling-limits-ex*[*OF F G deg a wc*]) *auto*
obtain γ **where** *Aform*: ⋀*x*. *0* < *x* ⟹ *A x* = *x powr* γ
by (*rule multiplicative-measurable-powr*[**where** *A*=*A*,*OF Ameas Apos Amul*]) *auto*
have *Nx-ctble*: *countable* {*s* ∈ {*0*<..}. ¬ *isCont* (*limit-quantile G*) *s*}
by (*intro mono-on-ctble-discont-open*)
(*fastforce intro*!: *mono-onI limit-quantile-mono*[*OF G*])+
have *DeqFull*: *limit-quantile G* (*x* $*$ *y*) = *limit-quantile G x* $*$ *A y* + *B y*
if *x*: *0* < *x* **and** *y*: *0* < *y* **for** *x y*
proof −
define *Nx* **where** *Nx* = {*s* ∈ {*0*<..}. ¬ *isCont* (*limit-quantile G*) *s*}
have *Nx-ctble′*: *countable Nx*
unfolding *Nx-def* **by** (*rule Nx-ctble*)

```
    define N where N = Nx ∪ (λs. s / y) ‘ Nx
    have N-ctble: countable N
      unfolding N-def using Nx-ctble' by (intro countable-Un countable-image)
auto
    define f where f = (λx. limit-quantile G (x * y))
    define h where h = (λx. limit-quantile G x * A y + B y)
    have eq0: f x' = h x' if x'pos: 0 < x' and x'N: x' ∉ N for x'
    proof −
      have cx': isCont (limit-quantile G) x'
        using x'N x'pos by (auto simp: N-def Nx-def)
      have cxy': isCont (limit-quantile G) (x' * y)
      proof (rule ccontr)
        assume ¬ isCont (limit-quantile G) (x' * y)
        hence (x' * y) / y ∈ (λs. s / y) ‘ Nx
          using x'pos y unfolding Nx-def by fastforce
        hence x' ∈ N
          using y by (auto simp add: N-def)
        with x'N show False
          by simp
      qed
      show ?thesis
        unfolding f-def h-def using Deq[OF x'pos y cx' cxy'] .
    qed
    have left-cont-f: (f ⟶ f x) (at-left x)
      unfolding f-def using limit-quantile-left-cont-scale[OF G y x] .
    have left-cont-h: (h ⟶ h x) (at-left x)
      unfolding h-def using limit-quantile-left-cont[OF G x] by (intro tend-
sto-intros)
    have f x = h x
      by (rule left-cont-agree-cocountable-eq[OF N-ctble x left-cont-f left-cont-h])
        (use eq0 in blast)
    thus ?thesis
      by (simp add: f-def h-def)
  qed

  define d where d = limit-quantile G 1
  have Bform: B y = limit-quantile G y − d * A y if y: 0 < y for y
    using DeqFull[OF zero-less-one y] by (simp add: d-def)
  define H where H = (λu. limit-quantile G (exp u) − d)
  have Acomp: A (exp v) = exp (γ * v) for v
    by(simp add: Aform powr-def)
  have H-eq: H (u + v) = H u * exp (γ * v) + H v for u v
  proof −
    have limit-quantile G (exp (u + v)) = limit-quantile G (exp u * exp v)
      by (simp add: exp-add)
    also have ... = limit-quantile G (exp u) * A (exp v) + B (exp v)
      by (rule DeqFull[OF exp-gt-zero exp-gt-zero])
    also have ... = limit-quantile G (exp u) * exp (γ * v)
                  + (limit-quantile G (exp v) − d * exp (γ * v))
```

```
    using Bform[OF exp-gt-zero] Acomp by simp
  also have ... = H u * exp (γ * v) + limit-quantile G (exp v)
    by (simp add: H-def algebra-simps)
  finally have limit-quantile G (exp (u + v)) = H u * exp (γ * v) + limit-quantile
G (exp v) .
  thus ?thesis
    by (simp add: H-def)
 qed
 have H-mono: mono H
  by(auto intro!: monoI limit-quantile-mono G simp: H-def)
 obtain c where c-nonneg: 0 ≤ c
  and Hform: ⋀u. H u = (if γ = 0 then c * u else c * (exp (γ * u) − 1) / γ)
  by (rule cocycle-mono-solution[OF H-mono H-eq]) simp
 have Dform: limit-quantile G y
    = (if γ = 0 then c * ln y + d else c * (y powr γ − 1) / γ + d) if y: 0 < y
for y
 proof −
  have limit-quantile G y = H (ln y) + d
    using y by (simp add: H-def)
  also have ... = (if γ = 0 then c * ln y else c * (exp (γ * ln y) − 1) / γ) + d
    using Hform[of ln y] by simp
  also have (if γ = 0 then c * ln y else c * (exp (γ * ln y) − 1) / γ) + d
            = (if γ = 0 then c * ln y + d else c * (y powr γ − 1) / γ + d)
    using y by (simp add: powr-def)
  finally show ?thesis .
 qed
 have c-pos: 0 < c
 proof (rule ccontr)
  assume ¬ 0 < c
  with c-nonneg have c0: c = 0
    by simp
  obtain y1 y2 where y12: 0 < y1 y1 < y2
    and D12: limit-quantile G y1 < limit-quantile G y2
    by (rule nondegenerate-growth-points[OF G deg countable-empty]) auto
  have y2pos: 0 < y2
    using y12 by linarith
  have limit-quantile G y1 = d limit-quantile G y2 = d
    using Dform[OF y12(1)] Dform[OF y2pos] c0 by simp-all
  with D12 show False
    by simp
 qed
 show thesis
  by (rule that[of c γ d]) (use Dform c-pos in auto)
qed
```

The Fisher–Tippett–Gnedenko theorem, which is Theorem 1.1.3 of the book by de Haan and Ferreira [2] (Fisher and Tippett (1928) [4], Gnedenko (1943) [5]).

theorem *Fisher-Tippett-Gnedenko*:

```
  fixes F G :: real ⇒ real and a b :: nat ⇒ real
  assumes F: right-continuous-mono F 0 1 and G: right-continuous-mono G 0 1
    and deg: ¬ degenerate-df G
    and a: ⋀n. 0 < a n
    and wc: weak-conv (λn x. F (a n * x + b n) ^ n) G
  obtains γ μ σ where 0 < σ and G = EVD-cdf γ μ σ
proof −
  obtain γ c d where c-pos: 0 < c
    and Dform: ⋀y. 0 < y ⟹ limit-quantile G y
          = (if γ = 0 then c * ln y + d else c * (y powr γ − 1) / γ + d)
    by (rule limit-quantile-form[OF assms]) simp
  have Dform': ⋀y. 0 < y ⟹ limit-quantile (EVD-cdf γ d c) y
          = (if γ = 0 then c * ln y + d else c * (y powr γ − 1) / γ + d)
    using limit-quantile-EVD-cdf[OF c-pos] by simp
  have eq: ⋀y. 0 < y ⟹ limit-quantile G y = limit-quantile (EVD-cdf γ d c) y
    using Dform Dform' by simp
  have G = EVD-cdf γ d c
    by (intro dist-function-unique-limit-quantile[OF G - eq]
        right-continuous-mono-cdf[OF real-distribution-EVD[OF c-pos]])
  with c-pos that
  show thesis
    by blast
qed

theorem Fisher-Tippett-Gnedenko':
  fixes F G :: real ⇒ real and a b :: nat ⇒ real
  assumes right-continuous-mono F 0 1 right-continuous-mono G 0 1 ¬ degener-
ate-df G
    and ⋀n. 0 < a n
    and weak-conv (λn x. F (a n * x + b n) ^ n) G
  obtains γ α β where 0 < α ⋀x. G x = EVD-cdf γ 0 1 (α * x + β)
proof −
  obtain γ μ σ where 0 < σ and G = EVD-cdf γ μ σ
    using Fisher-Tippett-Gnedenko[OF assms] by metis
  thus ?thesis
    using that[of 1 / σ γ − μ / σ]
    by(simp add: EVD-cdf-location-scale[of σ γ,symmetric] diff-divide-distrib)
qed

corollary Fisher-Tippett-Gnedenko-MDA':
  assumes deg: ¬ degenerate-df G
    and dom: domain-of-attraction F G
  obtains γ μ σ where 0 < σ and G = EVD-cdf γ μ σ
proof −
  obtain a b where a: ⋀n. 0 < a n
    and wc: weak-conv (λn x. F (a n * x + b n) ^ n) G
    using dom unfolding domain-of-attraction-def by blast
  have G: right-continuous-mono G 0 1 and F:right-continuous-mono F 0 1
    using dom unfolding domain-of-attraction-def by auto
```

obtain γ μ σ **where** $0 < \sigma$ **and** $G = EVD\text{-}cdf\ \gamma\ \mu\ \sigma$
by (*rule Fisher-Tippett-Gnedenko*[*OF F G deg a wc*])
thus *thesis*
by (*rule that*)
qed

corollary *Fisher-Tippett-Gnedenko-MDA*:
assumes ¬ *degenerate-df G*
and *d*:*domain-of-attraction F G*
shows ∃γ. *domain-of-attraction F* (*EVD-cdf* γ *0 1*)
proof −
obtain γ μ σ **where** *sigma*:$0 < \sigma$ **and** *G*:$G = EVD\text{-}cdf\ \gamma\ \mu\ \sigma$
using *Fisher-Tippett-Gnedenko-MDA'*[*OF assms*] **by** *metis*
thus *?thesis*
using *d domain-of-attraction-EVD-cdf-iff*[*of* σ]
by(*auto intro*!: *exI*[**where** *x*=γ])
qed

3.3 Alternative Characterizations of the Domain of Attraction

We formalize Theorems 1.1.2 and 1.1.6 of the book [2].

3.3.1 Theorem 1.1.2 Part

lemma *MDA-tail-probability-witness*:
fixes *F a* :: *real* ⇒ *real* **and** γ :: *real*
assumes *F*: *right-continuous-mono F 0 1*
and *a-pos*: ⋀*t*. $0 < a\ t$ **and** *sigma*[*arith*]:$0 < \sigma$
and *tail*: ⋀*x*. $1 + \gamma * (x - \mu) / \sigma > 0 \Longrightarrow$
$((\lambda t.\ t * (1 - F\ (a\ t * x + b\ t))) \longrightarrow - ln\ (EVD\text{-}cdf\ \gamma\ \mu\ \sigma\ x))$ *at-top*
shows $(\lambda n.\ F\ (a\ (real\ n) * x + b\ (real\ n))\ \hat{}\ n) \longrightarrow EVD\text{-}cdf\ \gamma\ \mu\ \sigma\ x$
proof −
have *inside-conv*: $(\lambda n.\ F\ (a\ (real\ n) * x' + b\ (real\ n))\ \hat{}\ n) \longrightarrow EVD\text{-}cdf\ \gamma\ \mu\ \sigma\ x'$
if *supp'*: $1 + \gamma * (x' - \mu) / \sigma > 0$ **for** x'
proof −
have *seqtail*: $(\lambda n{::}nat.\ real\ n * (1 - F\ (a\ (real\ n) * x' + b\ (real\ n)))) \longrightarrow - ln\ (EVD\text{-}cdf\ \gamma\ \mu\ \sigma\ x')$
by (*rule filterlim-compose*[*OF tail*[*OF supp'*] *filterlim-real-sequentially*])
have *Fbound*: ⋀*n*. $0 \le F\ (a\ (real\ n) * x' + b\ (real\ n))$
⋀*n*. $F\ (a\ (real\ n) * x' + b\ (real\ n)) \le 1$
using *right-continuous-mono.a-le-f*[*OF F*] *right-continuous-mono.f-le-b*[*OF F*]
by *auto*
have *EVDbound*: $0 < EVD\text{-}cdf\ \gamma\ \mu\ \sigma\ x'$ $EVD\text{-}cdf\ \gamma\ \mu\ \sigma\ x' \le 1$
using *EVD-cdf-supp-bounds*[*OF supp'*] **by** *auto*
show *?thesis*
using *power-tendsto-iff-tendsto-tail*

```
        [of λn. F (a (real n) * x' + b (real n)) EVD-cdf γ μ σ x']
      Fbound EVDbound seqtail
    by auto
 qed
 consider 1 + γ * (x − μ) / σ > 0 | 1 + γ * (x − μ) / σ ≤ 0 γ > 0
  | 1 + γ * (x − μ) / σ ≤ 0 γ < 0
  by fastforce
 thus (λn. F (a (real n) * x + b (real n)) ^ n) ⟶ EVD-cdf γ μ σ x
 proof cases
  assume 1 + γ * (x − μ) / σ > 0
  thus ?thesis
   using inside-conv by simp
 next
  assume supp-out:1 + γ * (x − μ) / σ ≤ 0 and gamma:γ > 0
  define y0 where y0 = − σ / γ + μ
  have xy0: x ≤ y0
   using supp-out gamma
   by (simp add: y0-def field-simps)
  have val0: EVD-cdf γ μ σ x = 0
   using supp-out gamma by(simp add: EVD-cdf-def Let-def)
  have valY0: EVD-cdf γ μ σ y0 = 0
   using gamma by (simp add: EVD-cdf-def Let-def y0-def)
  have contY0: isCont (EVD-cdf γ μ σ) y0
   by (rule isCont-EVD-cdf) simp
  have base-nonneg: 0 ≤ F (a (real n) * x + b (real n)) ^ n for n
   using right-continuous-mono.a-le-f[OF F] by simp
  show ?thesis
   unfolding val0 tendsto-iff dist-real-def
  proof safe
   fix e :: real
   assume e: 0 < e
   obtain δ where δ: 0 < δ
     ⋀x''. x'' ≠ y0 ∧ |x'' − y0| < δ ⟹ |EVD-cdf γ μ σ x'' − EVD-cdf γ μ σ
y0| < e
    using LIM-D[OF contY0[unfolded isCont-def], of e] e by auto
   define x' where x' = y0 + δ / 2
   have x'y0: y0 < x' x' ≠ y0 ∧ |x' − y0| < δ
    using δ(1) by (simp-all add: x'-def)
   have xx': x ≤ x'
    using xy0 x'y0(1) by linarith
   have anpos: ⋀n. 0 ≤ a (real n)
    using a-pos by (simp add: less-imp-le)
   have supp'': 1 + γ * (x' − μ) / σ > 0
    using x'y0(1) gamma by (simp add: y0-def field-simps)
   have target: EVD-cdf γ μ σ x' < e
    using δ(2)[OF x'y0(2)] valY0 EVD-cdf-supp-bounds[OF supp'']
    by simp
   have convx': (λn. F (a (real n) * x' + b (real n)) ^ n)
                ⟶ EVD-cdf γ μ σ x'
```

```
      using inside-conv[OF supp'] .
    obtain N where N: ⋀n. N ≤ n ⟹ F (a (real n) * x' + b (real n)) ^ n < e
      using order-tendstoD(2)[OF convx' target]
      unfolding eventually-sequentially by blast
    show ∀F n in sequentially. |F (a (real n) * x + b (real n)) ^ n - 0| < e
    proof (intro eventually-sequentiallyI[where c=N])
      fix n
      assume Nn: N ≤ n
      have F (a (real n) * x + b (real n)) ^ n ≤ F (a (real n) * x' + b (real n)) ^ n
        using right-continuous-mono.a-le-f[OF F] anpos xx'
        by (auto intro!: power-mono monoD[OF right-continuous-mono.mono[OF F]] mult-left-mono)
      thus |F (a (real n) * x + b (real n)) ^ n - 0| < e
        using base-nonneg N[OF Nn] by simp
    qed
  qed
 next
  assume supp-out:1 + γ * (x - μ) / σ ≤ 0 and neg:γ < 0
  define y0 where y0 = - σ / γ + μ
  have xy0: y0 ≤ x
    using supp-out neg by (simp add: y0-def field-simps)
  have val0: EVD-cdf γ μ σ x = 1
    using neg supp-out by (simp add: EVD-cdf-def Let-def)
  have valY0: EVD-cdf γ μ σ y0 = 1
    using neg by (simp add: EVD-cdf-def y0-def Let-def)
  have contY0: isCont (EVD-cdf γ μ σ) y0
    by (rule isCont-EVD-cdf) simp
  have base-le1: F (a (real n) * x + b (real n)) ^ n ≤ 1 for n
   using right-continuous-mono.a-le-f[OF F] right-continuous-mono.f-le-b[OF F]
    by (simp add: power-le-one)
  show ?thesis
    unfolding val0 tendsto-iff dist-real-def
  proof safe
    fix e :: real
    assume e: 0 < e
    obtain δ where δ: 0 < δ
      ⋀x''. x'' ≠ y0 ∧ |x'' - y0| < δ ⟹ |EVD-cdf γ μ σ x'' - EVD-cdf γ μ σ y0| < e
      using LIM-D[OF contY0[unfolded isCont-def], of e] e by auto
    define x' where x' = y0 - δ / 2
    have x'y0: x' < y0 x' ≠ y0 ∧ |x' - y0| < δ
      using δ(1) by (simp-all add: x'-def)
    have xx': x' ≤ x
      using xy0 x'y0(1) by linarith
    have anpos: ⋀n. 0 ≤ a (real n)
      using a-pos by (simp add: less-imp-le)
    have supp'': 1 + γ * (x' - μ) / σ > 0
      using x'y0(1) neg by (simp add: y0-def field-simps)
```

```
    have target: 1 − e < EVD-cdf γ μ σ x'
      using δ(2)[OF x'y0(2)] valY0 EVD-cdf-supp-bounds[OF supp'] by simp
    have convx': (λn. F (a (real n) * x' + b (real n)) ^ n)
                  ⟶ EVD-cdf γ μ σ x'
      using inside-conv[OF supp'] .
    obtain N where N: ⋀n. N ≤ n ⟹ 1 − e < F (a (real n) * x' + b (real
n)) ^ n
      using order-tendstoD(1)[OF convx' target]
      unfolding eventually-sequentially by blast
    show ∀F n in sequentially. |F (a (real n) * x + b (real n)) ^ n − 1| < e
    proof (intro eventually-sequentiallyI[where c=N])
      fix n
      assume Nn: N ≤ n
      have F (a (real n) * x' + b (real n)) ^ n ≤ F (a (real n) * x + b (real n))
^ n
        using right-continuous-mono.a-le-f[OF F] anpos xx'
        by (auto intro!: power-mono monoD[OF right-continuous-mono.mono[OF
F]] mult-left-mono)
      thus |F (a (real n) * x + b (real n)) ^ n − 1| < e
        using base-le1 N[OF Nn] by simp
    qed
  qed
 qed
qed

lemma MDA-tail-quantile-to-tail-probability:
 fixes F a :: real ⇒ real and γ x :: real
 assumes F: right-continuous-mono F 0 1
   and a-pos: ⋀t. 0 < a t
   and sigma[arith]:0 < σ
   and quant: ⋀x. 0 < x ⟹
              ((λt. (tail-quantile F (t * x) − b t) / a t)
                ⟶ (if γ = 0
                     then σ * ln x + μ
                     else σ * (x powr γ − 1) / γ + μ)) at-top
   (is ⋀x. 0 < x ⟹ (- ⟶ ?G x) at-top)
   and supp: 1 + γ * (x − μ) / σ > 0
 shows ((λt. t * (1 − F (a t * x + b t)))
        ⟶ − ln (EVD-cdf γ μ σ x)) at-top
proof −
 have st-mono1:σ * ln y + μ < σ * ln z + μ
   if 0 < y y < z for y z
   using that by auto
 have st-mono2:σ * (y powr γ − 1) / γ + μ < σ * (z powr γ − 1) / γ + μ
   if 0 < y y < z γ ≠ 0 for y z
 proof(cases γ > 0)
   show 0 < γ ⟹
        σ * (y powr γ − 1) / γ + μ < σ * (z powr γ − 1) / γ + μ
     using that by(auto intro!: divide-strict-right-mono powr-less-mono2)
```

```
  next
    show ¬ 0 < γ ⟹
          σ * (y powr γ − 1) / γ + μ < σ * (z powr γ − 1) / γ + μ
      using that by(auto intro!: divide-strict-right-mono-neg powr-less-mono2-neg)
  qed
  define ϱ where ϱ = (if γ = 0 then exp ((x − μ) / σ) else (1 + γ * (x − μ) /
σ) powr (1 / γ))
  have ϱ-pos: 0 < ϱ
    using supp by (simp add: ϱ-def)
  have Dϱ: ?G ϱ = x
    using supp by(auto simp: ϱ-def powr-powr)
  have EVD-val: EVD-cdf γ μ σ x
          = exp (− (if γ = 0 then exp (− ((x − μ) / σ))
                   else (1 + γ * (x − μ) / σ) powr (− 1 / γ)))
    using supp by (simp add: EVD-cdf-def Let-def)
  have target-eq: − ln (EVD-cdf γ μ σ x) = 1 / ϱ
    by(auto simp add: EVD-val ϱ-def exp-minus[of (x − μ) / σ] inverse-eq-divide
powr-minus)
  show ((λt. t * (1 − F (a t * x + b t)))
            ⟶ − ln (EVD-cdf γ μ σ x)) at-top
    unfolding target-eq tendsto-iff dist-real-def
  proof safe
    fix e :: real
    assume e: 0 < e
    have inv-cont: isCont (λr. 1 / r) ϱ
      using ϱ-pos by (intro continuous-intros) simp
    obtain δ where δpos: 0 < δ
      and δbound: ⋀r. r ≠ ϱ ⟹ |r − ϱ| < δ ⟹ |1 / r − 1 / ϱ| < e
      using LIM-D[OF inv-cont[unfolded isCont-def], of e] e by auto
    define η where η = min δ ϱ / 2
    have ηpos: 0 < η and ηltϱ: η < ϱ and ηltδ: η < δ
      using δpos ϱ-pos by (simp-all add: η-def)
    define ϱ1 where ϱ1 = ϱ − η
    define ϱ2 where ϱ2 = ϱ + η
    have ϱ1pos: 0 < ϱ1 and ϱ1ltϱ: ϱ1 < ϱ and ϱltϱ2: ϱ < ϱ2
      using ηpos ηltϱ by (simp-all add: ϱ1-def ϱ2-def)
    have e1: |1 / ϱ1 − 1 / ϱ| < e
      using δbound[of ϱ1] ηpos ηltδ by (simp add: ϱ1-def)
    have e2: |1 / ϱ2 − 1 / ϱ| < e
      using δbound[of ϱ2] ηpos ηltδ by (simp add: ϱ2-def)
    have ϱ2pos: 0 < ϱ2
      using ϱltϱ2 ϱ-pos by linarith
    have x2gt: x < ?G ϱ2
      using st-mono1[OF ϱ-pos ϱltϱ2] st-mono2[OF ϱ-pos ϱltϱ2] Dϱ by auto
    have x1lt: ?G ϱ1 < x
      using st-mono1[OF ϱ1pos ϱ1ltϱ] st-mono2[OF ϱ1pos ϱ1ltϱ] Dϱ by auto
    obtain T1 where T1: ⋀t. T1 ≤ t ⟹ x < (tail-quantile F (t * ϱ2) − b t) /
a t
      using order-tendstoD(1)[OF quant[OF ϱ2pos] x2gt]
```

```
    unfolding eventually-at-top-linorder by blast
  obtain T2 where T2: ⋀t. T2 ≤ t ⟹ (tail-quantile F (t * ϱ1) − b t) / a t
< x
    using order-tendstoD(2)[OF quant[OF ϱ1pos] x1lt]
    unfolding eventually-at-top-linorder by blast
  define T where T = max (max T1 T2) (max (1 / ϱ1 + 1) (1 / ϱ2 + 1))
  show ∀F t in at-top. |t * (1 − F (a t * x + b t)) − 1 / ϱ| < e
  proof (intro eventually-at-top-linorderI[where c=T])
    fix t
    assume Tt: T ≤ t
    have T1t: T1 ≤ t and T2t: T2 ≤ t
      and t-ge1: 1 / ϱ1 + 1 ≤ t and t-ge2: 1 / ϱ2 + 1 ≤ t
      using Tt by (simp-all add: T-def)
    have ϱ1inv-pos: 0 < 1 / ϱ1
      using ϱ1pos by simp
    have tpos: 0 < t
      using t-ge1 ϱ1inv-pos by linarith
    have tϱ1: 1 < t * ϱ1
      using mult-strict-right-mono[OF - ϱ1pos, of 1 / ϱ1 t] t-ge1 ϱ1pos
      by simp
    have tϱ2: 1 < t * ϱ2
      using mult-strict-right-mono[OF - ϱ2pos, of 1 / ϱ2 t] t-ge2 ϱ2pos
      by simp
    have ineq1: a t * x + b t < tail-quantile F (t * ϱ2)
      using T1[OF T1t] a-pos[of t] by (simp add: field-simps)
    have ineq2: tail-quantile F (t * ϱ1) < a t * x + b t
      using T2[OF T2t] a-pos[of t] by (simp add: field-simps)
    have p-bounds2: 0 < 1 − 1 / (t * ϱ2) 1 − 1 / (t * ϱ2) < 1
      using tϱ2 by simp-all
    have lower: 1 / ϱ2 < t * (1 − F (a t * x + b t))
    proof −
      have 1 / (t * ϱ2) < 1 − F (a t * x + b t)
        using right-continuous-mono.ginv-le-iff[OF F p-bounds2, of a t * x + b t]
ineq1
        by linarith
      hence t * (1 / (t * ϱ2)) < t * (1 − F (a t * x + b t))
        using tpos by (intro mult-strict-left-mono) auto
      thus ?thesis
        using tpos ϱ2pos by (simp add: field-simps)
    qed
    have p-bounds1: 0 < 1 − 1 / (t * ϱ1) 1 − 1 / (t * ϱ1) < 1
      using tϱ1 by simp-all
    have upper: t * (1 − F (a t * x + b t)) ≤ 1 / ϱ1
    proof −
      have F (tail-quantile F (t * ϱ1)) ≤ F (a t * x + b t)
      by (rule monoD[OF right-continuous-mono.mono[OF F], OF less-imp-le[OF
ineq2]])
      moreover have 1 − 1 / (t * ϱ1) ≤ F (tail-quantile F (t * ϱ1))
        using right-continuous-mono.ginv-attained[OF F p-bounds1] by simp
```

```
      ultimately have 1 − 1 / (t ∗ ϱ1) ≤ F (a t ∗ x + b t)
        by linarith
      hence t ∗ (1 − F (a t ∗ x + b t)) ≤ t ∗ (1 / (t ∗ ϱ1))
        using tpos by (intro mult-left-mono) auto
      moreover have t ∗ (1 / (t ∗ ϱ1)) = 1 / ϱ1
        using tpos ϱ1pos by (simp add: field-simps)
      ultimately show ?thesis
        by linarith
    qed
    show |t ∗ (1 − F (a t ∗ x + b t)) − 1 / ϱ| < e
      using lower upper e1 e2 by linarith
  qed
 qed
qed

lemma MDA-tail-quantile-witness:
  fixes F a :: real ⇒ real and γ :: real
  assumes F: right-continuous-mono F 0 1
    and a-pos: ⋀t. 0 < a t and sigma:0 < σ
    and quant: ⋀x. 0 < x ⟹
              ((λt. (tail-quantile F (t ∗ x) − b t) / a t)
                ⟶ (if γ = 0
                    then σ ∗ ln x + μ
                    else σ ∗ (x powr γ − 1) / γ + μ)) at-top
  shows (λn. F (a (real n) ∗ x + b (real n)) ^ n) ⟶ EVD-cdf γ μ σ x
  by(intro MDA-tail-probability-witness[OF F a-pos]
      MDA-tail-quantile-to-tail-probability[OF F a-pos - quant] sigma) auto

lemma MDA-imp-tail-probability:
  fixes F :: real ⇒ real and γ :: real
  assumes F: right-continuous-mono F 0 1
    and G: right-continuous-mono G 0 1
    and a:⋀n. 0 < a n
    and deg:¬ degenerate-df G
    and lim:⋀y. (λn. F (a n ∗ y + b n) ^ n) ⟶ EVD-cdf γ 0 1 y
    and sigma[arith]:0 < σ
    and supp:1 + γ ∗ x > 0
  shows((λt. t ∗ (1 − F (a (nat ⌊t⌋) ∗ x + tail-quantile F t)))
          ⟶ − ln (EVD-cdf γ 0 1 x)) at-top
proof −
  have G': right-continuous-mono (EVD-cdf γ 0 1) 0 1
    by (rule right-continuous-mono-EVD-cdf) simp
  have deg': ¬ degenerate-df (EVD-cdf γ 0 1)
    by (rule EVD-cdf-nondegenerate) simp
  have wc: weak-conv (λn x'. F (a n ∗ x' + b n) ^ n) (EVD-cdf γ 0 1)
    by(subst weak-conv-EVD-cdf-iff) (auto simp: lim)
  have cont: isCont (limit-quantile (EVD-cdf γ 0 1)) y if y: 0 < y for y
    by(intro isCont-limit-quantile-EVD-cdf y) simp
```

```
  have quant: ((λt. (tail-quantile F (t * x') - tail-quantile F t) / a (nat ⌊t⌋))
                ⟶ (if γ = 0 then 1 * ln x' + 0 else 1 * (x' powr γ - 1) / γ +
0)) at-top
    if x': 0 < x' for x'
  proof -
    note at-x' = real-tail-quantile-tendsto[OF F G' deg' a wc x' cont[OF x']]
    have at-1: ((λt. (tail-quantile F t - b (nat ⌊t⌋))) / a (nat ⌊t⌋))
                  ⟶ limit-quantile (EVD-cdf γ 0 1) 1) at-top
       using real-tail-quantile-tendsto[OF F G' deg' a wc zero-less-one cont[OF
zero-less-one]]
      by simp
    have diff: ((λt. (tail-quantile F (t * x') - b (nat ⌊t⌋))) / a (nat ⌊t⌋)
              - (tail-quantile F t - b (nat ⌊t⌋))) / a (nat ⌊t⌋))
             ⟶ limit-quantile (EVD-cdf γ 0 1) x' - limit-quantile (EVD-cdf γ
0 1) 1) at-top
      by (intro tendsto-diff at-x' at-1)
    have val-1: limit-quantile (EVD-cdf γ 0 1) 1 = 0
     using limit-quantile-EVD-cdf[OF zero-less-one zero-less-one, of γ 0] by simp
    show ?thesis
      using diff val-1 limit-quantile-EVD-cdf[OF zero-less-one x', of γ 0]
      by (simp add: diff-divide-distrib)
  qed
  have a': ⋀t. 0 < a (nat ⌊t⌋)
    using a by simp
  have supp01: 1 + γ * (x - 0) / 1 > 0
    using supp by simp
  show ?thesis
    using MDA-tail-quantile-to-tail-probability
          [where a = λt. a (nat ⌊t⌋) and b = tail-quantile F,
           OF F a' zero-less-one quant supp01]
    by simp
qed

lemma MDA-imp-tail-quantile:
  fixes F :: real ⇒ real and γ :: real
  assumes F: right-continuous-mono F 0 1
    and G: right-continuous-mono G 0 1
    and a:⋀n. 0 < a n
    and deg:¬ degenerate-df G
    and lim:⋀y. (λn. F (a n * y + b n) ^ n) ⟶ EVD-cdf γ 0 1 y
    and sigma[arith]:0 < σ
    and x:x > 0
  shows((λt. (tail-quantile F (t * x) - tail-quantile F t) / a (nat (floor t)))
                      ⟶ (if γ = 0 then ln x else (x powr γ - 1) / γ)) at-top
proof -
  have G': right-continuous-mono (EVD-cdf γ 0 1) 0 1
    by (rule right-continuous-mono-EVD-cdf) simp
  have deg': ¬ degenerate-df (EVD-cdf γ 0 1)
    by (rule EVD-cdf-nondegenerate) simp
```

```
  have wc: weak-conv (λn x'. F (a n * x' + b n) ^ n) (EVD-cdf γ 0 1)
    by(auto intro!: weak-conv-EVD-cdf-iff[THEN iffD2] lim)
  have cont: isCont (limit-quantile (EVD-cdf γ 0 1)) y if y: 0 < y for y
    by(intro isCont-limit-quantile-EVD-cdf y) simp
  have at-x: ((λt. (tail-quantile F (t * x) − b (nat ⌊t⌋)) / a (nat ⌊t⌋))
                ⟶⟶ limit-quantile (EVD-cdf γ 0 1) x) at-top
    by (rule real-tail-quantile-tendsto[OF F G' deg' a wc x cont[OF x]])
  have at-1: ((λt. (tail-quantile F t − b (nat ⌊t⌋)) / a (nat ⌊t⌋))
                ⟶⟶ limit-quantile (EVD-cdf γ 0 1) 1) at-top
   using real-tail-quantile-tendsto[OF F G' deg' a wc zero-less-one cont[OF zero-less-one]]
    by simp
  have diff: ((λt. (tail-quantile F (t * x) − b (nat ⌊t⌋)) / a (nat ⌊t⌋)
              − (tail-quantile F t − b (nat ⌊t⌋)) / a (nat ⌊t⌋))
                ⟶⟶ limit-quantile (EVD-cdf γ 0 1) x − limit-quantile (EVD-cdf γ
0 1) 1) at-top
    by (intro tendsto-diff at-x at-1)
  have val-1: limit-quantile (EVD-cdf γ 0 1) 1 = 0
    using limit-quantile-EVD-cdf[OF zero-less-one zero-less-one, of γ 0] by simp
  show ?thesis
    using diff val-1 limit-quantile-EVD-cdf[OF zero-less-one x, of γ 0]
    by (auto simp: diff-divide-distrib)
qed

lemma MDA-iff-tail-probability:
  fixes F :: real ⇒ real and γ :: real
  assumes F: right-continuous-mono F 0 1
    and a:⋀n. 0 < a n
    and sigma[arith]:0 < σ
  shows (∀ x. (λn. F (a n * x + b n) ^ n) ⟶⟶ EVD-cdf γ μ σ x) ⟷
        (∀ x. (1 + γ * (x − μ) / σ > 0 ⟶
                ((λt. t * (1 − F (a (nat ⌊t⌋) * x + b (nat ⌊t⌋))))
                  ⟶⟶ − ln (EVD-cdf γ μ σ x)) at-top))
proof
  assume wc:∀ x. (λn. F (a n * x + b n) ^ n) ⟶⟶ EVD-cdf γ μ σ x
  have cont: isCont (limit-quantile (EVD-cdf γ μ σ)) y if y: 0 < y for y
    by(intro isCont-limit-quantile-EVD-cdf y) simp
  have ⋀y. 0 < y ⟹
          ((λt. (tail-quantile F (t * y) − b (nat ⌊t⌋)) / a (nat ⌊t⌋))
            ⟶⟶ (if γ = 0 then σ * ln y + μ
                        else σ * (y powr γ − 1) / γ + μ)) at-top
    using real-tail-quantile-tendsto[OF F right-continuous-mono-EVD-cdf
       EVD-cdf-nondegenerate a weak-conv-EVD-cdf-iff[THEN iffD2] - cont]
       wc sigma
    by(auto simp add: limit-quantile-EVD-cdf[OF sigma,of - γ μ])
  from MDA-tail-quantile-to-tail-probability[OF F a sigma this]
  show ∀ x. 0 < 1 + γ * (x − μ) / σ ⟶
          ((λt. t * (1 − F (a (nat ⌊t⌋) * x + b (nat ⌊t⌋))))
            ⟶⟶ − ln (EVD-cdf γ μ σ x)) at-top
    by blast
```

qed(*use MDA-tail-probability-witness*[**where** *a*=λ*t*. *a* (*nat* ⌊*t*⌋)
and *b* =λ*t*. *b* (*nat* ⌊*t*⌋),*OF F a sigma*]
in *simp*)

lemma *MDA-iff-tail-quantile*:
fixes *F* :: *real* ⇒ *real* **and** γ :: *real*
assumes *F*: *right-continuous-mono F 0 1*
and *a*:⋀*n*. *0* < *a n*
and *sigma*[*arith*]:*0* < σ
shows (∀ *x*. (λ*n*. *F* (*a n* ∗ *x* + *b n*) ^ *n*) ⟶ *EVD-cdf* γ μ σ *x*) ⟷
(∀ *x*>*0*. ((λ*t*. (*tail-quantile F* (*t* ∗ *x*) − *b* (*nat* ⌊*t*⌋)) / *a* (*nat* ⌊*t*⌋))
⟶ (*if* γ = *0 then* σ ∗ *ln x* + μ *else* σ ∗ (*x powr* γ − *1*) / γ +
μ)) *at-top*)
proof
assume *wc*:∀ *x*. (λ*n*. *F* (*a n* ∗ *x* + *b n*) ^ *n*) ⟶ *EVD-cdf* γ μ σ *x*
have *cont*: *isCont* (*limit-quantile* (*EVD-cdf* γ μ σ)) *y* **if** *y*: *0* < *y* **for** *y*
by(*intro isCont-limit-quantile-EVD-cdf y*) *simp*
show ∀ *x*>*0*. ((λ*t*. (*tail-quantile F* (*t* ∗ *x*) − *b* (*nat* ⌊*t*⌋)) / *a* (*nat* ⌊*t*⌋))
⟶ (*if* γ = *0 then* σ ∗ *ln x* + μ *else* σ ∗ (*x powr* γ − *1*) / γ +
μ)) *at-top*
using *real-tail-quantile-tendsto*[*OF F right-continuous-mono-EVD-cdf*
EVD-cdf-nondegenerate a weak-conv-EVD-cdf-iff[*THEN iffD2*] - *cont*]
wc
by(*auto simp*: *limit-quantile-EVD-cdf*[*OF sigma*,*of* - γ μ])
qed(*use MDA-tail-quantile-witness*[**where** *a*=λ*t*. *a* (*nat* ⌊*t*⌋) **and** *b* =λ*t*. *b* (*nat*
⌊*t*⌋),*OF assms*]
in *simp*)

lemma *MDA-iff-tail-quantile′*:
fixes *F* :: *real* ⇒ *real* **and** γ :: *real*
assumes *F*: *right-continuous-mono F 0 1*
shows *domain-of-attraction F* (*EVD-cdf* γ *0 1*)
⟷ (∃ *a*. (∀ *t*. *0* < *a t*) ∧
(∀ *x*>*0*. ((λ*t*. (*tail-quantile F* (*t* ∗ *x*) − *tail-quantile F t*) / *a t*)
⟶ (*if* γ = *0 then ln x else* (*x powr* γ − *1*) / γ)) *at-top*))
proof *safe*
assume *dom*: *domain-of-attraction F* (*EVD-cdf* γ *0 1*)
obtain *a b* **where** *a*: ⋀*n*. *0* < *a n*
and *lim*: ⋀*y*. (λ*n*. *F* (*a n* ∗ *y* + *b n*) ^ *n*) ⟶ *EVD-cdf* γ *0 1 y*
using *dom* **unfolding** *domain-of-attraction-def*
by (*auto simp*: *weak-conv-EVD-cdf-iff*)
show ∃ *a*. (∀ *t*. *0* < *a t*) ∧
(∀ *x*>*0*. ((λ*t*. (*tail-quantile F* (*t* ∗ *x*) − *tail-quantile F t*) / *a t*)
⟶ (*if* γ = *0 then ln x else* (*x powr* γ − *1*) / γ)) *at-top*)
using *a*
by(*intro exI*[*of* - λ*t*. *a* (*nat* ⌊*t*⌋)] *conjI allI*
MDA-imp-tail-quantile[*OF F right-continuous-mono-EVD-cdf*
a EVD-cdf-nondegenerate lim] *impI*)
auto

```
next
  fix a
  assume a: ∀ t. 0 < a t
    and quant: ∀ x>0.
                ((λt. (tail-quantile F (t * x) − tail-quantile F t) / a t)
                  ⟶ (if γ = 0 then ln x else (x powr γ − 1) / γ)) at-top
  thus domain-of-attraction F (EVD-cdf γ 0 1)
    by(auto intro!: exI[of - λn. a (real n),
          OF conjI[OF - exI[of - λn. tail-quantile F (real n)]]]
        MDA-tail-quantile-witness[where b = tail-quantile F, OF F]
        right-continuous-mono-EVD-cdf F
        simp: weak-conv-EVD-cdf-iff domain-of-attraction-def)
qed

lemma MDA-iff-tail-probability':
  fixes F :: real ⇒ real and γ :: real
  assumes F: right-continuous-mono F 0 1
  shows domain-of-attraction F (EVD-cdf γ 0 1)
        ⟷ (∃ a. (∀ t. 0 < a t) ∧
                (∀ x. 1 + γ * x > 0 ⟶
                    ((λt. t * (1 − F (a t * x + tail-quantile F t)))
                      ⟶ − ln (EVD-cdf γ 0 1 x)) at-top))
proof safe
  assume dom: domain-of-attraction F (EVD-cdf γ 0 1)
  obtain a b where a: ⋀n. 0 < a n
    and lim: ⋀y. (λn. F (a n * y + b n) ^ n) ⟶ EVD-cdf γ 0 1 y
    using dom unfolding domain-of-attraction-def
    by (auto simp: weak-conv-EVD-cdf-iff)
  show ∃ a. (∀ t. 0 < a t) ∧
        (∀ x. 1 + γ * x > 0 ⟶
            ((λt. t * (1 − F (a t * x + tail-quantile F t)))
              ⟶ − ln (EVD-cdf γ 0 1 x)) at-top)
    using a
    by(intro exI[of - λt. a (nat ⌊t⌋)] conjI allI
        MDA-imp-tail-probability[OF F right-continuous-mono-EVD-cdf
          a EVD-cdf-nondegenerate lim] impI)
      auto
next
  fix a
  assume a: ∀ t. 0 < a t
    and tail: ∀ x. 1 + γ * x > 0 ⟶
                ((λt. t * (1 − F (a t * x + tail-quantile F t)))
                  ⟶ − ln (EVD-cdf γ 0 1 x)) at-top
  thus domain-of-attraction F (EVD-cdf γ 0 1)
    by(auto intro!: exI[of - λn. a (real n),
          OF conjI[OF - exI[of - λn. tail-quantile F (real n)]]]
        MDA-tail-probability-witness[where b = tail-quantile F, OF F]
        right-continuous-mono-EVD-cdf F
        simp: weak-conv-EVD-cdf-iff domain-of-attraction-def)
```

qed

3.3.2 Theorem 1.1.6 Part

definition *x-star* :: (*real* ⇒ *real*) ⇒ *real* **where**
x-star F = *Sup* {*x*. *F x* < *1*}

abbreviation *bdd-support* :: (*real* ⇒ *real*) ⇒ *bool* **where**
bdd-support F ≡ *bdd-above* {*x*. *F x* < *1*}

lemma *not-bdd-supportI*:
assumes ⋀*x*. ∃ *y*>*x*. *F y* < *1*
shows ¬ *bdd-support F*
using *assms* **by**(*auto simp*: *bdd-above-def*) (*meson linorder-not-le*)

lemma *not-bdd-supportI'*:
assumes ⋀*x*. *F x* < *1*
shows ¬ *bdd-support F*
by(*intro not-bdd-supportI*) (*meson assms less-add-one*)

definition *endpoint-filter* :: (*real* ⇒ *real*) ⇒ *real filter* **where**
endpoint-filter F = (*if bdd-support F then at-left* (*x-star F*) *else at-top*)

lemma *endpoint-filter*:
bdd-support F ⟹ *endpoint-filter F* = *at-left* (*x-star F*)
¬ *bdd-support F* ⟹ *endpoint-filter F* = *at-top*
by(*auto simp*: *endpoint-filter-def*)

lemma *endpoint-filter-neq-bot*: *endpoint-filter F* ≠ ⊥
by (*simp add*: *endpoint-filter-def*)

lemma *F-x-star-eq-1*:
assumes *right-continuous-mono F 0 1*
and *bdd*:*bdd-above* {*x*. *F x* < *1*}
shows *F* (*x-star F*) = *1*
proof(*rule antisym*)
interpret *right-continuous-mono F 0 1*
by *fact*
show *1* ≤ *F* (*x-star F*)
proof(*rule ccontr*)
assume *h*:¬ *1* ≤ *F* (*x-star F*)
hence *Sup* {*x*. *F x* < *1*} ∈ {*x*. *F x* < *1*}
by(*auto simp*: *x-star-def*)
have ∀ *ε*>*0*. ∃ *δ*>*0*. *F* (⨆ {*x*. *F x* < *1*} + *δ*) − *F* (⨆ {*x*. *F x* < *1*}) < *ε*
using *mono cont*
by(*subst continuous-at-right-real-increasing*[*symmetric*]) (*auto simp*: *monoD*)
then obtain *d* **where** *d*:*d* > *0*
F (⨆ {*x*. *F x* < *1*} + *d*) − *F* (⨆ {*x*. *F x* < *1*}) < *1* − *F* (*Sup* {*x*. *F x* < *1*})

```
    using h unfolding x-star-def
    by (metis diff-less-0-iff-less less-eq-real-def linorder-not-less right-minus-eq)
  hence F (⨆ {x. F x < 1} + d) < 1
    by simp
  hence (⨆ {x. F x < 1} + d) ≤ (⨆ {x. F x < 1})
    by (metis bdd cSup-upper mem-Collect-eq)
  with d show False
    by simp
 qed
qed(auto intro!: assms right-continuous-mono.f-le-b[of F])

lemma F-tendsto-1-endpoint:
 fixes F :: real ⇒ real and γ :: real
 assumes F: right-continuous-mono F 0 1
  and a:⋀n. 0 < a n
  and sigma[arith]:0 < σ
  and wc:⋀x. (λn. F (a n * x + b n) ^ n) ⟶ EVD-cdf γ μ σ x
 shows (F ⟶ 1) (endpoint-filter F)
proof (cases bdd-support F)
 assume ¬ bdd-support F
 with F show ?thesis
  by(simp add: endpoint-filter right-continuous-mono-def)
next
 assume bdd:bdd-support F
 have EVD-mu: EVD-cdf γ μ σ μ = exp (- 1)
  by (simp add: EVD-cdf-def)
 have supp-mu: 0 < 1 + γ * (μ - μ) / σ
  by simp
 have tail-at-mu: ((λt. t * (1 - F (a (nat ⌊t⌋) * μ + b (nat ⌊t⌋)))))
                  ⟶ - ln (EVD-cdf γ μ σ μ)) at-top
  using F MDA-iff-tail-probability a sigma supp-mu wc
  by blast
 have tail-at-mu': ((λt. t * (1 - F (a (nat ⌊t⌋) * μ + b (nat ⌊t⌋)))) ⟶ 1)
at-top
  using tail-at-mu unfolding EVD-mu by simp
 have inv-tendsto: ((λt::real. 1 / t) ⟶ 0) at-top
  using tendsto-inverse-0-at-top[OF filterlim-ident] by (simp add: inverse-eq-divide)
 have step0: ((λt::real. 1 - F (a (nat ⌊t⌋) * μ + b (nat ⌊t⌋))) ⟶ 0) at-top
 proof -
  have eq: ∀⇩F t::real in at-top. 1 - F (a (nat ⌊t⌋) * μ + b (nat ⌊t⌋))
           = t * (1 - F (a (nat ⌊t⌋) * μ + b (nat ⌊t⌋))) * (1 / t)
    by (rule eventually-mono[OF eventually-gt-at-top[of 0]]) simp
  have prod-tendsto: ((λt. t * (1 - F (a (nat ⌊t⌋) * μ
                      + b (nat ⌊t⌋))) * (1 / t)) ⟶ 0) at-top
    using tendsto-mult[OF tail-at-mu' inv-tendsto] by simp
  show ?thesis
    using tendsto-cong[OF eq] prod-tendsto by simp
 qed
 have Fxstar: F (x-star F) = 1
```

```
  by(intro F-x-star-eq-1 bdd assms)
have filter-eq: endpoint-filter F = at-left (x-star F)
  using bdd by (simp add: endpoint-filter)
have upper-ev: ∀F t in at-left (x-star F). F t ≤ 1
  using right-continuous-mono.f-le-b[OF F] by (intro always-eventually) simp
show ?thesis
  unfolding filter-eq tendsto-iff dist-real-def
proof safe
  fix e :: real
  assume e: 0 < e
  have c2: (1::real) / 2 < 1
    by simp
  have ev-c: ∀F t in at-top. 1 − F (a (nat ⌊t⌋) ∗ μ + b (nat ⌊t⌋)) < e
                ∧ 1 / 2 < t ∗ (1 − F (a (nat ⌊t⌋) ∗ μ + b (nat ⌊t⌋)))
    using eventually-conj[OF order-tendstoD(2)
        [OF step0 e] order-tendstoD(1)[OF tail-at-mu′ c2]] .
  then obtain t0 where t0: 1 − F (a (nat ⌊t0⌋) ∗ μ + b (nat ⌊t0⌋)) < e
     1 / 2 < t0 ∗ (1 − F (a (nat ⌊t0⌋) ∗ μ + b (nat ⌊t0⌋)))
    using eventually-happens′[OF trivial-limit-at-top-linorder] by blast
  define z where z = a (nat ⌊t0⌋) ∗ μ + b (nat ⌊t0⌋)
  have Fz: 1 − e < F z
    using t0(1) by (simp add: z-def)
  have t0pos: 0 < t0
  proof (rule ccontr)
    assume ¬ 0 < t0
    hence t0 ∗ (1 − F z) ≤ 0
      using right-continuous-mono.f-le-b[OF F] unfolding z-def
      by (intro mult-nonpos-nonneg) auto
    with t0(2) show False
      unfolding z-def by simp
  qed
  have Fzlt1: F z < 1
  proof (rule ccontr)
    assume ¬ F z < 1
    hence F z = 1
      using right-continuous-mono.f-le-b[OF F, of z] by linarith
    thus False
      using t0(2) t0pos unfolding z-def by simp
  qed
  have zlt: z < x-star F
  proof −
    have z ≤ x-star F
      using Fzlt1 bdd unfolding x-star-def z-def by (intro cSup-upper) auto
    moreover have z ≠ x-star F
      using Fzlt1 Fxstar by auto
    ultimately show ?thesis
      by simp
  qed
  have lower-ev: ∀F t in at-left (x-star F). 1 − e < F t
```

```
    unfolding eventually-at-left-field
   proof (intro exI[of - z] conjI zlt allI impI)
    fix y
    assume z < y y < x-star F
    hence F z ≤ F y
     by (intro monoD[OF right-continuous-mono.mono[OF F]]) simp
    thus 1 − e < F y
     using Fz by simp
   qed
   show ∀F t in at-left (x-star F). |F t − 1| < e
    using lower-ev upper-ev by eventually-elim linarith
  qed
qed

lemma lt-x-star-iff-F-lt-1:
  fixes F :: real ⇒ real
  assumes F: right-continuous-mono F 0 1 and bdd: bdd-support F
  shows y < x-star F ⟷ F y < 1
proof −
  interpret right-continuous-mono F 0 1
   by fact
  show ?thesis
  proof
   assume c: y < x-star F
   show F y < 1
   proof (rule ccontr)
    assume ¬ F y < 1
    hence F y = 1
     using f-le-b[of y] by linarith
    hence y ≤ x ⟹ F x = 1 for x
     using monoD[OF mono,of y x] f-le-b[of x] by simp
    hence {x. F x < 1} ⊆ {..<y}
    by (metis lessThan-iff linorder-not-le mem-Collect-eq order-less-irrefl subsetI)
    from cSup-subset-mono[OF - - this] ex-less-a[of 1] c
    show False
     by(auto simp: x-star-def)
   qed
  next
   assume y: F y < 1
   hence y ≤ x-star F
    unfolding x-star-def using bdd by (auto intro!: cSup-upper)
   thus y < x-star F
    using y F-x-star-eq-1[OF F bdd] by (metis order-less-le)
  qed
qed

lemma F-lt-1-unbdd-support:
  assumes F: right-continuous-mono F 0 1 and bdd: ¬ bdd-support F
  shows F x < 1
```

proof (*rule ccontr*)
interpret *right-continuous-mono F 0 1*
by *fact*
assume ¬ *F x* < *1*
hence *Fx*: *F x* = *1*
using *f-le-b*[*of x*] **by** *linarith*
hence *Fx1*:*x* ≤ *y* ⟹ *F y* = *1* **for** *y*
using *monoD*[*OF mono*,*of x y*] *f-le-b*[*of y*] **by** *simp*
hence {*x*. *F x* < *1*} ⊆ {..<*x*}
by (*metis lessThan-iff linorder-not-le mem-Collect-eq order-less-irrefl subsetI*)
with *bdd* **show** *False*
by (*meson bdd-above.mono bdd-above-Iio*)
qed

lemma *F-lt-1-endpoint*:
fixes *F* :: *real* ⇒ *real*
assumes *F*: *right-continuous-mono F 0 1*
shows ∀ *F* *t in endpoint-filter F*. *F t* < *1*
using *F-lt-1-unbdd-support*[*OF F*] *lt-x-star-iff-F-lt-1*[*OF F*, *THEN iffD1*]
by (*cases bdd-support F*)
(*auto simp*: *endpoint-filter intro*!: *eventually-at-leftI*[*of x-star F* − *1*])

lemma *tail-quantile-tendsto-x-star*:
fixes *F* :: *real* ⇒ *real*
assumes *F*: *right-continuous-mono F 0 1* **and** *bdd*: *bdd-support F*
shows (*tail-quantile F* ⟶ *x-star F*) *at-top*
proof −
interpret *right-continuous-mono F 0 1*
by *fact*
have *U-le-xstar*: *tail-quantile F t* ≤ *x-star F* **if** *t1*: *1* < *t* **for** *t*
using *t1 F-x-star-eq-1*[*OF F bdd*] **by** (*simp add*: *field-simps ginv-le-iff*)
show *?thesis*
unfolding *tendsto-iff dist-real-def*
proof *safe*
fix *e* :: *real*
assume *e*: *0* < *e*
define *c* **where** *c* = *x-star F* − *e*
have *Fc*: *F c* < *1*
by (*rule lt-x-star-iff-F-lt-1*[*OF F bdd*, *THEN iffD1*]) (*simp add*: *c-def e*)
have *tendsto1*: ((λ*t*::*real*. *1* − *1* / *t*) ⟶ *1*) *at-top*
using *tendsto-diff*[*OF tendsto-const*[*of 1*] *tendsto-inverse-0-at-top*[*OF filter-lim-ident*]]
by (*simp add*: *inverse-eq-divide*)
obtain *T1* **where** *T1*: ⋀*t*. *T1* ≤ *t* ⟹ *F c* < *1* − *1* / *t*
using *order-tendstoD*(*1*)[*OF tendsto1 Fc*]
unfolding *eventually-at-top-linorder* **by** *blast*
define *T* **where** *T* = *max 2 T1*
show ∀ *F* *t in at-top*. |*tail-quantile F t* − *x-star F*| < *e*
proof (*intro eventually-at-top-linorderI*[**where** *c*=*T*])

```
    fix t
    assume Tt: T ≤ t
    have t1: 1 < t and T1t: T1 ≤ t
      using Tt by (simp-all add: T-def)
    have p-bounds: 0 < 1 − 1 / t 1 − 1 / t < 1
      using t1 by simp-all
    have lower: c < tail-quantile F t
      using ginv-le-iff[OF p-bounds, of c] T1[OF T1t] by linarith
    show |tail-quantile F t − x-star F| < e
      using lower U-le-xstar[OF t1] by (simp add: c-def)
  qed
 qed
qed

lemma tail-quantile-lt-x-star:
 fixes F :: real ⇒ real
 assumes F: right-continuous-mono F 0 1 and bdd: bdd-support F
   and Flim: (F ⟶ 1) (endpoint-filter F) and t1: 1 < t
 shows tail-quantile F t < x-star F
proof −
 interpret right-continuous-mono F 0 1
   by fact
 have p0: 0 < 1 − 1 / t and p1: 1 − 1 / t < 1
   using t1 by simp-all
 obtain y where y: 1 − 1 / t < F y F y < 1
   using eventually-happens'[OF endpoint-filter-neq-bot
       eventually-conj[OF order-tendstoD(1)[OF Flim p1] F-lt-1-endpoint[OF F]]]
by blast
 have F (tail-quantile F t) < 1
   using ginv-le-iff[OF p0 p1, of y] y monoD[OF mono, of ginv F (1 − 1 / t) y]
by force
 moreover have tail-quantile F t ≤ x-star F
   using calculation bdd by (auto simp: x-star-def intro!: cSup-upper)
 ultimately show ?thesis
   using F-x-star-eq-1[OF F bdd] by (metis order-less-le)
qed

lemma tail-quantile-tendsto-endpoint:
 fixes F :: real ⇒ real
 assumes F: right-continuous-mono F 0 1
   and Flim: (F ⟶ 1) (endpoint-filter F)
 shows filterlim (tail-quantile F) (endpoint-filter F) at-top
proof (cases bdd-support F)
 case True
 have ∀ F t in at-top. tail-quantile F t ∈ {..<x-star F} ∧ tail-quantile F t ≠ x-star
F
   using eventually-gt-at-top[of 1::real]
   by eventually-elim (use tail-quantile-lt-x-star[OF F True Flim] in force)
 thus ?thesis
```

unfolding *endpoint-filter*(*1*)[*OF True*] *filterlim-at*
using *tail-quantile-tendsto-x-star*[*OF F True*] **by** *simp*
next
case *False*
thus *?thesis*
using *tail-quantile-at-top*[*OF F F-lt-1-unbdd-support*[*OF F*]] **by** (*simp add*: *endpoint-filter*)
qed

lemma *F-tendsto-1-endpoint-of-lim*:
fixes $F :: real \Rightarrow real$ **and** $\gamma :: real$
assumes *F*: *right-continuous-mono F 0 1*
and *f1*:$\bigwedge t$. *bdd-support F* $\Longrightarrow$ $t <$ *x-star F* $\Longrightarrow$ $0 <$ *f t*
and *f2*:$\bigwedge t$. $\neg$ *bdd-support F* $\Longrightarrow$ $0 <$ *f t*
and *sigma*[*arith*]: $0 < \sigma$
and *lim*:$\bigwedge x$. $0 < 1 + \gamma * (x - \mu) / \sigma$
$\Longrightarrow$ $((\lambda t.\ (1 - F\ (t + (x - \mu) * f\ t))\ /\ (1 - F\ t))$
$\longrightarrow - ln\ (EVD\text{-}cdf\ \gamma\ \mu\ \sigma\ x))\ (endpoint\text{-}filter\ F)$
shows $(F \longrightarrow 1)\ (endpoint\text{-}filter\ F)$
proof (*cases bdd-support F*)
assume $\neg$ *bdd-support F*
thus *?thesis*
using *right-continuous-mono.top*[*OF F*]
by (*simp add*: *endpoint-filter*)
next
assume *bdd*:*bdd-support F*
have *goal-eq*: *endpoint-filter F* = *at-left* (*x-star F*)
using *bdd* **by** (*simp add*: *endpoint-filter*)
have *op*: *open* $\{x'.\ 0 < 1 + \gamma * (x' - \mu) / \sigma\}$
by (*intro open-Collect-less continuous-intros*) *simp*
hence $\forall_F\ x'$ *in nhds* μ. $x' \in \{x'.\ 0 < 1 + \gamma * (x' - \mu) / \sigma\}$
by (*intro eventually-nhds-in-open*) *auto*
then obtain η **where** *eta*: $0 < \eta$ **and** *eta-supp*: $\bigwedge x'.\ |x' - \mu| < \eta \Longrightarrow 0 < 1 + \gamma * (x' - \mu) / \sigma$
using *eventually-nhds-metric*[*of* $\lambda x'.\ 0 < 1 + \gamma * (x' - \mu) / \sigma\ \mu$]
by (*auto simp*: *dist-real-def*)
define *x1* **where** $x1 = \mu + \eta / 2$
have *x1-supp*: $0 < 1 + \gamma * (x1 - \mu) / \sigma$
using *eta-supp*[*of x1*] *eta* **by** (*simp add*: *x1-def*)
have *x1-gt*: $\mu < x1$
using *eta* **by** (*simp add*: *x1-def*)
define *c1* **where** $c1 = - ln\ (EVD\text{-}cdf\ \gamma\ \mu\ \sigma\ x1)$
have *c1-pos*: $0 < c1$
using *x1-supp*
by(*cases* $\gamma = 0$)
(*auto simp*: *c1-def ln-EVD-cdf-0 ln-EVD-cdf-1*[*OF sigma*])
have *c1-lt1*: $c1 < 1$
proof (*cases* $\gamma = 0$)
show $\gamma = 0 \Longrightarrow$ *?thesis*

```
    using x1-gt by(simp add: ln-EVD-cdf-0 c1-def)
  next
    assume h:¬ γ = 0
    have eq: c1 = (1 + γ * (x1 − μ) / σ) powr (− 1 / γ)
      using ln-EVD-cdf-1[OF sigma x1-supp h] by (simp add: c1-def)
    show ?thesis
      by (smt (verit) h eq eta mult-eq-0-iff mult-pos-pos mult-sign-intros(3)
          powr01-less-one powr-less-one sigma x1-def x1-supp
          zero-compare-simps(11,7))
  qed
   have limx1: ((λt. (1 − F (t + (x1 − μ) * f t)) / (1 − F t)) ⟶ c1)
(endpoint-filter F)
    using lim[OF x1-supp] by (simp add: c1-def)
  define φ where φ = (λt. 1 − F t)
  have phi-mono: ⋀s t. s ≤ t ⟹ φ t ≤ φ s
    using monoD[OF right-continuous-mono.mono[OF F]] by (auto simp: φ-def)
  have phi-ge0: ⋀t. 0 ≤ φ t
    using right-continuous-mono.f-le-b[OF F] by (simp add: φ-def)
  define L where L = Inf (φ ‘ {..<x-star F})
  have ne: φ ‘ {..<x-star F} ≠ {}
    by force
  have bddb: bdd-below (φ ‘ {..<x-star F})
    using phi-ge0 by (intro bdd-belowI[of - 0]) auto
  have Lge0: 0 ≤ L
    unfolding L-def using phi-ge0 ne bddb by (intro cInf-greatest) auto
  have Lbound: ⋀t. t < x-star F ⟹ L ≤ φ t
    unfolding L-def using bddb by (intro cInf-lower) auto
  have phi-tendsto-L: (φ ⟶ L) (at-left (x-star F))
  proof (rule order-tendstoI)
    fix c
    assume c: c < L
    have ∀F t in at-left (x-star F). t < x-star F
      by (rule eventually-at-leftI[of x-star F − 1]) auto
    thus ∀F t in at-left (x-star F). c < φ t
      by(eventually-elim) (use Lbound c in fastforce)
  next
    fix c
    assume c: L < c
    obtain t0 where t0: t0 < x-star F φ t0 < c
      using cInf-lessD[OF ne, of c] c unfolding L-def by force
    have ∀F t in at-left (x-star F). t0 < t
      using t0 by(auto intro!: eventually-at-leftI[of t0])
    thus ∀F t in at-left (x-star F). φ t < c
      by eventually-elim (auto intro!: order.strict-trans1[of - φ t0 c,OF phi-mono
t0(2)])
  qed
  have ∀F t in at-left (x-star F). c1 / 2 < (1 − F (t + (x1 − μ) * f t)) / (1 −
F t)
    using order-tendstoD(1)[OF limx1, of c1 / 2] c1-pos goal-eq by simp
```

hence *t'-lt*: $\forall_F\ t$ *in at-left* (*x-star F*). $t + (x1 - \mu) * f\ t < x\text{-}star\ F$
proof *eventually-elim*
case (*elim t*)
show *?case*
proof (*rule ccontr*)
assume $\neg\ t + (x1 - \mu) * f\ t < x\text{-}star\ F$
hence $F\ (x\text{-}star\ F) \leq F\ (t + (x1 - \mu) * f\ t)$
by (*intro monoD*[*OF right-continuous-mono.mono*[*OF F*]]) *simp*
hence $F\ (t + (x1 - \mu) * f\ t) = 1$
using *right-continuous-mono.f-le-b*[*OF F*, *of* $t + (x1 - \mu) * f\ t$]
F-x-star-eq-1[*OF F bdd*] **by** *simp*
with *elim c1-pos* **show** *False*
by *simp*
qed
qed
have $\forall_F\ t$ *in at-left* (*x-star F*). $t < x\text{-}star\ F$
by (*rule eventually-at-leftI*[*of x-star F* − *1*]) *auto*
hence *t'-ge*: $\forall_F\ t$ *in at-left* (*x-star F*). $t \leq t + (x1 - \mu) * f\ t$
by *eventually-elim* (*use f1*[*OF bdd*] *x1-gt* **in** *fastforce*)
have *t'-tendsto*: $((\lambda t.\ t + (x1 - \mu) * f\ t) \longrightarrow x\text{-}star\ F)\ (at\text{-}left\ (x\text{-}star\ F))$
unfolding *tendsto-iff dist-real-def*
proof *safe*
fix *e* :: *real*
assume *e*: $0 < e$
have *ev1*: $\forall_F\ t$ *in at-left* (*x-star F*). $x\text{-}star\ F - e < t$
by (*intro order-tendstoD*(*1*)[*OF tendsto-ident-at*]) (*use e* **in** *simp*)
have *lower*: $\forall_F\ t$ *in at-left* (*x-star F*). $x\text{-}star\ F - e < t + (x1 - \mu) * f\ t$
using *eventually-conj*[*OF ev1 t'-ge*] **by** *eventually-elim simp*
show $\forall_F\ t$ *in at-left* (*x-star F*). $|t + (x1 - \mu) * f\ t - x\text{-}star\ F| < e$
using *lower t'-lt* **by** *eventually-elim linarith*
qed
have *t'-filterlim*: *filterlim* $(\lambda t.\ t + (x1 - \mu) * f\ t)$
$(at\text{-}left\ (x\text{-}star\ F))\ (at\text{-}left\ (x\text{-}star\ F))$
using *tendsto-imp-filterlim-at-left*[*OF t'-tendsto*] *t'-lt* **by** *blast*
have *phit'-to-L*: $((\lambda t.\ \varphi\ (t + (x1 - \mu) * f\ t)) \longrightarrow L)\ (at\text{-}left\ (x\text{-}star\ F))$
using *filterlim-compose*[*OF phi-tendsto-L t'-filterlim*] **by** *simp*
have *eq-ratio*: $\forall_F\ t$ *in at-left* (*x-star F*). $((1 - F\ (t + (x1 - \mu) * f\ t)) / (1 - F\ t)) * \varphi\ t$
$= \varphi\ (t + (x1 - \mu) * f\ t)$
using *F-lt-1-endpoint*[*OF F*] **unfolding** *goal-eq*
by *eventually-elim* (*simp add*: φ*-def*)
have *prod-tendsto*: $((\lambda t.\ ((1 - F\ (t + (x1 - \mu) * f\ t)) / (1 - F\ t)) * \varphi\ t)$
$\longrightarrow c1 * L)\ (at\text{-}left\ (x\text{-}star\ F))$
using *tendsto-mult*[*OF limx1*[*unfolded goal-eq*] *phi-tendsto-L*] **by** *simp*
note *phit'-to-c1L* = *Lim-transform-eventually*[*OF prod-tendsto eq-ratio*]
have *LeqC1L*: $L = c1 * L$
by (*rule tendsto-unique*[*OF trivial-limit-at-left-real*[*of x-star F*]
phit'-to-L phit'-to-c1L])
have *Lzero*: $L = 0$

```
    using c1-lt1 LeqC1L by simp
  have z: ((λt. 1 − F t) ⟶ 0) (at-left (x-star F))
    using phi-tendsto-L Lzero by (simp add: φ-def)
  have ((λt. 1 − (1 − F t)) ⟶ 1 − (0::real)) (at-left (x-star F))
    by (rule tendsto-diff[OF tendsto-const z])
  thus ?thesis
    by(simp add: goal-eq)
qed

lemma s-tendsto-at-top-endpoint:
  fixes F :: real ⇒ real and γ :: real
  assumes F: right-continuous-mono F 0 1
    and a:⋀n. 0 < a n
    and sigma[arith]:0 < σ
    and wc:⋀x. (λn. F (a n * x + b n) ^ n) ⟶ EVD-cdf γ μ σ x
  shows filterlim (λt. 1 / (1 − F t)) at-top (endpoint-filter F)
proof −
  have Ftend: ((λt. 1 − F t) ⟶ 0) (endpoint-filter F)
    using tendsto-diff[OF tendsto-const[where k=1::real]
        F-tendsto-1-endpoint[OF F a sigma wc]]
    by simp
  have Fpos: ∀F t in endpoint-filter F. 0 < 1 − F t
    using F-lt-1-endpoint[OF F] by (rule eventually-mono) simp
  show ?thesis
    using filterlim-inverse-at-top[OF Ftend Fpos]
    by (simp add: inverse-eq-divide)
qed

lemma t-div-a-tendsto-mu-endpoint:
  fixes F :: real ⇒ real and γ :: real
  assumes F: right-continuous-mono F 0 1
    and a:⋀n. 0 < a n
    and sigma[arith]:0 < σ
    and wc:⋀x. (λn. F (a n * x + b n) ^ n) ⟶ EVD-cdf γ μ σ x
  shows ((λt. (t − b (nat ⌊1 / (1 − F t)⌋)))
            / a (nat ⌊1 / (1 − F t)⌋)) ⟶ μ) (endpoint-filter F)
proof −
  define s where s = (λt::real. 1 / (1 − F t))
  define N where N = (λt::real. nat ⌊s t⌋)
  have goal-eq: (λt. (t − b (nat ⌊1 / (1 − F t)⌋))) / a (nat ⌊1 / (1 − F t)⌋))
                = (λt. (t − b (N t)) / a (N t))
    by (simp add: s-def N-def)
  have stop: filterlim s at-top (endpoint-filter F)
    unfolding s-def by (rule s-tendsto-at-top-endpoint[OF F a sigma wc])
  note Flt1 = F-lt-1-endpoint[OF F]
  have spos: ∀F t in endpoint-filter F. 0 < s t
    using stop[unfolded filterlim-at-top-dense, rule-format, of 0] by simp
  have MDAtq: ∀y>0. ((λt. (tail-quantile F (t * y) − b (nat ⌊t⌋)) / a (nat ⌊t⌋))
              ⟶ (if γ = 0 then σ * ln y + μ  else σ * (y powr γ − 1) / γ +
```

```
μ)) at-top
    using MDA-iff-tail-quantile[OF F a sigma] wc by fast
  define D where D = (λy::real. if γ = 0 then σ * ln y + μ else σ * (y powr γ
- 1) / γ + μ)
  have D1:D 1 = μ
    by(simp add: D-def)
  have Dcont: isCont D 1
    unfolding D-def
    by(cases γ = 0)
      (auto intro!: isCont-add isCont-mult isCont-ln isCont-divide isCont-diff con-
tinuous-powr)
  have conv: ⋀y'. |y' - 1| < 1 ⟹
    ((λt. (tail-quantile F (s t * y') - b (N t)) / a (N t)) ⟶ D y') (endpoint-filter
F)
    using MDAtq by(auto simp: D-def N-def intro!: filterlim-compose[OF - stop])
  have low: ∀ F t in endpoint-filter F. (tail-quantile F (s t * y') - b (N t)) / a (N
t)
                                    ≤ (t - b (N t)) / a (N t)
    if y': (1::real) - 1 < y' y' < 1 for y'
    using stop[unfolded filterlim-at-top-dense, rule-format, of 1 / y'] spos Flt1
  proof eventually-elim
    case (elim t)
    hence s-gt: 1 / y' < s t and s-pos: 0 < s t and Ft: F t < 1
      by auto
    have u1: 1 < s t * y'
      using mult-strict-right-mono[OF s-gt] y' by force
    have u2: s t * y' < 1 / (1 - F t)
      using mult-strict-left-mono[OF y'(2) s-pos]
      by(simp add: s-def)
    show ?case
      using a[of N t] Ft u1 u2
      by(auto intro!: divide-right-mono tail-quantile-le-of-lt-inv[OF F])
  qed
  have up: ∀ F t in endpoint-filter F. (t - b (N t)) / a (N t)
                                    ≤ (tail-quantile F (s t * y') - b (N t)) / a (N t)
    if y': (1::real) < y' y' < 1 + 1 for y'
    using spos Flt1
  proof eventually-elim
    case (elim t)
    hence s-pos: 0 < s t and Ft: F t < 1
      by auto
    have u: 1 / (1 - F t) < s t * y'
      using mult-strict-left-mono[OF - s-pos] y'
      by(simp add: s-def)
    show ?case
      using a[of N t]
     by(auto intro!: divide-right-mono less-imp-le[OF t-lt-tail-quantile-of-gt-inv[OF
F Ft u]])
  qed
```

show *?thesis*
using *squeeze-tendsto-local*[*OF zero-less-one Dcont conv low up*]
unfolding *goal-eq*
by (*simp add*: *D1*)
qed

lemma *MDA-imp-divide-tail*:
assumes *F*: *right-continuous-mono F 0 1*
and *a*:$\bigwedge n.\ 0 < a\ n$
and *sigma*[*arith*]:$0 < \sigma$
and *wc*:$\bigwedge x.\ (\lambda n.\ F\ (a\ n * x + b\ n)\ \hat{}\ n) \longrightarrow EVD\text{-}cdf\ \gamma\ \mu\ \sigma\ x$
and *x*:$0 < 1 + \gamma * (x - \mu) / \sigma$
shows $((\lambda t.\ (1 - F\ (t + (x - \mu) * a\ (nat\ \lfloor 1 / (1 - F\ (t))\rfloor)))) / (1 - F\ t))$
$\longrightarrow - ln\ (EVD\text{-}cdf\ \gamma\ \mu\ \sigma\ x))\ (endpoint\text{-}filter\ F)$
proof −
define *s* **where** $s = (\lambda t{::}real.\ 1 / (1 - F\ t))$
define *N* **where** $N = (\lambda t{::}real.\ nat\ \lfloor s\ t\rfloor)$
have *goal-eq*: $(\lambda t.\ (1 - F\ (t + (x - \mu) * a\ (nat\ \lfloor 1 / (1 - F\ (t))\rfloor)))) / (1 - F\ t))$
$= (\lambda t.\ (1 - F\ (t + (x - \mu) * a\ (N\ t))) / (1 - F\ t))$
by (*simp add*: *s-def N-def*)
have *stop*: *filterlim s at-top* (*endpoint-filter F*)
unfolding *s-def* **by**(*intro s-tendsto-at-top-endpoint*[*OF - - - wc*] *assms*)
note *Flt1* = *F-lt-1-endpoint*[*OF F*]
have *delta-tendsto*: $((\lambda t.\ (t - b\ (N\ t)) / a\ (N\ t) - \mu) \longrightarrow 0)\ (endpoint\text{-}filter\ F)$
using *tendsto-diff*[*OF t-div-a-tendsto-mu-endpoint*[*OF F a sigma wc*] *tendsto-const*[*of* μ]]
by(*simp add*: *s-def N-def*)
have *op*: *open* $\{x'.\ 0 < 1 + \gamma * (x' - \mu) / \sigma\}$
by (*auto intro*!: *open-Collect-less continuous-on-add*
continuous-on-divide continuous-on-mult continuous-on-diff)
have $\forall_F\ x'$ *in nhds x*. $x' \in \{x'.\ 0 < 1 + \gamma * (x' - \mu) / \sigma\}$
using *x op* **by** (*intro eventually-nhds-in-open*) *auto*
hence *ev*: $\forall_F\ x'$ *in nhds x*. $0 < 1 + \gamma * (x' - \mu) / \sigma$
by *eventually-elim simp*
then obtain η **where** *eta*: $0 < \eta$
and *eta-supp*: $\bigwedge x'.\ |x' - x| < \eta \Longrightarrow 0 < 1 + \gamma * (x' - \mu) / \sigma$
using *eventually-nhds-metric*[*of* $\lambda x'.\ 0 < 1 + \gamma * (x' - \mu) / \sigma\ x$]
by(*auto simp*: *dist-real-def*)
have *Dcont*: *isCont* $(\lambda x'.\ ln\ (EVD\text{-}cdf\ \gamma\ \mu\ \sigma\ x'))\ x$
using *EVD-cdf-supp-bounds*[*OF x sigma*]
by(*auto intro*!: *isCont-ln' isCont-EVD-cdf*)
have *conv*: $((\lambda t.\ -((1 - F\ (a\ (N\ t) * x' + b\ (N\ t))) / (1 - F\ t)))$
$\longrightarrow ln\ (EVD\text{-}cdf\ \gamma\ \mu\ \sigma\ x'))\ (endpoint\text{-}filter\ F)$
if x': $|x' - x| < \eta$ **for** x'
proof −
have *supp'*: $0 < 1 + \gamma * (x' - \mu) / \sigma$
using *that eta-supp* **by** *simp*

```
    have base: ((λt. t * (1 − F (a (nat ⌊t⌋) * x′ + b (nat ⌊t⌋)))))
                  ⟶ − ln (EVD-cdf γ μ σ x′)) at-top
      using MDA-iff-tail-probability[where a=a,OF F a sigma] wc eta-supp supp′
by blast
    show ?thesis
      using tendsto-minus[OF filterlim-compose[OF base stop]] by(simp add: N-def
s-def)
  qed
  have low: ∀F t in endpoint-filter F. − ((1 − F (a (N t) * x′ + b (N t))) / (1 −
F t))
                                  ≤ − ((1 − F (t + (x − μ) * a (N t))) / (1 − F t))
    if x′: x − η < x′ x′ < x for x′
  proof −
    have dlt: x′ − x < 0
      using x′ by simp
    have ev1: ∀F t in endpoint-filter F. x′ − x < (t − b (N t)) / a (N t) − μ
      using order-tendstoD(1)[OF delta-tendsto dlt] .
    show ∀F t in endpoint-filter F. − ((1 − F (a (N t) * x′ + b (N t))) / (1 − F
t))
                                  ≤ − ((1 − F (t + (x − μ) * a (N t))) / (1 − F t))
      using ev1 Flt1
    proof eventually-elim
      case (elim t)
      hence d: x′ − x < (t − b (N t)) / a (N t) − μ and Ft: F t < 1
        by auto
      have apos: 0 < a (N t)
        using a by simp
      have (x′ − x) * a (N t) < ((t − b (N t)) / a (N t) − μ) * a (N t)
        using mult-strict-right-mono[OF d apos] .
      also have ... = t − b (N t) − μ * a (N t)
        using apos by (simp add: field-simps)
      finally have ineq: (x′ − x) * a (N t) < t − b (N t) − μ * a (N t) .
      have a (N t) * x′ + b (N t) < t + (x − μ) * a (N t)
        using ineq by (simp add: field-simps)
      hence fmono: F (a (N t) * x′ + b (N t)) ≤ F (t + (x − μ) * a (N t))
        by (intro monoD[OF right-continuous-mono.mono[OF F]]) simp
      have hFQ: 1 − F (t + (x − μ) * a (N t)) ≤ 1 − F (a (N t) * x′ + b (N t))
        using fmono by simp
      have (1 − F (t + (x − μ) * a (N t))) / (1 − F t)
            ≤ (1 − F (a (N t) * x′ + b (N t))) / (1 − F t)
        by (rule divide-right-mono[OF hFQ]) (use Ft in simp)
      thus ?case
        by simp
    qed
  qed
  have up: ⋀x′. x < x′ ⟹ x′ < x + η ⟹
     ∀F t in endpoint-filter F. − ((1 − F (t + (x − μ) * a (N t))) / (1 − F t))
                                  ≤ − ((1 − F (a (N t) * x′ + b (N t))) / (1 − F t))
  proof −
```

```
    fix x' :: real
    assume x': x < x' x' < x + η
    have dpos: 0 < x' − x
      using x' by simp
    have ev1: ∀F t in endpoint-filter F. (t − b (N t)) / a (N t) − μ < x' − x
      using order-tendstoD(2)[OF delta-tendsto dpos] .
    show ∀F t in endpoint-filter F. − ((1 − F (t + (x − μ) * a (N t))) / (1 − F
t))
                                  ≤ − ((1 − F (a (N t) * x' + b (N t))) / (1 − F t))
      using ev1 Flt1
    proof eventually-elim
      case (elim t)
      hence d: (t − b (N t)) / a (N t) − μ < x' − x and Ft: F t < 1
        by auto
      have apos: 0 < a (N t)
        using a by simp
      have ((t − b (N t)) / a (N t) − μ) * a (N t) < (x' − x) * a (N t)
        using mult-strict-right-mono[OF d apos] .
      also have ((t − b (N t)) / a (N t) − μ) * a (N t) = t − b (N t) − μ * a (N
t)
        using apos by (simp add: field-simps)
      finally have ineq: t − b (N t) − μ * a (N t) < (x' − x) * a (N t)
        by simp
      have t + (x − μ) * a (N t) < a (N t) * x' + b (N t)
        using ineq by (simp add: field-simps)
      hence fmono: F (t + (x − μ) * a (N t)) ≤ F (a (N t) * x' + b (N t))
        by (intro monoD[OF right-continuous-mono.mono[OF F]]) simp
      have hFQ: 1 − F (a (N t) * x' + b (N t)) ≤ 1 − F (t + (x − μ) * a (N t))
        using fmono by simp
      have (1 − F (a (N t) * x' + b (N t))) / (1 − F t)
            ≤ (1 − F (t + (x − μ) * a (N t))) / (1 − F t)
        by (rule divide-right-mono[OF hFQ]) (use Ft in simp)
      thus ?case
        by simp
    qed
  qed
  show ?thesis
    unfolding goal-eq
    using tendsto-minus[OF squeeze-tendsto-local[OF eta Dcont conv low up]]
    by simp
qed

lemma tail-quantile-asymp-inverse-of-lim:
  fixes F :: real ⇒ real and γ :: real
  assumes F: right-continuous-mono F 0 1
    and f1:⋀t. bdd-support F ⟹ t < x-star F ⟹ 0 < f t
    and f2:⋀t. ¬ bdd-support F ⟹ 0 < f t
    and sigma[arith]: 0 < σ
    and lim:⋀x. 0 < 1 + γ * (x − μ) / σ
```

$\Longrightarrow ((\lambda t.\ (1 - F\ (t + (x - \mu) * f\ t))\ /\ (1 - F\ t))$
$\longrightarrow - ln\ (EVD\text{-}cdf\ \gamma\ \mu\ \sigma\ x))\ (endpoint\text{-}filter\ F)$
shows $((\lambda t.\ t * (1 - F\ (tail\text{-}quantile\ F\ t))) \longrightarrow 1)\ at\text{-}top$
proof −
interpret *right-continuous-mono F 0 1*
by *fact*
note *Flim = F-tendsto-1-endpoint-of-lim[OF F f1 f2 sigma lim]*
note *Uendpoint = tail-quantile-tendsto-endpoint[OF F Flim]*
have *fU-pos*: $0 < f\ (tail\text{-}quantile\ F\ t)$ **if** *t1*: $1 < t$ **for** t
proof (*cases bdd-support F*)
case *True*
show *?thesis*
by (*rule f1[OF True tail-quantile-lt-x-star[OF F True Flim t1]]*)
next
case *False*
show *?thesis*
by (*rule f2[OF False]*)
qed
have *fact-le1*: $\bigwedge t.\ 1 < t \Longrightarrow t * (1 - F\ (tail\text{-}quantile\ F\ t)) \leq 1$
by (*smt* (*verit*, *best*) *divide-less-eq zero-less-mult-iff t-lt-tail-quantile-of-gt-inv*[*OF F*])
have *fact-gt1*: $\bigwedge t\ z.\ 1 < t \Longrightarrow z < tail\text{-}quantile\ F\ t \Longrightarrow 1 < t * (1 - F\ z)$
by (*smt* (*verit*, *ccfv-SIG*) *divide-less-eq-1-pos divide-pos-pos*
nonzero-mult-div-cancel-left pos-less-divide-eq
ginv-le-iff)
have *FUlt1*: $\forall_F\ t\ in\ at\text{-}top.\ F\ (tail\text{-}quantile\ F\ t) < 1$
by (*rule eventually-compose-filterlim*[*OF F-lt-1-endpoint*[*OF F*] *Uendpoint*])
define D **where** $D = (\lambda x'.\ -\ ln\ (EVD\text{-}cdf\ \gamma\ \mu\ \sigma\ x'))$
have *Dmu*: $D\ \mu = 1$
by (*cases* $\gamma = 0$) (*simp-all add*: *D-def ln-EVD-cdf-0 ln-EVD-cdf-1*)
have *supp-mu*: $0 < 1 + \gamma * (\mu - \mu)\ /\ \sigma$
by *simp*
have *Dcont*: *isCont D* μ
unfolding *D-def*
using *isCont-EVD-cdf*[*OF sigma*] *EVD-cdf-supp-bounds*[*OF supp-mu sigma*]
by (*intro continuous-intros*) *auto*
have *op*: *open* $\{x'.\ 0 < 1 + \gamma * (x' - \mu)\ /\ \sigma\}$
by (*intro open-Collect-less continuous-intros*) *simp*
have $\forall_F\ x'\ in\ nhds\ \mu.\ x' \in \{x'.\ 0 < 1 + \gamma * (x' - \mu)\ /\ \sigma\}$
using *op* **by** (*intro eventually-nhds-in-open*) *auto*
hence *ev*: $\forall_F\ x'\ in\ nhds\ \mu.\ 0 < 1 + \gamma * (x' - \mu)\ /\ \sigma$
by *eventually-elim simp*
then obtain η **where** *eta*: $0 < \eta$ **and** *eta-supp*: $\bigwedge x'.\ |x' - \mu| < \eta \Longrightarrow 0 < 1 + \gamma * (x' - \mu)\ /\ \sigma$
using *eventually-nhds-metric*[*of* $\lambda x'.\ 0 < 1 + \gamma * (x' - \mu)\ /\ \sigma\ \mu$]
by(*auto simp*: *dist-real-def*)
have *upper*: $t * (1 - F\ (tail\text{-}quantile\ F\ t)) \leq 1$ **if** *t1*: $1 < t$ **for** t
using *fact-le1*[*OF t1*] **.**
show *?thesis*

```
  unfolding tendsto-iff dist-real-def
 proof safe
  fix e :: real
  assume e: 0 < e
  define e' where e' = min (1 / 2) e
  have e'-pos: 0 < e' and e'-le: e' ≤ e and e'-lt1: e' < 1
   using e by (simp-all add: e'-def)
  define c where c = 1 − e'
  have cpos[arith]: 0 < c and clt1[arith]: c < 1
   using e'-pos e'-lt1 by (simp-all add: c-def)
  define c' where c' = 1 / c
  have c'gt1: 1 < c'
   by (simp add: c'-def)
  obtain δ where δpos: 0 < δ
   and δbound: ⋀x''. x'' ≠ μ ∧ |x'' − μ| < δ ⟹ |D x'' − D μ| < c' − 1
   using LIM-D[OF Dcont[unfolded isCont-def], of c' − 1] c'gt1 by auto
  have minpos: 0 < min δ η min δ η ≤ δ min δ η ≤ η
   using δpos eta by auto
  define x1 where x1 = μ − min δ η / 2
  have x1-diff: x1 − μ = − (min δ η / 2)
   by (simp add: x1-def)
  have x1-abs: |x1 − μ| = min δ η / 2
   using minpos(1) by (simp add: x1-diff)
  have x1-ne: x1 ≠ μ ∧ |x1 − μ| < δ
   using minpos x1-abs by linarith
  have x1-eta: |x1 − μ| < η
   using minpos x1-abs by linarith
  have x1-lt: x1 < μ
   using minpos x1-diff by linarith
  have x1-supp: 0 < 1 + γ * (x1 − μ) / σ
   using eta-supp[OF x1-eta] .
  have Dx1: D x1 < c'
   using δbound[OF x1-ne] Dmu by linarith
  have Dx1-pos: 0 < D x1
   using x1-supp by (cases γ = 0)
                    (auto simp: EVD-cdf-def D-def)
  have ratio: ((λt. (1 − F (tail-quantile F t + (x1 − μ) * f (tail-quantile F t)))
                    / (1 − F (tail-quantile F t))) ⟶ D x1) at-top
   using filterlim-compose[OF lim[OF x1-supp] Uendpoint] by (simp add: D-def)
  obtain T1 where T1: ⋀t. T1 ≤ t ⟹
     (1 − F (tail-quantile F t + (x1 − μ) * f (tail-quantile F t)))
       / (1 − F (tail-quantile F t)) < c'
    using order-tendstoD(2)[OF ratio Dx1] unfolding eventually-at-top-linorder
by blast
  obtain T2 where T2: ⋀t. T2 ≤ t ⟹ 0 <
     (1 − F (tail-quantile F t + (x1 − μ) * f (tail-quantile F t)))
       / (1 − F (tail-quantile F t))
   using order-tendstoD(1)[OF ratio Dx1-pos] unfolding eventually-at-top-linorder
    by blast
```

```
    obtain T3 where T3: ⋀t. T3 ≤ t ⟹ F (tail-quantile F t) < 1
      using FUlt1 unfolding eventually-at-top-linorder by blast
    define T where T = max (max T1 T2) (max T3 2)
    have lower-ev: ∀F t in at-top. c < t * (1 − F (tail-quantile F t))
    proof (intro eventually-at-top-linorderI[where c=T])
      fix t
      assume Tt: T ≤ t
      have T1t: T1 ≤ t and T2t: T2 ≤ t and T3t: T3 ≤ t and t1: 1 < t
        using Tt by (simp-all add: T-def)
      have rlt: (1 − F (tail-quantile F t + (x1 − μ) * f (tail-quantile F t)))
                / (1 − F (tail-quantile F t)) < c'
        using T1[OF T1t] .
      have rpos: 0 < (1 − F (tail-quantile F t + (x1 − μ) * f (tail-quantile F t)))
                / (1 − F (tail-quantile F t))
        using T2[OF T2t] .
      have FUlt: F (tail-quantile F t) < 1
        using T3[OF T3t] .
      have gt1: 1 < t * (1 − F (tail-quantile F t + (x1 − μ) * f (tail-quantile F
t)))
        using x1-lt fU-pos[OF t1] by(auto intro!: fact-gt1 t1 mult-neg-pos)
      have t * (1 − F (tail-quantile F t + (x1 − μ) * f (tail-quantile F t)))
          < t * (c' * (1 − F (tail-quantile F t)))
        using rlt FUlt t1
        by(intro comm-mult-strict-left-mono)
          (auto simp: field-simps)
      hence c * 1 < c * (t * (1 − F (tail-quantile F t)) / c)
        using gt1 order.strict-trans
        by(intro mult-strict-left-mono) (fastforce simp: c'-def)+
      thus c < t * (1 − F (tail-quantile F t))
        by simp
    qed
    have upper-ev: ∀F t in at-top. t * (1 − F (tail-quantile F t)) ≤ 1
      using eventually-gt-at-top[of 1::real] by eventually-elim (use upper in blast)
    show ∀F t in at-top. |t * (1 − F (tail-quantile F t)) − 1| < e
      using lower-ev upper-ev
      by eventually-elim (use e'-le in simp add: c-def)
  qed
qed

lemma divide-tail-imp-MDA:
  assumes F: right-continuous-mono F 0 1
    and a:⋀n. 0 < a n
    and sigma[arith]:0 < σ
    and f1:⋀t. bdd-support F ⟹ t < x-star F ⟹ 0 < f t
    and f2:⋀t. ¬ bdd-support F ⟹ 0 < f t
    and lim:⋀x. 0 < 1 + γ * (x − μ) / σ
          ⟹ ((λt. (1 − F (t + (x − μ) * f t)) / (1 − F t))
                ⟶ − ln (EVD-cdf γ μ σ x)) (endpoint-filter F)
  shows ∃ a b. (∀ n. 0 < a n) ∧ (∀ x. (λn. F (a n * x + b n) ^ n) ⟶ EVD-cdf
```

γ μ σ x)
proof −
note *Flim* = *F-tendsto-1-endpoint-of-lim*[*OF F f1 f2 sigma lim*]
note *Uendpoint* = *tail-quantile-tendsto-endpoint*[*OF F Flim*]
have *fU-pos*: $0 < f$ (*tail-quantile F t*) **if** *t1*:$1 < t$ **for** t
proof (*cases bdd-support F*)
case *True*
show *?thesis*
by (*rule f1*[*OF True tail-quantile-lt-x-star*[*OF F True Flim t1*]])
next
case *False*
show *?thesis*
by (*rule f2*[*OF False*])
qed
note *asymp* = *tail-quantile-asymp-inverse-of-lim*[*OF F f1 f2 sigma lim*]
note *Flt1* = *F-lt-1-endpoint*[*OF F*]
have *FUlt1*: $\forall_F$ t *in at-top*. F (*tail-quantile F t*) < 1
by (*rule eventually-compose-filterlim*[*OF Flt1 Uendpoint*])
define A **where** $A = (\lambda t.$ *if* $1 < t$ *then* f (*tail-quantile F t*) *else* f (*tail-quantile F 2*))
define B **where** $B = (\lambda t.$ *if* $1 < t$ *then tail-quantile F t* $- \mu * f$ (*tail-quantile F t*)
else tail-quantile F 2 $- \mu * f$ (*tail-quantile F 2*))
have *A-pos*: $\bigwedge t.\ 0 < A\ t$
using *fU-pos*[*of 2*] *fU-pos* **by**(*auto simp*: *A-def*)
have *AB-eq*: $\bigwedge t.\ 1 < t \Longrightarrow A\ t = f$ (*tail-quantile F t*)
$\bigwedge t.\ 1 < t \Longrightarrow B\ t =$ *tail-quantile F t* $- \mu * f$ (*tail-quantile F t*)
unfolding *A-def B-def* **by** *simp-all*
have *tail*: $\bigwedge x.\ 0 < 1 + \gamma * (x - \mu) / \sigma \Longrightarrow$
$((\lambda t.\ t * (1 - F\ (A\ t * x + B\ t))) \longrightarrow - ln\ (EVD\text{-}cdf\ \gamma\ \mu\ \sigma\ x))$ *at-top*
proof −
fix x :: *real*
assume *supp*: $0 < 1 + \gamma * (x - \mu) / \sigma$
have *argeq*: $\bigwedge t.\ 1 < t \Longrightarrow$
$A\ t * x + B\ t =$ *tail-quantile F t* $+ (x - \mu) * f$ (*tail-quantile F t*)
using *AB-eq* **by** (*simp add*: *algebra-simps*)
have *prod*: $((\lambda t.\ (t * (1 - F\ (tail\text{-}quantile\ F\ t)))) *$
$((1 - F\ (tail\text{-}quantile\ F\ t + (x - \mu) * f\ (tail\text{-}quantile\ F\ t)))$
$/ (1 - F\ (tail\text{-}quantile\ F\ t))))$
$\longrightarrow 1 * (- ln\ (EVD\text{-}cdf\ \gamma\ \mu\ \sigma\ x)))$ *at-top*
by (*intro tendsto-mult asymp filterlim-compose*[*OF lim*[*OF supp*] *Uendpoint*])
have *eqprod*: $\forall_F$ t *in at-top*. $(t * (1 - F\ (tail\text{-}quantile\ F\ t))) *$
$((1 - F\ (tail\text{-}quantile\ F\ t + (x - \mu) * f\ (tail\text{-}quantile\ F\ t)))$
$/ (1 - F\ (tail\text{-}quantile\ F\ t)))$
$= t * (1 - F\ (tail\text{-}quantile\ F\ t + (x - \mu) * f\ (tail\text{-}quantile\ F\ t)))$
using *FUlt1* **by** *eventually-elim* (*simp add*: *field-simps*)
have *main*: $((\lambda t.\ t * (1 - F\ (tail\text{-}quantile\ F\ t + (x - \mu) * f\ (tail\text{-}quantile\ F\ t))))$
$\longrightarrow - ln\ (EVD\text{-}cdf\ \gamma\ \mu\ \sigma\ x))$ *at-top*

```
    using Lim-transform-eventually[OF prod eqprod] by simp
  have eqarg: ∀ F t in at-top.
 t * (1 - F (tail-quantile F t + (x - μ) * f (tail-quantile F t)))
 = t * (1 - F (A t * x + B t))
    using eventually-gt-at-top[of 1::real] by eventually-elim (simp add: argeq)
  show ((λt. t * (1 - F (A t * x + B t))) ⟶ - ln (EVD-cdf γ μ σ x)) at-top
    by (rule Lim-transform-eventually[OF main eqarg])
 qed
 define aa where aa = (λn::nat. A (real n))
 define bb where bb = (λn::nat. B (real n))
 have aa-pos: ∀ n. 0 < aa n
  unfolding aa-def using A-pos by simp
 have main: ∀ x. (λn. F (aa n * x + bb n) ^ n) ⟶ EVD-cdf γ μ σ x
  using MDA-tail-probability-witness[OF F A-pos sigma tail]
  by(auto simp: aa-def bb-def)
 show ?thesis
  using aa-pos main by blast
qed

lemma MDA-iff-divide-tail:
 fixes F :: real ⇒ real and γ :: real
 assumes F: right-continuous-mono F 0 1
 shows domain-of-attraction F (EVD-cdf γ 0 1)
       ⟷ (∃f. (∀ t. 0 < f t) ∧
            (∀ x. 1 + γ * x > 0 ⟶
                ((λt. (1 - F (t + x * f t)) / (1 - F t))
                  ⟶ - ln (EVD-cdf γ 0 1 x)) (endpoint-filter F)))
proof
 assume dom: domain-of-attraction F (EVD-cdf γ 0 1)
 obtain a b where a: ⋀n. 0 < a n
  and weak-conv (λn x. F (a n * x + b n) ^ n) (EVD-cdf γ 0 1)
  using dom unfolding domain-of-attraction-def by blast
 hence wc: ⋀x. (λn. F (a n * x + b n) ^ n) ⟶ EVD-cdf γ 0 1 x
  by(simp add: weak-conv-EVD-cdf-iff)
 define f where f = (λt. a (nat ⌊1 / (1 - F t)⌋))
 have fpos: ⋀t. 0 < f t
  using a by (simp add: f-def)
 have lim: ⋀x. 0 < 1 + γ * x ⟹
       ((λt. (1 - F (t + x * f t)) / (1 - F t))
         ⟶ - ln (EVD-cdf γ 0 1 x)) (endpoint-filter F)
 proof -
  fix x :: real
  assume x: 0 < 1 + γ * x
  have x': 0 < 1 + γ * (x - 0) / 1
   using x by simp
  show ((λt. (1 - F (t + x * f t)) / (1 - F t))
        ⟶ - ln (EVD-cdf γ 0 1 x)) (endpoint-filter F)
   using MDA-imp-divide-tail[OF F a zero-less-one wc x']
   by (simp add: f-def)
```

qed
show $\exists f.\ (\forall t.\ 0 < f\ t) \wedge$
$(\forall x.\ 1 + \gamma * x > 0 \longrightarrow$
$((\lambda t.\ (1 - F\ (t + x * f\ t))\ /\ (1 - F\ t))$
$\longrightarrow - ln\ (EVD\text{-}cdf\ \gamma\ 0\ 1\ x))\ (endpoint\text{-}filter\ F))$
using *fpos lim* **by** *blast*
next
assume $\exists f.\ (\forall t.\ 0 < f\ t) \wedge$
$(\forall x.\ 1 + \gamma * x > 0 \longrightarrow$
$((\lambda t.\ (1 - F\ (t + x * f\ t))\ /\ (1 - F\ t))$
$\longrightarrow - ln\ (EVD\text{-}cdf\ \gamma\ 0\ 1\ x))\ (endpoint\text{-}filter\ F))$
then obtain f **where** *fpos*: $\bigwedge t.\ 0 < f\ t$
and *lim*: $\bigwedge x.\ 1 + \gamma * x > 0 \Longrightarrow$
$((\lambda t.\ (1 - F\ (t + x * f\ t))\ /\ (1 - F\ t))$
$\longrightarrow - ln\ (EVD\text{-}cdf\ \gamma\ 0\ 1\ x))\ (endpoint\text{-}filter\ F)$
by *blast*
have *lim'*: $\bigwedge x.\ 0 < 1 + \gamma * (x - 0)\ /\ 1 \Longrightarrow$
$((\lambda t.\ (1 - F\ (t + (x - 0) * f\ t))\ /\ (1 - F\ t))$
$\longrightarrow - ln\ (EVD\text{-}cdf\ \gamma\ 0\ 1\ x))\ (endpoint\text{-}filter\ F)$
using *lim* **by** *simp*
have *ex*: $\exists a\ b.\ (\forall n.\ 0 < a\ n) \wedge (\forall x.\ (\lambda n.\ F\ (a\ n * x + b\ n)\ \hat{}\ n) \longrightarrow EVD\text{-}cdf\ \gamma\ 0\ 1\ x)$
using *divide-tail-imp-MDA*[**where** $\mu = 0$ **and** $\sigma = 1$ **and** $f = f$ **and** $a = \lambda n{::}nat.\ 1$
,*OF F - - - - lim'*] *fpos*
by *auto*
thus *domain-of-attraction F* (*EVD-cdf* γ *0 1*)
unfolding *domain-of-attraction-def*
using *F right-continuous-mono-EVD-cdf*[*OF zero-less-one*]
by(*auto simp*: *weak-conv-EVD-cdf-iff*)
qed

3.4 von Mises' Conditions

lemma
fixes $F :: real \Rightarrow real$ **and** *x0 t* :: *real*
assumes *F*: *right-continuous-mono F 0 1*
and *x0*: *F x0* < *1*
and *Flim*: (*F* $\longrightarrow$ *1*) (*endpoint-filter F*)
and *Fcont*: $\bigwedge x.\ x0 < x \Longrightarrow F\ x < 1 \Longrightarrow isCont\ F\ x$
and *t*: *1 / (1 − F x0)* < *t*
shows *tail-quantile-tail-eq*: *1 − F* (*tail-quantile F t*) = *1 / t*
and *F-tail-quantile-lt-1*: *F* (*tail-quantile F t*) < *1*
proof −
interpret *right-continuous-mono F 0 1*
by *fact*
have *t1*: *1* < *t*
using *one-less-of-inverse-tail-less*[*OF F x0 t*] **.**
define *p* **where** *p* = *1 − 1 / t*

```
  have p0: 0 < p and p1: p < 1
    using t1 by (simp-all add: p-def)
  have px0: F x0 < p
    using t x0 t1 by (simp add: p-def field-simps)
  have c-gt: x0 < ginv F p
    using ginv-le-iff[OF p0 p1, of x0] px0 by force
  obtain y where y: p < F y F y < 1
    using eventually-happens'[OF endpoint-filter-neq-bot
        eventually-conj[OF order-tendstoD(1)[OF Flim p1] F-lt-1-endpoint[OF F]]]
by blast
  have Fc1: F (ginv F p) < 1
    using ginv-le-iff[OF p0 p1, of y] y monoD[OF mono, of ginv F p y] by force
  have F (ginv F p) ≤ p
  proof (rule tendsto-upperbound[where F = at-left (ginv F p)])
    show (F ⟶ F (ginv F p)) (at-left (ginv F p))
     using Fcont[OF c-gt Fc1] by (simp add: isCont-def Lim-at-imp-Lim-at-within)
    show ∀F z in at-left (ginv F p). F z ≤ p
      by (intro eventually-at-leftI[OF - c-gt])
        (force simp: linorder-not-le[symmetric] ginv-le-iff[OF p0 p1])
  qed (rule trivial-limit-at-left-real)
  thus eq: 1 − F (tail-quantile F t) = 1 / t
    using ginv-attained[OF p0 p1] unfolding p-def by linarith
  show F (tail-quantile F t) < 1
    using eq zero-less-divide-1-iff[of t] t1 by linarith
qed

context
  fixes F f :: real ⇒ real and x0 :: real
  assumes F: right-continuous-mono F 0 1
    and x0: F x0 < 1
    and Flim: (F ⟶ 1) (endpoint-filter F)
    and Fderiv: ⋀y. x0 < y ⟹ F y < 1 ⟹ (F has-real-derivative f y) (at y)
    and fpos: ⋀y. x0 < y ⟹ F y < 1 ⟹ 0 < f y
begin

lemma tail-quantile-has-deriv:
  fixes t :: real
  assumes t: 1 / (1 − F x0) < t
  shows (tail-quantile F has-real-derivative
          1 / (t² * f (tail-quantile F t))) (at t)
proof −
  interpret right-continuous-mono F 0 1
    by fact
  define T where T = 1 / (1 − F x0)
  define U where U = tail-quantile F
  define phi where phi = (λs. 1 / (1 − F s))
  have smono: F v < F w if x0 < v v < w F w < 1 for v w
    by (rule strict-mono-of-deriv[OF Fderiv fpos that])
  have s1: 1 < s if T < s for s
```

```
    using one-less-of-inverse-tail-less[OF F x0] that by (simp add: T-def)
  have Ugt: x0 < U s if T < s for s
    using t-lt-tail-quantile-of-gt-inv[OF F x0] that by (simp add: U-def T-def)
  have UF: 1 − F (U s) = 1 / s if T < s for s
    using tail-quantile-tail-eq[OF F x0 Flim DERIV-isCont[OF Fderiv]] that
    by (simp add: U-def T-def)
  have UFlt: F (U s) < 1 if T < s for s
    using F-tail-quantile-lt-1[OF F x0 Flim DERIV-isCont[OF Fderiv]] that
    by (simp add: U-def T-def)
  have phi-U: phi (U s) = s if T < s for s
    using UF[OF that] s1[OF that] by (simp add: phi-def)
  have phi-deriv: (phi has-real-derivative f z / (1 − F z)^2) (at z)
    if z: x0 < z F z < 1 for z
    unfolding phi-def using Fderiv[OF z] z
    by (auto intro!: derivative-eq-intros simp: power2-eq-square)
  have t-gt: T < t
    using t by (simp add: T-def)
  obtain w where wgt: U t < w and wlt1: F w < 1
    using ex-gt-less UFlt t-gt by blast
  define d where d = min (U t − x0) (w − U t) / 2
  have d0: 0 < d
    using Ugt[OF t-gt] wgt by (simp add: d-def)
  have inreg: x0 < z F z < 1 if zd: |z − U t| ≤ d for z
  proof −
    have dlo: d < U t − x0 and dhi: d < w − U t
      using Ugt[OF t-gt] wgt by (auto simp: d-def)
    show x0 < z
      using zd dlo by (simp add: abs-le-iff)
    have z < w
      using zd dhi by (simp add: abs-le-iff)
    thus F z < 1
      using monoD[OF mono, of z w] wlt1 by simp
  qed
  have isC: isCont U t
  proof (subst phi-U[OF t-gt, symmetric],
        rule isCont-inverse-function[where f = phi and g = U and x = U t and
d = d,
          OF d0])
    fix z
    assume z: |z − U t| ≤ d
    show U (phi z) = z isCont phi z
      using ginv-f-eq[OF smono inreg[OF z]] inreg[OF z]
        DERIV-isCont[OF phi-deriv[OF inreg[OF z]]]
      by (simp-all add: U-def phi-def)
  qed
  have D: (phi has-real-derivative t^2 * f (U t)) (at (U t))
    using UF[OF t-gt] s1[OF t-gt] phi-deriv[OF Ugt[OF t-gt] UFlt[OF t-gt]]
    by (simp add: power-one-over mult.commute)
  have (U has-real-derivative inverse (t^2 * f (U t))) (at t)
```

```
  proof (rule DERIV-inverse-function[OF D - t-gt less-add-one - isC])
    show t^2 * f (U t) ≠ 0
      using fpos[OF Ugt[OF t-gt] UFlt[OF t-gt]] s1[OF t-gt] by simp
    fix y
    assume T < y y < t + 1
    thus phi (U y) = y
      by (simp add: phi-U)
  qed
  thus ?thesis
    by (simp add: U-def inverse-eq-divide)
qed

lemma f-tail-quantile-pos:
  fixes t :: real
  assumes t: 1 / (1 - F x0) < t
  shows 0 < f (tail-quantile F t)
  by (intro fpos t-lt-tail-quantile-of-gt-inv[OF F x0 t]
      F-tail-quantile-lt-1[OF F x0 Flim DERIV-isCont[OF Fderiv] t])

lemma tail-quantile-deriv-has-deriv:
  fixes f' :: real ⇒ real and t :: real
  assumes fderiv: ⋀x. x0 < x ⟹ F x < 1 ⟹ (f has-real-derivative f' x) (at x)
    and t: 1 / (1 - F x0) < t
  shows ((λs. 1 / (s^2 * f (tail-quantile F s))) has-real-derivative
          - (2 * t * f (tail-quantile F t)
              + f' (tail-quantile F t) / f (tail-quantile F t))
            / (t^2 * f (tail-quantile F t))^2) (at t)
proof -
  define T where T = 1 / (1 - F x0)
  define U where U = tail-quantile F
  have t-gt: T < t
    using t by (simp add: T-def)
  have t1: 1 < t
    using one-less-of-inverse-tail-less[OF F x0 t] .
  have Ugt: x0 < U t
    using t-lt-tail-quantile-of-gt-inv[OF F x0 t] by (simp add: U-def)
  have UFlt: F (U t) < 1
   using F-tail-quantile-lt-1[OF F x0 Flim DERIV-isCont[OF Fderiv] t] by (simp
add: U-def)
  have nz: t ≠ 0 f (U t) ≠ 0 and nz2: t^2 * f (U t) ≠ 0
    using t1 fpos[OF Ugt UFlt] by auto
  have c1: ((λs. f (U s)) has-real-derivative f' (U t) * (1 / (t^2 * f (U t)))) (at t)
    unfolding U-def
    by (rule DERIV-chain2[OF fderiv[OF Ugt UFlt, unfolded U-def]
          tail-quantile-has-deriv[OF t]])
  have c2: ((λs. s^2 * f (U s)) has-real-derivative
              2 * t * f (U t) + f' (U t) / f (U t)) (at t)
  proof -
    have 2 * t * f (U t) + t^2 * (f' (U t) * (1 / (t^2 * f (U t))))
```

```
        = 2 * t * f (U t) + f' (U t) / f (U t)
    using nz by (simp add: field-simps)
  thus ?thesis
    by (auto intro!: derivative-eq-intros c1)
 qed
 have ((λs. 1 / (s² * f (U s))) has-real-derivative
       − (2 * t * f (U t) + f' (U t) / f (U t)) / (t² * f (U t))²) (at t)
  using DERIV-divide[OF DERIV-const[of 1] c2 nz2] by (simp add: power2-eq-square)
 thus ?thesis
   by (simp add: U-def)
qed

lemma tail-quantile-second-deriv-eq:
 fixes f' u u' :: real ⇒ real and t :: real
 assumes fderiv: ⋀x. x0 < x ⟹ F x < 1 ⟹ (f has-real-derivative f' x) (at x)
   and uderiv: ⋀s. 1 / (1 − F x0) < s
             ⟹ (tail-quantile F has-real-derivative u s) (at s)
   and u'deriv: ⋀s. 1 / (1 − F x0) < s ⟹ (u has-real-derivative u' s) (at s)
   and t: 1 / (1 − F x0) < t
 shows t * u' t / u t
       = − 2 − (1 − F (tail-quantile F t)) * f' (tail-quantile F t)
             / (f (tail-quantile F t))²
proof −
 define T where T = 1 / (1 − F x0)
 define U where U = tail-quantile F
 have t-gt: T < t
   using t by (simp add: T-def)
 have s1: 1 < s if T < s for s
   using one-less-of-inverse-tail-less[OF F x0] that by (simp add: T-def)
 have Ugt: x0 < U s if T < s for s
   using t-lt-tail-quantile-of-gt-inv[OF F x0] that by (simp add: U-def T-def)
 have UF: 1 − F (U s) = 1 / s if T < s for s
   using tail-quantile-tail-eq[OF F x0 Flim DERIV-isCont[OF Fderiv]] that
   by (simp add: U-def T-def)
 have UFlt: F (U s) < 1 if T < s for s
   using F-tail-quantile-lt-1[OF F x0 Flim DERIV-isCont[OF Fderiv]] that
   by (simp add: U-def T-def)
 have fUpos: 0 < f (U s) if T < s for s
   using f-tail-quantile-pos that by (simp add: T-def U-def)
 have u-eq: u s = 1 / (s² * f (U s)) if T < s for s
   using DERIV-unique[OF uderiv tail-quantile-has-deriv] that
   by (simp add: T-def U-def)
 have hd: ((λs. 1 / (s² * f (U s))) has-real-derivative
     − (2 * t * f (U t) + f' (U t) / f (U t)) / (t² * f (U t))²) (at t)
   using tail-quantile-deriv-has-deriv[OF fderiv t]
   by (simp add: U-def)
 have uderiv': (u has-real-derivative
     − (2 * t * f (U t) + f' (U t) / f (U t)) / (t² * f (U t))²) (at t)
  by(intro has-field-derivative-transform-within-open[OF hd, where S = {T<..}])
```

```
    (auto simp: u-eq t-gt)
  have Uval: tail-quantile F t = U t
    by (simp add: U-def)
  have nz': t ≠ 0 f (U t) ≠ 0
    using s1[OF t-gt] fUpos[OF t-gt] by auto
  show ?thesis
    unfolding Uval DERIV-unique[OF u'deriv[OF t] uderiv'] u-eq[OF t-gt] UF[OF
t-gt]
    using nz' by (simp add: field-simps power2-eq-square)
qed

context
  fixes f' :: real ⇒ real and γ :: real
  assumes fderiv: ⋀y. x0 < y ⟹ F y < 1 ⟹ (f has-real-derivative f' y) (at y)
    and f'cont: continuous-on {y. x0 < y ∧ F y < 1} f'
    and lim: ((λy. (1 − F y) ∗ f' y / (f y)²) ⟶ − γ − 1) (endpoint-filter F)
begin

lemma von-Mises-quantile-diff:
  fixes x :: real
  assumes x: 0 < x
  shows ((λt. (tail-quantile F (t ∗ x) − tail-quantile F t)
                ∗ (t ∗ f (tail-quantile F t)))
          ⟶ (if γ = 0 then ln x else (x powr γ − 1) / γ)) at-top
proof −
  define T where T = 1 / (1 − F x0)
  define U where U = tail-quantile F
  define u where u = (λs. 1 / (s² ∗ f (U s)))
  define u' where u' = (λs. − (2 ∗ s ∗ f (U s) + f' (U s) / f (U s)) / (s² ∗ f (U
s))²)
  have Tpos: 0 < T
    using x0 right-continuous-mono.a-le-f[OF F, of x0] by (simp add: T-def)
  have s1: 1 < s if T < s for s
    using one-less-of-inverse-tail-less[OF F x0] that by (simp add: T-def)
  have Ugt: x0 < U s if T < s for s
    using t-lt-tail-quantile-of-gt-inv[OF F x0] that by (simp add: U-def T-def)
  have UF: 1 − F (U s) = 1 / s if T < s for s
    using tail-quantile-tail-eq[OF F x0 Flim DERIV-isCont[OF Fderiv]] that
    by (simp add: U-def T-def)
  have UFlt: F (U s) < 1 if T < s for s
    using F-tail-quantile-lt-1[OF F x0 Flim DERIV-isCont[OF Fderiv]] that
    by (simp add: U-def T-def)
  have fUpos: 0 < f (U s) if T < s for s
    using f-tail-quantile-pos that by (simp add: T-def U-def)
  have UderivT: (tail-quantile F has-real-derivative u s) (at s)
    if 1 / (1 − F x0) < s for s
    using tail-quantile-has-deriv[OF that] by (simp add: u-def U-def)
  have uderivT: (u has-real-derivative u' s) (at s) if 1 / (1 − F x0) < s for s
    using tail-quantile-deriv-has-deriv[OF fderiv that]
```

```
  by (simp add: u-def u'-def U-def)
 have Uderiv: (U has-real-derivative u s) (at s) if T < s for s
  using UderivT that by (simp add: T-def U-def)
 have uderiv: (u has-real-derivative u' s) (at s) if T < s for s
  using uderivT that by (simp add: T-def)
 have upos: 0 < u s if T < s for s
  using s1[OF that] fUpos[OF that] by (simp add: u-def)
 have regopen: open {y. x0 < y ∧ F y < 1}
 proof −
  have {y. x0 < y ∧ F y < 1} = {x0<..} ∩ {y. F y < 1}
   by auto
  moreover have open ({x0<..} ∩ {y. F y < 1})
   by (intro open-Int open-greaterThan right-continuous-mono.open-less[OF F])
  ultimately show ?thesis
   by simp
 qed
 have f'isCont: isCont f' y if x0 < y F y < 1 for y
   by (rule continuous-on-interior[OF f'cont]) (simp add: interior-open[OF re-
gopen] that)
 have u'cont: continuous-on {T<..} u'
 proof (safe intro!: continuous-at-imp-continuous-on)
  fix s
  assume sT: T < s
  show isCont u' s
   unfolding u'-def using fUpos[OF sT] s1[OF sT]
   by (intro continuous-intros
       isCont-o2[OF DERIV-isCont[OF Uderiv[OF sT]]
        DERIV-isCont[OF fderiv[OF Ugt[OF sT] UFlt[OF sT]]]]
       isCont-o2[OF DERIV-isCont[OF Uderiv[OF sT]]
        f'isCont[OF Ugt[OF sT] UFlt[OF sT]]])
      auto
 qed
 have lim2: ((λs. s * u' s / u s) ⟶ γ − 1) at-top
 proof −
  have ((λs. (1 − F (U s)) * f' (U s) / (f (U s))²) ⟶ − γ − 1) at-top
   unfolding U-def
   by (rule filterlim-compose[OF lim tail-quantile-tendsto-endpoint[OF F Flim]])
  from tendsto-diff[OF tendsto-const[of − 2 :: real] this]
  have ((λs. − 2 − (1 − F (U s)) * f' (U s) / (f (U s))²) ⟶ γ − 1) at-top
   by simp
  moreover have ∀F s in at-top.
     − 2 − (1 − F (U s)) * f' (U s) / (f (U s))² = s * u' s / u s
   using eventually-gt-at-top[of T]
   by eventually-elim
    (use tail-quantile-second-deriv-eq[OF fderiv UderivT uderivT]
     in simp add: T-def U-def)
  ultimately show ?thesis
   by (rule Lim-transform-eventually)
 qed
```

```
  have ∀F s in at-top. (U (s * x) - U s) / (s * u s)
        = (U (s * x) - U s) * (s * f (U s))
    using eventually-gt-at-top[of T]
   by eventually-elim (use s1 fUpos in simp add: u-def field-simps power2-eq-square)
  from Lim-transform-eventually
     [OF deriv-diff-ratio-regularly-varying[OF Tpos Uderiv uderiv u'cont upos lim2
x] this]
  show ?thesis
    by (simp add: U-def)
qed

lemma von-Mises-normalizing-ex:
  shows ∃ a. (∀ t. 0 < a t)
          ∧ (∀F t in at-top. a t = 1 / (t * f (tail-quantile F t)))
          ∧ (∀ x>0. ((λt. (tail-quantile F (t * x) - tail-quantile F t) / a t)
                    ⟶ (if γ = 0 then ln x else (x powr γ - 1) / γ)) at-top)
proof -
  define T where T = 1 / (1 - F x0)
  define a where a = (λt. if T < t then 1 / (t * f (tail-quantile F t)) else 1)
  have t1: 1 < t if T < t for t
    using one-less-of-inverse-tail-less[OF F x0] that by (simp add: T-def)
  have fU: 0 < f (tail-quantile F t) if T < t for t
    using f-tail-quantile-pos that by (simp add: T-def)
  have apos: 0 < a t for t
    by (cases T < t) (use t1[of t] fU[of t] in auto simp: a-def)
  have aev: ∀F t in at-top. a t = 1 / (t * f (tail-quantile F t))
    using eventually-gt-at-top[of T] by eventually-elim (simp add: a-def)
  have ((λt. (tail-quantile F (t * x) - tail-quantile F t) / a t)
        ⟶ (if γ = 0 then ln x else (x powr γ - 1) / γ)) at-top if x: 0 < x for
x
  proof -
    have ∀F t in at-top. (tail-quantile F (t * x) - tail-quantile F t)
           * (t * f (tail-quantile F t))
         = (tail-quantile F (t * x) - tail-quantile F t) / a t
      using eventually-gt-at-top[of T]
      by eventually-elim (use t1 fU in simp add: a-def)
    from Lim-transform-eventually[OF von-Mises-quantile-diff[OF x] this]
    show ?thesis .
  qed
  thus ?thesis
    using apos aev by blast
qed
```

von Mises' condition (Theorem 1.1.8 of the book by de Haan and Ferreira [2]).

```
theorem von-Mises-condition: domain-of-attraction F (EVD-cdf γ 0 1)
  unfolding MDA-iff-tail-quantile'[OF F]
  using von-Mises-normalizing-ex by blast
```

corollary *von-Mises-normalization*:
fixes $x :: real$
shows $(\lambda n.\ F\ (x\ /\ (real\ n * f\ (tail\text{-}quantile\ F\ (real\ n)))$
$+\ tail\text{-}quantile\ F\ (real\ n))\ \hat{}\ n) \longrightarrow EVD\text{-}cdf\ \gamma\ 0\ 1\ x$
proof −
obtain a **where** *apos*: $\bigwedge t.\ 0 < a\ t$
and *aev*: $\forall_F\ t\ in\ at\text{-}top.\ a\ t = 1\ /\ (t * f\ (tail\text{-}quantile\ F\ t))$
and *quant*: $\bigwedge y.\ 0 < y \Longrightarrow ((\lambda t.\ (tail\text{-}quantile\ F\ (t * y) - tail\text{-}quantile\ F\ t)\ /\ a\ t)$
$\longrightarrow (if\ \gamma = 0\ then\ ln\ y\ else\ (y\ powr\ \gamma - 1)\ /\ \gamma))\ at\text{-}top$
using *von-Mises-normalizing-ex* **by** *blast*
have $(\lambda n.\ F\ (a\ (real\ n) * x + tail\text{-}quantile\ F\ (real\ n))\ \hat{}\ n) \longrightarrow EVD\text{-}cdf\ \gamma\ 0\ 1\ x$
proof (*rule MDA-tail-quantile-witness*[*OF F apos zero-less-one*])
fix $y :: real$
assume *y*: $0 < y$
show $((\lambda t.\ (tail\text{-}quantile\ F\ (t * y) - tail\text{-}quantile\ F\ t)\ /\ a\ t)$
$\longrightarrow (if\ \gamma = 0\ then\ 1 * ln\ y + 0\ else\ 1 * (y\ powr\ \gamma - 1)\ /\ \gamma + 0))\ at\text{-}top$
by (*cases* $\gamma = 0$) (*use quant*[*OF y*] **in** *simp-all*)
qed
moreover have $\forall_F\ n\ in\ sequentially.\ F\ (a\ (real\ n) * x + tail\text{-}quantile\ F\ (real\ n))\ \hat{}\ n$
$= F\ (x\ /\ (real\ n * f\ (tail\text{-}quantile\ F\ (real\ n))) + tail\text{-}quantile\ F\ (real\ n))\ \hat{}\ n$
using *eventually-compose-filterlim*[*OF aev filterlim-real-sequentially*]
by *eventually-elim simp*
ultimately show *?thesis*
by (*rule Lim-transform-eventually*)
qed

end

end

Relation (1.1.35) of the book by de Haan and Ferreira [2].

lemma *tail-quantile-deriv-ratio-pos*:
fixes $F\ f\ u :: real \Rightarrow real$ **and** $\gamma\ x0\ t0 :: real$
assumes *F*: *right-continuous-mono F 0 1*
and *unbdd*: $\neg$ *bdd-support F*
and *gamma*: $0 < \gamma$
and *Fderiv*: $\bigwedge x.\ x0 < x \Longrightarrow (F\ has\text{-}real\text{-}derivative\ f\ x)\ (at\ x)$
and *fpos*: $\bigwedge x.\ x0 < x \Longrightarrow 0 < f\ x$
and *uderiv*: $\bigwedge t.\ t0 < t \Longrightarrow (tail\text{-}quantile\ F\ has\text{-}real\text{-}derivative\ u\ t)\ (at\ t)$
and *lim*: $((\lambda t.\ t * f\ t\ /\ (1 - F\ t)) \longrightarrow 1\ /\ \gamma)\ at\text{-}top$
shows $((\lambda t.\ t * u\ t\ /\ tail\text{-}quantile\ F\ t) \longrightarrow \gamma)\ at\text{-}top$
proof −
have *x0lt*: $F\ x0 < 1$
by (*rule F-lt-1-unbdd-support*[*OF F unbdd*])
have *Flim*: $(F \longrightarrow 1)\ (endpoint\text{-}filter\ F)$

```
    using right-continuous-mono.top[OF F] unbdd by (simp add: endpoint-filter)
  have Fderiv': (F has-real-derivative f y) (at y) if y: x0 < y and F y < 1 for y
    by (rule Fderiv[OF y])
  have fpos': 0 < f y if y: x0 < y and F y < 1 for y
    by (rule fpos[OF y])
  define T where T = max t0 (1 / (1 - F x0))
  have t1: 1 < t if T < t for t
    using one-less-of-inverse-tail-less[OF F x0lt] that by (simp add: T-def)
  have fU: 0 < f (tail-quantile F t) if T < t for t
   using f-tail-quantile-pos[OF F x0lt Flim Fderiv' fpos'] that by (simp add: T-def)
  have UF: 1 - F (tail-quantile F t) = 1 / t if T < t for t
    using tail-quantile-tail-eq[OF F x0lt Flim DERIV-isCont[OF Fderiv']] that
    by (simp add: T-def)
  have u-eq: u t = 1 / (t² * f (tail-quantile F t)) if T < t for t
    using DERIV-unique[OF uderiv tail-quantile-has-deriv[OF F x0lt Flim Fderiv'
fpos']] that
    by (simp add: T-def)
  note Uat = tail-quantile-at-top[OF F F-lt-1-unbdd-support[OF F unbdd]]
  have ((λs. (1 - F s) / (s * f s)) ⟶ γ) at-top
    using tendsto-inverse[OF lim] gamma by simp
  from filterlim-compose[OF this Uat]
  have comp: ((λt. (1 - F (tail-quantile F t))
                  / (tail-quantile F t * f (tail-quantile F t))) ⟶ γ) at-top .
  have evU: ∀F t in at-top. 1 ≤ tail-quantile F t
    using Uat unfolding filterlim-at-top by blast
  have ∀F t in at-top. (1 - F (tail-quantile F t))
        / (tail-quantile F t * f (tail-quantile F t)) = t * u t / tail-quantile F t
    using eventually-gt-at-top[of T] evU
  proof eventually-elim
    fix t :: real
    assume tT: T < t
      and U1: 1 ≤ tail-quantile F t
    show (1 - F (tail-quantile F t)) / (tail-quantile F t * f (tail-quantile F t))
          = t * u t / tail-quantile F t
      unfolding UF[OF tT] u-eq[OF tT]
      using t1[OF tT] fU[OF tT] U1 by (simp add: field-simps power2-eq-square)
  qed
  from Lim-transform-eventually[OF comp this]
  show ?thesis .
qed
```

von Mises' condition for positive γ (Theorem 1.1.11 of the book by de Haan and Ferreira [2]).

```
theorem von-Mises-condition-pos:
  fixes F f :: real ⇒ real and γ x0 :: real
  assumes F: right-continuous-mono F 0 1
    and unbdd: ¬ bdd-support F
    and gamma: 0 < γ
    and Fderiv: ⋀x. x0 < x ⟹ (F has-real-derivative f x) (at x)
```

```
    and fcont: continuous-on {x0<..} f
    and fpos: ⋀x. x0 < x ⟹ 0 < f x
    and lim: ((λt. t * f t / (1 - F t)) ⟶ 1 / γ) at-top
  shows domain-of-attraction F (EVD-cdf γ 0 1)
proof -
  have x0lt: F x0 < 1
    by (rule F-lt-1-unbdd-support[OF F unbdd])
  have Flim: (F ⟶ 1) (endpoint-filter F)
    using right-continuous-mono.top[OF F] unbdd by (simp add: endpoint-filter)
  have Fderiv': (F has-real-derivative f y) (at y) if y: x0 < y and F y < 1 for y
    by (rule Fderiv[OF y])
  have fpos': 0 < f y if y: x0 < y and F y < 1 for y
    by (rule fpos[OF y])
  have fcont': isCont f y if x0 < y for y
    by (rule continuous-on-interior[OF fcont]) (simp add: interior-open that)
  obtain T1 where T1: ⋀t. T1 ≤ t ⟹ 1 ≤ tail-quantile F t
    using tail-quantile-at-top[OF F F-lt-1-unbdd-support[OF F unbdd]]
    unfolding filterlim-at-top eventually-at-top-linorder by blast
  define T where T = max (1 / (1 - F x0)) (max T1 1)
  define u where u = (λs. 1 / (s² * f (tail-quantile F s)))
  have Tpos: 0 < T
    by (simp add: T-def)
  have t1: 1 < t if T < t for t
    using one-less-of-inverse-tail-less[OF F x0lt] that by (simp add: T-def)
  have Upos: 0 < tail-quantile F t if T < t for t
    using T1[of t] that by (simp add: T-def)
  have fU: 0 < f (tail-quantile F t) if T < t for t
   using f-tail-quantile-pos[OF F x0lt Flim Fderiv' fpos'] that by (simp add: T-def)
  have Uderiv: (tail-quantile F has-real-derivative u t) (at t) if T < t for t
    using tail-quantile-has-deriv[OF F x0lt Flim Fderiv' fpos'] that by (simp add:
u-def T-def)
  have ucont: continuous-on {T<..} u
  proof (safe intro!: continuous-at-imp-continuous-on)
    fix s
    assume sT: T < s
    show isCont u s
      unfolding u-def using fU[OF sT] t1[OF sT]
      by (intro continuous-intros
          isCont-o2[OF DERIV-isCont[OF Uderiv[OF sT]]
            fcont'[OF t-lt-tail-quantile-of-gt-inv[OF F x0lt, of s]]])
         (use sT in auto simp: T-def)
  qed
  note lim2 = tail-quantile-deriv-ratio-pos[OF F unbdd gamma Fderiv fpos Uderiv
lim]
  define a where a = (λt. if T < t then γ * tail-quantile F t else 1)
  have apos: 0 < a t for t
    by (cases T < t) (use gamma Upos in simp-all add: a-def)
  have ((λt. (tail-quantile F (t * x) - tail-quantile F t) / a t)
        ⟶ (if γ = 0 then ln x else (x powr γ - 1) / γ)) at-top if x: 0 < x for
```

```
x
  proof −
    have ((λt. (tail-quantile F (t ∗ x) / tail-quantile F t − 1) / γ)
          ⟶ (x powr γ − 1) / γ) at-top
      by (intro tendsto-divide tendsto-diff
           deriv-ratio-regularly-varying[OF Tpos Uderiv ucont Upos lim2 x] tend-
sto-const)
         (use gamma in simp-all)
    moreover have ∀F t in at-top. (tail-quantile F (t ∗ x) / tail-quantile F t −
1) / γ
          = (tail-quantile F (t ∗ x) − tail-quantile F t) / a t
      using eventually-gt-at-top[of T]
    proof eventually-elim
      case (elim t)
      show ?case
        using Upos[OF elim] gamma elim by (simp add: a-def field-simps)
    qed
    ultimately have ((λt. (tail-quantile F (t ∗ x) − tail-quantile F t) / a t)
          ⟶ (x powr γ − 1) / γ) at-top
      by (rule Lim-transform-eventually)
    thus ?thesis
      using gamma by simp
  qed
  thus ?thesis
    unfolding MDA-iff-tail-quantile'[OF F] using apos by blast
qed

lemma F-tendsto-1-endpoint-of-deriv:
  fixes F f :: real ⇒ real and x0 c :: real
  assumes F: right-continuous-mono F 0 1
    and bdd: bdd-support F
    and x0: x0 < x-star F
    and Fderiv: ⋀y. x0 < y ⟹ y < x-star F ⟹ (F has-real-derivative f y) (at y)
    and cpos: 0 < c
    and lim: ((λy. (x-star F − y) ∗ f y / (1 − F y)) ⟶ c) (at-left (x-star F))
  shows (F ⟶ 1) (endpoint-filter F)
proof −
  interpret right-continuous-mono F 0 1
    by fact
  define X where X = x-star F
  define G where G = Sup (F ‘ {..<X})
  have ne: F ‘ {..<X} ≠ {}
    by auto
  have leG: F y ≤ G if y < X for y
    unfolding G-def using that by (auto intro!: cSup-upper bdd-aboveI2[of - - 1]
f-le-b)
  have G1: G = 1
  proof (rule ccontr)
    assume G ≠ 1
```

moreover have $G \leq 1$
unfolding *G-def* **by** (*auto intro*!: *cSup-least ne f-le-b*)
ultimately have *Glt*: $G < 1$
by *simp*
define e **where** $e = 1 - G$
have *epos*: $0 < e$
using *Glt* **by** (*simp add*: *e-def*)
have *tail-ge*: $e \leq 1 - F\ y$ **if** $y < X$ **for** y
using *leG*[*OF that*] **by** (*simp add*: *e-def*)
obtain *b0* **where** *b0X*: $b0 < X$
and *b0*: $\bigwedge y.\ b0 < y \Longrightarrow y < X \Longrightarrow c\ /\ 2 < (X - y) * f\ y\ /\ (1 - F\ y)$
using *order-tendstoD*(*1*)[*OF lim*, *of c / 2*] *cpos*
unfolding *X-def eventually-at-left-field* **by** *auto*
define B **where** $B = max\ b0\ x0$
have *BX*: $B < X$
using *b0X x0* **by** (*simp add*: *B-def X-def*)
have *fbound*: $c * e\ /\ 2\ /\ (X - y) < f\ y$ **if** *yB*: $B < y$ **and** *yX*: $y < X$ **for** y
proof −
have *pos*: $0 < 1 - F\ y$
using *tail-ge*[*OF yX*] *epos* **by** *simp*
have $c\ /\ 2 * e \leq c\ /\ 2 * (1 - F\ y)$
using *tail-ge*[*OF yX*] *cpos* **by** *simp*
also have $\ldots < (X - y) * f\ y$
using *b0*[*of y*] *yB yX pos* **by** (*simp add*: *B-def pos-less-divide-eq*)
finally show *?thesis*
using *yX* **by** (*simp add*: *pos-divide-less-eq algebra-simps*)
qed
obtain s **where** *sX*: $s < X$ **and** *Fs*: $G - c * e\ /\ 4 < F\ s$
using *less-cSupD*[*OF ne*, *of G − c * e / 4*] *cpos epos* **by** (*auto simp*: *G-def*)
define p **where** $p = max\ s\ ((B + X)\ /\ 2)$
have *pX*: $p < X$ **and** *Bp*: $B < p$ **and** *sp*: $s \leq p$
using *sX BX* **by** (*auto simp*: *p-def less-max-iff-disj*)
have *Fp*: $G - c * e\ /\ 4 < F\ p$
using *Fs monoD*[*OF mono*, *of s p*] *sp* **by** *simp*
define q **where** $q = (p + X)\ /\ 2$
have *pq*: $p < q$ **and** *qX*: $q < X$
using *pX* **by** (*simp-all add*: *q-def*)
have *deriv*: $(F\ has\text{-}real\text{-}derivative\ f\ y)\ (at\ y)$ **if** $p \leq y\ y \leq q$ **for** y
using *that Bp qX x0* **by** (*auto intro*!: *Fderiv simp*: *B-def X-def*)
obtain z **where** *z*: $p < z\ z < q$ **and** *Fz*: $F\ q - F\ p = (q - p) * f\ z$
using *MVT2*[*OF pq deriv*] **by** *blast*
have $c * e\ /\ 4 < (q - p) * f\ z$
proof −
have $c * e\ /\ 2\ /\ (X - p) \leq c * e\ /\ 2\ /\ (X - z)$
using *z pX qX cpos epos* **by** (*intro divide-left-mono*) *auto*
also have $\ldots < f\ z$
using *fbound*[*of z*] *Bp z qX* **by** *simp*
finally show *?thesis*
using *pX* **by** (*simp add*: *q-def field-simps*)

```
    qed
    thus False
      using Fz Fp leG[OF qX] by simp
  qed
  show ?thesis
    unfolding endpoint-filter(1)[OF bdd] X-def[symmetric]
  proof (rule order-tendstoI)
    fix a :: real
    assume a < 1
    then obtain s where sX: s < X and Fs: a < F s
      using less-cSupD[OF ne, of a] by (auto simp: G-def G1[symmetric])
    show ∀ F y in at-left X. a < F y
    proof (intro eventually-at-leftI[OF - sX])
      fix y
      assume y ∈ {s<..<X}
      thus a < F y
        using Fs monoD[OF mono, of s y] by simp
    qed
  qed (auto intro!: always-eventually order.strict-trans1[OF f-le-b])
qed
```

Relation (1.1.38) of the book by de Haan and Ferreira [2].

```
lemma tail-quantile-deriv-ratio-neg:
  fixes F f u :: real ⇒ real and γ x0 t0 :: real
  assumes F: right-continuous-mono F 0 1
    and bdd: bdd-support F
    and gamma: γ < 0
    and x0: x0 < x-star F
    and Fderiv: ⋀x. x0 < x ⟹ x < x-star F ⟹ (F has-real-derivative f x) (at x)
    and fpos: ⋀x. x0 < x ⟹ x < x-star F ⟹ 0 < f x
    and uderiv: ⋀t. t0 < t ⟹ (tail-quantile F has-real-derivative u t) (at t)
    and lim: ((λt. (x-star F − t) ∗ f t / (1 − F t)) ⟶ − 1 / γ) (at-left (x-star
F))
  shows ((λt. t ∗ u t / (x-star F − tail-quantile F t)) ⟶ − γ) at-top
proof −
  have Flim: (F ⟶ 1) (endpoint-filter F)
    by (intro F-tendsto-1-endpoint-of-deriv[OF F bdd x0 Fderiv - lim])
       (use gamma in auto simp: divide-neg-neg)
  have iff: y < x-star F ⟷ F y < 1 for y
    by (rule lt-x-star-iff-F-lt-1[OF F bdd])
  have x0lt: F x0 < 1
    using x0 iff by simp
  have Fderiv': (F has-real-derivative f y) (at y) if x0 < y F y < 1 for y
    using that iff by (auto intro!: Fderiv)
  have fpos': 0 < f y if x0 < y F y < 1 for y
    using that iff by (auto intro!: fpos)
  define T where T = max t0 (1 / (1 − F x0))
  have t1: 1 < t if T < t for t
    using one-less-of-inverse-tail-less[OF F x0lt] that by (simp add: T-def)
```

```
  have fU: 0 < f (tail-quantile F t) if T < t for t
   using f-tail-quantile-pos[OF F x0lt Flim Fderiv' fpos'] that by (simp add: T-def)
  have UF: 1 − F (tail-quantile F t) = 1 / t if T < t for t
    using tail-quantile-tail-eq[OF F x0lt Flim DERIV-isCont[OF Fderiv']] that
    by (simp add: T-def)
  have u-eq: u t = 1 / (t² * f (tail-quantile F t)) if T < t for t
    using DERIV-unique[OF uderiv tail-quantile-has-deriv[OF F x0lt Flim Fderiv'
fpos']] that
    by (simp add: T-def)
  have Uend: filterlim (tail-quantile F) (at-left (x-star F)) at-top
    using tail-quantile-tendsto-endpoint[OF F Flim] by (simp add: endpoint-filter
bdd)
  have ((λs. (1 − F s) / ((x-star F − s) * f s)) ⟶ − γ) (at-left (x-star F))
    using tendsto-inverse[OF lim] gamma by simp
  from filterlim-compose[OF this Uend]
  have comp: ((λt. (1 − F (tail-quantile F t))
                  / ((x-star F − tail-quantile F t) * f (tail-quantile F t)))
              ⟶ − γ) at-top .
  have evU: ∀F t in at-top. tail-quantile F t < x-star F
    using eventually-gt-at-top[of 1 :: real]
    by eventually-elim (rule tail-quantile-lt-x-star[OF F bdd Flim])
  have ∀F t in at-top. (1 − F (tail-quantile F t))
         / ((x-star F − tail-quantile F t) * f (tail-quantile F t))
       = t * u t / (x-star F − tail-quantile F t)
    using eventually-gt-at-top[of T] evU
  proof eventually-elim
    fix t :: real
    assume tT: T < t
      and Ult: tail-quantile F t < x-star F
    show (1 − F (tail-quantile F t))
           / ((x-star F − tail-quantile F t) * f (tail-quantile F t))
         = t * u t / (x-star F − tail-quantile F t)
      unfolding UF[OF tT] u-eq[OF tT]
      using t1[OF tT] fU[OF tT] Ult by (simp add: field-simps power2-eq-square)
  qed
  from Lim-transform-eventually[OF comp this]
  show ?thesis .
qed
```

von Mises' condition for negative γ (Theorem 1.1.13 of the book by de Haan and Ferreira [2]).

```
theorem von-Mises-condition-neg:
  fixes F f :: real ⇒ real and γ x0 :: real
  assumes F: right-continuous-mono F 0 1
    and bdd: bdd-support F
    and gamma: γ < 0
    and x0: x0 < x-star F
    and Fderiv: ⋀x. x0 < x ⟹ x < x-star F ⟹ (F has-real-derivative f x) (at x)
    and fcont: continuous-on {x0<..<x-star F} f
```

```
    and fpos: ⋀x. x0 < x ⟹ x < x-star F ⟹ 0 < f x
    and lim: ((λt. (x-star F − t) * f t / (1 − F t)) ⟶ − 1 / γ) (at-left (x-star
F))
  shows domain-of-attraction F (EVD-cdf γ 0 1)
proof −
  have Flim: (F ⟶ 1) (endpoint-filter F)
    by (intro F-tendsto-1-endpoint-of-deriv[OF F bdd x0 Fderiv - lim])
       (use gamma in auto simp: divide-neg-neg)
  have iff: y < x-star F ⟷ F y < 1 for y
    by (rule lt-x-star-iff-F-lt-1[OF F bdd])
  have x0lt: F x0 < 1
    using x0 iff by simp
  have Fderiv': (F has-real-derivative f y) (at y) if x0 < y F y < 1 for y
    using that iff by (auto intro!: Fderiv)
  have fpos': 0 < f y if x0 < y F y < 1 for y
    using that iff by (auto intro!: fpos)
  have fcont': isCont f y if x0 < y y < x-star F for y
    by (rule continuous-on-interior[OF fcont]) (simp add: interior-open that)
  define T where T = max (1 / (1 − F x0)) 1
  define u where u = (λs. 1 / (s² * f (tail-quantile F s)))
  have Tpos: 0 < T
    by (simp add: T-def)
  have t1: 1 < t if T < t for t
    using one-less-of-inverse-tail-less[OF F x0lt] that by (simp add: T-def)
  have Ult: tail-quantile F t < x-star F if T < t for t
    by (rule tail-quantile-lt-x-star[OF F bdd Flim t1[OF that]])
  have Ugt: x0 < tail-quantile F t if T < t for t
    using t-lt-tail-quantile-of-gt-inv[OF F x0lt] that by (simp add: T-def)
  have fU: 0 < f (tail-quantile F t) if T < t for t
    by (rule fpos[OF Ugt[OF that] Ult[OF that]])
  have Uderiv: (tail-quantile F has-real-derivative u t) (at t) if T < t for t
    using tail-quantile-has-deriv[OF F x0lt Flim Fderiv' fpos'] that by (simp add:
u-def T-def)
  have ucont: continuous-on {T<..} u
  proof (safe intro!: continuous-at-imp-continuous-on)
    fix s
    assume sT: T < s
    show isCont u s
      unfolding u-def using fU[OF sT] t1[OF sT]
      by (intro continuous-intros
            isCont-o2[OF DERIV-isCont[OF Uderiv[OF sT]] fcont'[OF Ugt[OF sT]
Ult[OF sT]]])
         auto
  qed
  note lim2 = tail-quantile-deriv-ratio-neg[OF F bdd gamma x0 Fderiv fpos Uderiv
lim]
  define a where a = (λt. if T < t then − γ * (x-star F − tail-quantile F t) else
1)
  have apos: 0 < a t for t
```

```
    by (cases T < t) (use gamma Ult in simp-all add: a-def mult-neg-pos)
  have ((λt. (tail-quantile F (t * x) − tail-quantile F t) / a t)
        ⟶ (if γ = 0 then ln x else (x powr γ − 1) / γ)) at-top if x: 0 < x for
x
  proof −
    have ((λt. (1 − (x-star F − tail-quantile F (t * x))
                    / (x-star F − tail-quantile F t)) / (− γ))
          ⟶ (1 − x powr γ) / (− γ)) at-top
      by (intro tendsto-divide tendsto-diff
           deriv-ratio-regularly-varying-uminus[OF Tpos Uderiv ucont Ult lim2 x]
tendsto-const)
         (use gamma in simp-all)
    moreover have ∀F t in at-top. (1 − (x-star F − tail-quantile F (t * x))
                    / (x-star F − tail-quantile F t)) / (− γ)
          = (tail-quantile F (t * x) − tail-quantile F t) / a t
      using eventually-gt-at-top[of T]
    proof eventually-elim
      case (elim t)
      show ?case
        using Ult[OF elim] gamma elim by (simp add: a-def field-simps)
    qed
    ultimately have ((λt. (tail-quantile F (t * x) − tail-quantile F t) / a t)
          ⟶ (1 − x powr γ) / (− γ)) at-top
      by (rule Lim-transform-eventually)
    thus ?thesis
      using gamma by (simp add: minus-divide-left field-simps)
  qed
  thus ?thesis
    unfolding MDA-iff-tail-quantile'[OF F] using apos by blast
qed
```

3.5 Relation to Regularly Varying Functions

We formalize results for $\gamma \neq 0$.

```
lemma tail-quantile-S-at-top:
  fixes F :: real ⇒ real and ϱ :: real
  assumes F: right-continuous-mono F 0 1
    and bdd: bdd-support F
    and rv: regularly-varying (λs. 1 − F (x-star F − 1 / s)) ϱ
    and rho: ϱ < 0
  shows filterlim (λt. 1 / (x-star F − tail-quantile F t)) at-top at-top
proof −
  interpret right-continuous-mono F 0 1
    by fact
  have phi-to-0: ((λs. 1 − F (x-star F − 1 / s)) ⟶ 0) at-top
    using regularly-varying-neg-at-top[OF rv rho] .
  have Ult: tail-quantile F t < x-star F
    if t1[arith]: 1 < t for t
  proof −
```

```
  have ev: ∀F s in at-top. 1 − F (x-star F − 1 / s) < 1 / t ∧ 0 < s
    using eventually-conj[OF order-tendstoD(2)[OF phi-to-0, of 1 / t]
          eventually-gt-at-top[of 0::real]]
    by simp
  then obtain s0 where s0: 1 − F (x-star F − 1 / s0) < 1 / t 0 < s0
    using eventually-happens'[OF trivial-limit-at-top-linorder] by blast
  define z where z = x-star F − 1 / s0
  have Fz: 1 − 1 / t < F z
    using s0 by (simp add: z-def)
  have p-bounds: 0 < 1 − 1 / t 1 − 1 / t < 1
    using t1 by simp-all
  have tail-quantile F t ≤ z
    using ginv-le-iff[OF p-bounds, of z] Fz by simp
  also have z < x-star F
    using s0(2) by (simp add: z-def)
  finally show ?thesis .
 qed
 have main: ⋀M. 0 < M ⟹ ∀F t in at-top. M ≤ 1 / (x-star F − tail-quantile
F t)
 proof −
  fix M :: real
  assume Mpos: 0 < M
  define z0 where z0 = x-star F − 1 / M
  have z0lt1: F z0 < 1
    using regularly-varying-pos[OF rv, of M] Mpos by (simp add: z0-def)
  show ∀F t in at-top. M ≤ 1 / (x-star F − tail-quantile F t)
    using eventually-gt-at-top[of 1 / (1 − F z0)] eventually-gt-at-top[of 1::real]
  proof eventually-elim
    case (elim t)
    hence t1: 1 < t and t2: 1 / (1 − F z0) < t
      by auto
    have p-bounds: 0 < 1 − 1 / t 1 − 1 / t < 1
      using t1 by simp-all
    have F z0 < 1 − 1 / t
      using t1 t2 z0lt1 by (simp add: field-simps)
    hence zlt: z0 < tail-quantile F t
      using ginv-le-iff[OF p-bounds, of z0] by linarith
    have lt: x-star F − tail-quantile F t < 1 / M
      using zlt by (simp add: z0-def)
    show ?case
      using Ult[OF t1] lt Mpos by (simp add: field-simps)
  qed
 qed
 show ?thesis
  unfolding filterlim-at-top
 proof
  fix M :: real
  show ∀F t in at-top. M ≤ 1 / (x-star F − tail-quantile F t)
    using main[of max M 1] eventually-mono
```

```
    by(cases M ≤ 1) fastforce+
  qed
qed

lemma tail-quantile-diff-scale-ratio:
  fixes U a :: real ⇒ real and γ :: real
  assumes apos: ⋀t. 0 < a t
    and gamma: γ ≠ 0
    and quant: ⋀x. 0 < x ⟹
      ((λt. (U (t * x) − U t) / a t) ⟶ (x powr γ − 1) / γ) at-top
    and y: 0 < y
  shows ((λt. a (t * y) / a t) ⟶ y powr γ) at-top
proof −
  define D where D = (λx::real. (x powr γ − 1) / γ)
  have quant': ⋀x. 0 < x ⟹ ((λt. (U (t * x) − U t) / a t) ⟶ D x) at-top
    using quant by (simp add: D-def)
  have D2-ne: D 2 ≠ 0
    using gamma by(simp add: D-def)
  have R2y: ((λt. (U (t * (y * 2)) − U t) / a t) ⟶ D (y * 2)) at-top
    using quant'[of y * 2] y by simp
  have Ry: ((λt. (U (t * y) − U t) / a t) ⟶ D y) at-top
    using quant'[of y] y by simp
  have Rty2: ((λt. (U (t * y * 2) − U (t * y)) / a (t * y)) ⟶ D 2) at-top
    using filterlim-compose[OF quant'[of 2]
      filterlim-tendsto-pos-mult-at-top[OF tendsto-const y filterlim-ident]]
    by (simp add: mult.commute)
  have eq1: (U (t * (y * 2)) − U t) / a t − (U (t * y) − U t) / a t
        = (U (t * y * 2) − U (t * y)) / a t for t
    by (simp add: mult.assoc diff-divide-distrib)
  have main: ((λt. (U (t * y * 2) − U (t * y)) / a t) ⟶ D (y * 2) − D y)
at-top
    using tendsto-diff[OF R2y Ry] unfolding eq1 .
  note ev-ne = tendsto-imp-eventually-ne[OF Rty2 D2-ne]
  have ratio-eq: ∀F t in at-top.
      ((U (t * y * 2) − U (t * y)) / a t) / ((U (t * y * 2) − U (t * y)) / a (t *
y))
      = a (t * y) / a t
    using apos by(auto intro!: eventually-mono[OF ev-ne])
  have ratio-tendsto: ((λt.
      ((U (t * y * 2) − U (t * y)) / a t) / ((U (t * y * 2) − U (t * y)) / a (t *
y)))
      ⟶ (D (y * 2) − D y) / D 2) at-top
    by (intro tendsto-divide main Rty2 D2-ne)
  have final: ((λt. a (t * y) / a t) ⟶ (D (y * 2) − D y) / D 2) at-top
    using Lim-transform-eventually[OF ratio-tendsto ratio-eq] .
  have value-eq: (D (y * 2) − D y) / D 2 = y powr γ
    using gamma y D2-ne by (simp add: D-def powr-mult field-simps)
  show ((λt. a (t * y) / a t) ⟶ y powr γ) at-top
    using final value-eq by simp
```

qed

lemma *tail-quantile-cont-ratio*:
fixes $F :: real \Rightarrow real$ **and** $\gamma :: real$ **and** $a :: real \Rightarrow real$
assumes *apos*: $\bigwedge t.\ 0 < a\ t$
and *gamma*: $\gamma \neq 0$
and *quant*: $\bigwedge x.\ 0 < x \Longrightarrow$
$((\lambda t.\ (tail\text{-}quantile\ F\ (t * x) - tail\text{-}quantile\ F\ t) / a\ t)$
$\longrightarrow (x\ powr\ \gamma - 1) / \gamma)\ at\text{-}top$
and Z: $1 < Z$
shows $((\lambda t.\ (tail\text{-}quantile\ F\ (t * Z * Z) - tail\text{-}quantile\ F\ (t * Z))$
$/ (tail\text{-}quantile\ F\ (t * Z) - tail\text{-}quantile\ F\ t)) \longrightarrow Z\ powr\ \gamma)\ at\text{-}top$
proof −
define D **where** $D = (\lambda x{::}real.\ (x\ powr\ \gamma - 1) / \gamma)$
have *Zpos*: $0 < Z$
using Z **by** *simp*
have *quant'*: $\bigwedge x.\ 0 < x \Longrightarrow$
$((\lambda t.\ (tail\text{-}quantile\ F\ (t * x) - tail\text{-}quantile\ F\ t) / a\ t) \longrightarrow D\ x)\ at\text{-}top$
using *quant* **by** (*simp add*: *D-def*)
from *tail-quantile-diff-scale-ratio*[*OF apos gamma quant*]
have *a-rv*: $\bigwedge y.\ 0 < y \Longrightarrow ((\lambda t.\ a\ (t * y) / a\ t) \longrightarrow y\ powr\ \gamma)\ at\text{-}top$ **.**
have *DZ-ne*: $D\ Z \neq 0$
using *gamma* Z **by**(*auto simp*: *D-def*)
have *f-lim*: $((\lambda t.\ (tail\text{-}quantile\ F\ (t * Z) - tail\text{-}quantile\ F\ t) / a\ t) \longrightarrow D\ Z)$
at-top
using *quant'*[*of Z*] Z **by** *simp*
have *g-lim*: $((\lambda t.\ a\ (t * Z) / a\ t) \longrightarrow Z\ powr\ \gamma)\ at\text{-}top$
using *a-rv*[*of Z*] Z **by** *simp*
have *h-lim*: $((\lambda t.\ (tail\text{-}quantile\ F\ (t * Z * Z)$
$- tail\text{-}quantile\ F\ (t * Z)) / a\ (t * Z)) \longrightarrow D\ Z)\ at\text{-}top$
using *filterlim-compose*[*OF f-lim filterlim-tendsto-pos-mult-at-top*
[*OF tendsto-const Zpos filterlim-ident*]]
by(*simp add*: *mult.commute*)
have *ev-ne*: $\forall_F\ t\ in\ at\text{-}top.\ (tail\text{-}quantile\ F\ (t * Z) - tail\text{-}quantile\ F\ t) / a\ t \neq 0$
by (*intro tendsto-imp-eventually-ne*[**where** *c*=*D Z*] *f-lim DZ-ne*)
have *ratio-eq*: $\forall_F\ t\ in\ at\text{-}top.\ (a\ (t * Z) / a\ t) *$
$(((tail\text{-}quantile\ F\ (t * Z * Z) - tail\text{-}quantile\ F\ (t * Z)) / a\ (t * Z))$
$/ ((tail\text{-}quantile\ F\ (t * Z) - tail\text{-}quantile\ F\ t) / a\ t))$
$= (tail\text{-}quantile\ F\ (t * Z * Z) - tail\text{-}quantile\ F\ (t * Z))$
$/ (tail\text{-}quantile\ F\ (t * Z) - tail\text{-}quantile\ F\ t)$
using *apos* **by**(*intro eventually-mono*[*OF ev-ne*])
(*smt* (*verit*) *Groups.mult-ac*(3) *divide-divide-eq-left field-class.field-divide-inverse*
nonzero-mult-divide-mult-cancel-left2
times-divide-eq-right zero-less-divide-iff)
have *ratio-tendsto*: $((\lambda t.$
$(a\ (t * Z) / a\ t) *$
$(((tail\text{-}quantile\ F\ (t * Z * Z) - tail\text{-}quantile\ F\ (t * Z)) / a\ (t * Z))$
$/ ((tail\text{-}quantile\ F\ (t * Z) - tail\text{-}quantile\ F\ t) / a\ t))) \longrightarrow Z\ powr\ \gamma)$
at-top

```
    using tendsto-mult[OF g-lim] tendsto-divide[OF h-lim f-lim DZ-ne] DZ-ne
    by fastforce
  show ?thesis
    using Lim-transform-eventually[OF ratio-tendsto ratio-eq] .
qed

lemma tail-quantile-diff-geometric-seq:
  fixes F :: real ⇒ real and γ Z t0 :: real and a :: real ⇒ real
  assumes apos: ⋀t. 0 < a t
    and gamma: γ ≠ 0
    and quant: ⋀x. 0 < x ⟹
        ((λt. (tail-quantile F (t * x) − tail-quantile F t) / a t)
          ⟶ (x powr γ − 1) / γ) at-top
    and Z: 1 < Z and t0: 0 < t0
  obtains N where
    ⋀k. tail-quantile F (t0 * Z ^ (N + k)) < tail-quantile F (t0 * Z ^ Suc (N +
k))
    and (λk. (tail-quantile F (t0 * Z ^ Suc (N + Suc k))
              − tail-quantile F (t0 * Z ^ (N + Suc k)))
            / (tail-quantile F (t0 * Z ^ Suc (N + k)) − tail-quantile F (t0 * Z ^
(N + k))))
          ⟶ Z powr γ
proof −
  define D where D = (λx::real. (x powr γ − 1) / γ)
  have DZ-pos: 0 < D Z
  proof (cases 0 < γ)
    case True
    show ?thesis
      using powr-less-mono2[OF True zero-le-one Z] True by (simp add: D-def)
  next
    case False
    hence neg: γ < 0
      using gamma by simp
    show ?thesis
     using powr-less-mono[OF neg, of Z] Z neg by (simp add: D-def divide-neg-neg)
  qed
  have seq-top: filterlim (λn. t0 * Z ^ n) at-top sequentially
    using filterlim-tendsto-pos-mult-at-top[OF tendsto-const[of t0] t0
       LIMSEQ-power-at-top[OF Z]] .
  have f-lim: ((λt. (tail-quantile F (t * Z) − tail-quantile F t) / a t) ⟶ D Z)
at-top
    using quant[of Z] Z by (simp add: D-def)
  define Δ where Δ = (λn. tail-quantile F (t0 * Z ^ Suc n) − tail-quantile F (t0
* Z ^ n))
  have Δ-ratio: (λn. Δ (Suc n) / Δ n) ⟶ Z powr γ
    using filterlim-compose[OF tail-quantile-cont-ratio[OF apos gamma quant Z]
seq-top]
    by (simp add: algebra-simps Δ-def)
  have Δ-over-a: (λn. Δ n / a (t0 * Z ^ n)) ⟶ D Z
```

```
    using filterlim-compose[OF f-lim seq-top] by (simp add: algebra-simps Δ-def)
  obtain N where N: ⋀n. N ≤ n ⟹ 0 < Δ n
    using order-tendstoD(1)[OF Δ-over-a DZ-pos] apos
    unfolding eventually-sequentially by (auto simp: pos-less-divide-eq)
  show thesis
  proof (rule that)
    show tail-quantile F (t0 * Z ^ (N + k)) < tail-quantile F (t0 * Z ^ Suc (N +
k)) for k
      using N[of N + k] by (simp add: Δ-def)
    show (λk. (tail-quantile F (t0 * Z ^ Suc (N + Suc k))
                - tail-quantile F (t0 * Z ^ (N + Suc k)))
              / (tail-quantile F (t0 * Z ^ Suc (N + k)) - tail-quantile F (t0 * Z ^
(N + k))))
          ⟶ Z powr γ
      using LIMSEQ-ignore-initial-segment[OF Δ-ratio, of N]
      by (simp add: Δ-def add.commute)
  qed
qed

lemma tail-quantile-neg-gap-power-ratio:
  fixes F :: real ⇒ real and γ :: real and a :: real ⇒ real
  assumes F: right-continuous-mono F 0 1
    and apos: ⋀t. 0 < a t
    and gamma: γ < 0
    and quant: ⋀x. 0 < x ⟹
     ((λt. (tail-quantile F (t * x) - tail-quantile F t) / a t)
       ⟶ (x powr γ - 1) / γ) at-top
    and Uconv: (tail-quantile F ⟶ x-star F) at-top
    and Z: 1 < Z and t0: 0 < t0
  shows (λn. (x-star F - tail-quantile F (t0 * Z ^ Suc n))
              / (x-star F - tail-quantile F (t0 * Z ^ n))) ⟶ Z powr γ
proof -
  have gamma-ne: γ ≠ 0
    using gamma by simp
  have Zg-bounds: 0 < Z powr γ Z powr γ < 1
    using powr-less-mono[OF gamma, of Z] Z by simp-all
  obtain N where
    N-pos: ⋀k. tail-quantile F (t0 * Z ^ (N + k)) < tail-quantile F (t0 * Z ^ Suc
(N + k))
    and N-ratio: (λk. (tail-quantile F (t0 * Z ^ Suc (N + Suc k))
                        - tail-quantile F (t0 * Z ^ (N + Suc k)))
                      / (tail-quantile F (t0 * Z ^ Suc (N + k))
                         - tail-quantile F (t0 * Z ^ (N + k)))) ⟶ Z powr γ
    using tail-quantile-diff-geometric-seq[OF apos gamma-ne quant Z t0] by blast
  define d where d = (λk. tail-quantile F (t0 * Z ^ Suc (N + k))
                          - tail-quantile F (t0 * Z ^ (N + k)))
  have d-pos: 0 < d k for k
    using N-pos by (simp add: d-def)
  have d-eq: d k = tail-quantile F (t0 * Z ^ Suc (N + k)) - tail-quantile F (t0 *
```

$Z \hat{} (N + k))$
for k
by (*simp add*: *d-def*)
have *d-ratio*: $(\lambda k.\ d\ (Suc\ k)\ /\ d\ k) \longrightarrow Z\ powr\ \gamma$
using *N-ratio* **by** (*simp add*: *d-def*)
note *d-summable* = *geometric-ratio-tailsum-summable*[*OF d-pos Zg-bounds d-ratio*]
have *d-sum-eq*: $(\sum k{<}n.\ d\ k)$
$= tail\text{-}quantile\ F\ (t0 * Z \hat{} (N + n)) - tail\text{-}quantile\ F\ (t0 * Z \hat{} N)$ **for** n
by (*induction n*) (*auto simp*: *d-eq*)
have *shift-at-top*: *filterlim* $(\lambda n{::}nat.\ N + n)$ *at-top sequentially*
by (*simp add*: *Groups.add-ac*(*2*) *filterlim-add-const-nat-at-top*)
have *Ns-top*: *filterlim* $(\lambda n.\ t0 * Z \hat{} (N + n))$ *at-top sequentially*
using *filterlim-compose*[*OF filterlim-tendsto-pos-mult-at-top*
[*OF tendsto-const*[*of t0*] *t0 LIMSEQ-power-at-top*[*OF Z*]] *shift-at-top*] **.**
have *U-Ns-tendsto*: $(\lambda n.\ tail\text{-}quantile\ F\ (t0 * Z \hat{} (N + n))) \longrightarrow x\text{-}star\ F$
using *filterlim-compose*[*OF Uconv Ns-top*] **.**
have *x-star-eq*: $x\text{-}star\ F = tail\text{-}quantile\ F\ (t0 * Z \hat{} N) + suminf\ d$
proof −
have *lim1*: $(\lambda n.\ tail\text{-}quantile\ F\ (t0 * Z \hat{} N) + (\sum k{<}n.\ d\ k)) \longrightarrow x\text{-}star\ F$
using *U-Ns-tendsto* **by** (*simp add*: *d-sum-eq*)
have *lim2*: $(\lambda n.\ tail\text{-}quantile\ F\ (t0 * Z \hat{} N) + (\sum k{<}n.\ d\ k))$
$\longrightarrow tail\text{-}quantile\ F\ (t0 * Z \hat{} N) + suminf\ d$
by (*intro tendsto-add*[*OF tendsto-const*] *summable-LIMSEQ*[*OF d-summable*])
show *?thesis*
using *LIMSEQ-unique*[*OF lim1 lim2*] **by** *simp*
qed
have *gap-eq*: $x\text{-}star\ F - tail\text{-}quantile\ F\ (t0 * Z \hat{} (N + n)) = (\sum k.\ d\ (k + n))$ **for** n
proof −
have $x\text{-}star\ F - tail\text{-}quantile\ F\ (t0 * Z \hat{} (N + n)) = suminf\ d - (\sum k{<}n.\ d\ k)$
using *x-star-eq d-sum-eq*[*of n*] **by** *simp*
also have $\ldots = (\sum k.\ d\ (k + n))$
using *suminf-minus-initial-segment*[*OF d-summable*, *of n*] **by** *simp*
finally show *?thesis* **.**
qed
note *tailsum-ratio* = *geometric-ratio-tailsum-ratio*[*OF d-pos Zg-bounds d-ratio*]
have *eq-fun*: $\bigwedge n.\ (x\text{-}star\ F - tail\text{-}quantile\ F\ (t0 * Z \hat{} (N + Suc\ n)))$
$/\ (x\text{-}star\ F - tail\text{-}quantile\ F\ (t0 * Z \hat{} (N + n)))$
$= (\sum k.\ d\ (k + Suc\ n))\ /\ (\sum k.\ d\ (k + n))$
proof −
fix n
have *e1*: $x\text{-}star\ F - tail\text{-}quantile\ F\ (t0 * Z \hat{} (N + Suc\ n)) = (\sum k.\ d\ (k + Suc\ n))$
using *gap-eq*[*of Suc n*] **.**
have *e2*: $x\text{-}star\ F - tail\text{-}quantile\ F\ (t0 * Z \hat{} (N + n)) = (\sum k.\ d\ (k + n))$
using *gap-eq*[*of n*] **.**
show $(x\text{-}star\ F - tail\text{-}quantile\ F\ (t0 * Z \hat{} (N + Suc\ n)))$

```
        / (x-star F − tail-quantile F (t0 * Z ^ (N + n)))
      = (∑ k. d (k + Suc n)) / (∑ k. d (k + n))
    unfolding e1 e2 ..
 qed
 have gap-ratio-shifted: (λn. (x-star F − tail-quantile F (t0 * Z ^ (N + Suc n)))
                        / (x-star F − tail-quantile F (t0 * Z ^ (N + n)))) ⟶
Z powr γ
   using tailsum-ratio unfolding eq-fun .
 have gap-ratio-shifted': (λn. (x-star F − tail-quantile F (t0 * Z ^ Suc (n + N)))
                        / (x-star F − tail-quantile F (t0 * Z ^ (n + N)))) ⟶
Z powr γ
   using gap-ratio-shifted by (simp add: add.commute)
 show ?thesis
   using gap-ratio-shifted' by (rule LIMSEQ-offset)
qed

lemma tail-quantile-neg-gap-ratio-of-power-ratio:
 fixes F :: real ⇒ real and γ :: real
 assumes F: right-continuous-mono F 0 1 and gamma: γ < 0
   and gap-pos: ⋀t. 1 < t ⟹ 0 < x-star F − tail-quantile F t
   and gap-power-ratio: ⋀Z t0. 1 < Z ⟹ 0 < t0 ⟹
      (λn. (x-star F − tail-quantile F (t0 * Z ^ Suc n))
           / (x-star F − tail-quantile F (t0 * Z ^ n))) ⟶ Z powr γ
 shows ⋀y. 0 < y ⟹
    ((λt. (x-star F − tail-quantile F (t * y)) / (x-star F − tail-quantile F t))
      ⟶ y powr γ) at-top
proof −
 define W where W = (λt. 1 / (x-star F − tail-quantile F t))
 have Wmono: W s ≤ W t if s: 1 < s and st: s ≤ t for s t
   using gap-pos[OF s] gap-pos[OF order.strict-trans2[OF s st]]
     tail-quantile-mono[OF F s st]
   by(auto simp: W-def intro!: divide-left-mono)
 have Wpos: 0 < W t if t: 1 < t for t
   using gap-pos[OF t] by (simp add: W-def)
 have Wpos-ev: ∀ F t in at-top. 0 < W t
   by(intro eventually-mono[OF eventually-gt-at-top[of 1::real]] Wpos)
 have Wratio: (λn. W (t0 * Z ^ Suc n) / W (t0 * Z ^ n)) ⟶ Z powr (− γ)
   if Z: 1 < Z and t0: 0 < t0 for Z t0 :: real
   using tendsto-inverse[OF gap-power-ratio[OF Z t0]] Z
   by(simp add: W-def powr-minus)
 fix y :: real
 assume y: 0 < y
 have main: ((λt. W (t * y) / W t) ⟶ y powr (− γ)) at-top
   using Wratio y gamma
   by(intro monotone-geometric-ratio-imp-ratio[OF Wmono Wpos-ev]) auto
 have eq: (λt. inverse (W (t * y) / W t))
        = (λt. (x-star F − tail-quantile F (t * y)) / (x-star F − tail-quantile
F t))
   by(simp add: W-def)
```

```
  show ((λt. (x-star F − tail-quantile F (t ∗ y)) / (x-star F − tail-quantile F t))
        ⟶ y powr γ) at-top
    using tendsto-inverse[OF main] y
    unfolding eq by (simp add: powr-minus)
qed

lemma tail-quantile-neg-regularly-varying:
  fixes F :: real ⇒ real and γ :: real and a :: real ⇒ real
  assumes F: right-continuous-mono F 0 1
    and apos: ⋀t. 0 < a t
    and gamma: γ < 0
    and quant: ⋀x. 0 < x ⟹
      ((λt. (tail-quantile F (t ∗ x) − tail-quantile F t) / a t) ⟶ (x powr γ −
1) / γ) at-top
  shows tail-quantile-neg-bdd: bdd-support F
    and tail-quantile-neg-tendsto: (tail-quantile F ⟶ x-star F) at-top
    and tail-quantile-neg-ratio: ⋀y. 0 < y ⟹
      ((λt. (x-star F − tail-quantile F (t ∗ y)) / (x-star F − tail-quantile F t))
        ⟶ y powr γ) at-top
proof −
  interpret right-continuous-mono F 0 1
    by fact
  have gamma-ne0: γ ≠ 0
    using gamma by simp
  have cont-ratio2: ((λt. (tail-quantile F (t ∗ 2 ∗ 2) − tail-quantile F (t ∗ 2))
                    / (tail-quantile F (t ∗ 2) − tail-quantile F t)) ⟶ 2 powr γ)
at-top
    using tail-quantile-cont-ratio[OF apos gamma-ne0 quant, of 2] by simp
  have Zn-at-top: filterlim (λn::nat. (2::real) ^ n) at-top sequentially
    using LIMSEQ-power-at-top[of 2] by simp
  have D2-pos: 0 < (2 powr γ − 1) / γ
    using powr-less-mono[OF gamma, of 2]
    by(auto intro!: divide-neg-neg gamma)
  define Δ where Δ = (λn. tail-quantile F (2 ^ Suc n) − tail-quantile F (2 ^ n))
  have Δ-ratio: (λn. Δ (Suc n) / Δ n) ⟶ 2 powr γ
  proof −
    have step: (λn. (tail-quantile F (2 ^ n ∗ 2 ∗ 2) − tail-quantile F (2 ^ n ∗ 2))
                    / (tail-quantile F (2 ^ n ∗ 2) − tail-quantile F (2 ^ n)))
              ⟶ 2 powr γ
      using filterlim-compose[OF cont-ratio2 Zn-at-top] .
    have eq: (tail-quantile F (2 ^ n ∗ 2 ∗ 2) − tail-quantile F (2 ^ n ∗ 2))
              / (tail-quantile F (2 ^ n ∗ 2) − tail-quantile F (2 ^ n))
            = Δ (Suc n) / Δ n for n
      by (simp add: Δ-def algebra-simps)
    show ?thesis
      using step unfolding eq .
  qed
  have Δ-over-a: (λn. Δ n / a (2 ^ n)) ⟶ (2 powr γ − 1) / γ
  proof −
```

```
  have ((λt. (tail-quantile F (t * 2) − tail-quantile F t) / a t)
        ⟶ (2 powr γ − 1) / γ) at-top
    using quant[of 2] by simp
  from filterlim-compose[OF this Zn-at-top]
  show ?thesis
    by (simp add: Δ-def mult.commute)
 qed
 have ev-Δ-pos: ∀F n in sequentially. 0 < Δ n
 proof −
  have ∀F n in sequentially. 0 < Δ n / a (2 ^ n)
    using order-tendstoD(1)[OF Δ-over-a D2-pos] by simp
  thus ?thesis
  proof eventually-elim
    case (elim n)
    thus ?case
      using apos[of 2 ^ n] by (simp add: zero-less-divide-iff)
  qed
 qed
 obtain N where N: ⋀n. N ≤ n ⟹ 0 < Δ n
  using ev-Δ-pos unfolding eventually-sequentially by blast
 define d where d = (λk. Δ (N + k))
 have d-pos: 0 < d k for k
  using N by (simp add: d-def)
 have d-ratio: (λk. d (Suc k) / d k) ⟶ 2 powr γ
  using LIMSEQ-ignore-initial-segment[OF Δ-ratio, of N]
  by(simp add: d-def add.commute)
 have L-bounds: 0 < 2 powr γ 2 powr γ < 1
  using powr-less-mono[OF gamma, of 2] by simp-all
 note d-summable = geometric-ratio-tailsum-summable[OF d-pos L-bounds d-ratio]
 have d-sum-eq: (∑k<n. d k) = tail-quantile F (2 ^ (N + n)) − tail-quantile F
(2 ^ N) for n
 proof (induction n)
  case (Suc n)
  have (∑k<Suc n. d k) = (∑k<n. d k) + d n
    by simp
  also have ... = (tail-quantile F (2 ^ (N + n)) − tail-quantile F (2 ^ N)) +
d n
    using Suc.IH by simp
  also have ... = tail-quantile F (2 ^ (N + Suc n)) − tail-quantile F (2 ^ N)
    by (simp add: d-def Δ-def)
  finally show ?case .
 qed simp
 have partial-le: (∑k<n. d k) ≤ suminf d for n
  by(intro sum-le-suminf[OF d-summable finite-lessThan] less-imp-le[OF d-pos])
 define B where B = tail-quantile F (2 ^ N) + suminf d
 have U-2pow-le: tail-quantile F (2 ^ (N + n)) ≤ B for n
  using partial-le[of n] unfolding d-sum-eq[of n] B-def by simp
 have U-le-B: tail-quantile F t ≤ B if t1: 1 < t for t
 proof −
```

```
    obtain n where n: t ≤ 2 ^ (N + n)
    proof −
      obtain N0 where N0: ∀ k≥N0. t ≤ 2 ^ k
        using Zn-at-top unfolding filterlim-at-top eventually-sequentially by blast
      have N0 ≤ N + N0
        by simp
      hence t ≤ 2 ^ (N + N0)
        using N0 by simp
      thus thesis
        using that by blast
    qed
    have tail-quantile F t ≤ tail-quantile F (2 ^ (N + n))
      using t1 n by (intro tail-quantile-mono[OF F]) auto
    also have ... ≤ B
      by (rule U-2pow-le)
    finally show ?thesis .
  qed
  show bdd: bdd-support F
  proof (rule ccontr)
    assume nbdd: ¬ bdd-support F
    note unbdd = F-lt-1-unbdd-support[OF F nbdd]
    note U-at-top = tail-quantile-at-top[OF F unbdd]
    obtain t where t: 1 < t B < tail-quantile F t
    proof −
      have ∀ F t in at-top. B + 1 ≤ tail-quantile F t ∧ 1 < t
        using eventually-conj[OF U-at-top[unfolded filterlim-at-top, rule-format, of
B + 1]
              eventually-gt-at-top[of 1::real]] .
      then obtain t0 where t0: B + 1 ≤ tail-quantile F t0 1 < t0
        using eventually-happens'[OF trivial-limit-at-top-linorder] by blast
      show thesis
        using that[of t0] t0 by simp
    qed
    show False
      using U-le-B[OF t(1)] t(2) by simp
  qed
  note Fxstar = F-x-star-eq-1[OF F bdd]
  have U-le-xstar: tail-quantile F t ≤ x-star F if t1: 1 < t for t
    using t1 Fxstar by (simp add: field-simps ginv-le-iff)
  show tendsto-goal: (tail-quantile F ⟶ x-star F) at-top
    by (rule tail-quantile-tendsto-x-star[OF F bdd])
  have gap-pos: 0 < x-star F − tail-quantile F t if t1: 1 < t for t
  proof (rule ccontr)
    assume ¬ 0 < x-star F − tail-quantile F t
    hence eq: tail-quantile F t = x-star F
      using U-le-xstar[OF t1] by simp
    have const: tail-quantile F s = x-star F if s1: t ≤ s for s
    proof (rule antisym)
      show tail-quantile F s ≤ x-star F
```

```
      using U-le-xstar[of s] t1 s1 by simp
    have tail-quantile F t ≤ tail-quantile F s
      using tail-quantile-mono[OF F t1 s1] .
    thus x-star F ≤ tail-quantile F s
      using eq by simp
  qed
  have ev0: ∀F s in at-top. (tail-quantile F (s * 2) − tail-quantile F s) / a s = 0
    using const t1 by (auto intro!: eventually-mono[OF eventually-ge-at-top[of
t]])
  have ev0': ∀F s in at-top. 0 = (tail-quantile F (s * 2) − tail-quantile F s) / a
s
    using ev0 by eventually-elim simp
  have ratio-to-0: ((λs. (tail-quantile F (s * 2) − tail-quantile F s) / a s) ⟶
0) at-top
    using Lim-transform-eventually[OF tendsto-const ev0'] .
  have ratio-to-D2: ((λs. (tail-quantile F (s * 2) − tail-quantile F s) / a s)
                    ⟶ (2 powr γ − 1) / γ) at-top
    using quant[of 2] by simp
  have (2 powr γ − 1) / γ = 0
    using tendsto-unique[OF trivial-limit-at-top-linorder ratio-to-D2 ratio-to-0] .
  thus False
    using D2-pos by simp
qed
have gap-power-ratio: ⋀Z t0. 1 < Z ⟹ 0 < t0 ⟹
    (λn. (x-star F − tail-quantile F (t0 * Z ^ Suc n))
        / (x-star F − tail-quantile F (t0 * Z ^ n)))
      ⟶ Z powr γ
  using tail-quantile-neg-gap-power-ratio[OF F apos gamma quant tendsto-goal] .
show ⋀y. 0 < y ⟹
    ((λt. (x-star F − tail-quantile F (t * y)) / (x-star F − tail-quantile F t))
      ⟶ y powr γ) at-top
 using tail-quantile-neg-gap-ratio-of-power-ratio[OF F gamma gap-pos gap-power-ratio]
.
qed

lemma tail-quantile-gap-ratio-neg-imp-tail-regularly-varying:
  fixes F :: real ⇒ real and γ :: real
  assumes F: right-continuous-mono F 0 1 and gamma: γ < 0
    and bdd: bdd-support F
    and gap-rv: ⋀y. 0 < y ⟹
      ((λt. (x-star F − tail-quantile F (t * y)) / (x-star F − tail-quantile F t))
        ⟶ y powr γ) at-top
  shows regularly-varying (λs. 1 − F (x-star F − 1 / s)) (1 / γ)
proof −
  interpret right-continuous-mono F 0 1
    by fact
  have phi-pos: 0 < 1 − F (x-star F − 1 / s) if s: 0 < s for s
    using s lt-x-star-iff-F-lt-1[OF F bdd, THEN iffD1] by fastforce
  note Fxstar = F-x-star-eq-1[OF F bdd]
```

```
  have U-le-xstar: tail-quantile F t ≤ x-star F if t1: 1 < t for t
    using t1 Fxstar by (simp add: field-simps ginv-le-iff)
  note Utend = tail-quantile-tendsto-x-star[OF F bdd]
  have gap-pos: 0 < x-star F − tail-quantile F t if t1: 1 < t for t
  proof (rule ccontr)
    assume ¬ 0 < x-star F − tail-quantile F t
    hence eq: tail-quantile F t = x-star F
      using U-le-xstar[OF t1] by simp
    have const: tail-quantile F u = x-star F if u1: t ≤ u for u
    proof (rule antisym)
      show tail-quantile F u ≤ x-star F
        using U-le-xstar[of u] t1 u1 by simp
      have tail-quantile F t ≤ tail-quantile F u
        using tail-quantile-mono[OF F t1 u1] .
      thus x-star F ≤ tail-quantile F u
        using eq by simp
    qed
    have ev0: ∀F u in at-top.
              (x-star F − tail-quantile F (u * 2)) / (x-star F − tail-quantile F u)
= 0
      using const t1 by (auto intro!: eventually-mono[OF eventually-ge-at-top[of
t]])
    have ev0': ∀F u in at-top. (0::real)
          = (x-star F − tail-quantile F (u * 2)) / (x-star F − tail-quantile F u)
      using ev0 by eventually-elim simp
    have ratio-to-0: ((λu.
       (x-star F − tail-quantile F (u * 2)) / (x-star F − tail-quantile F u)) ⟶
0) at-top
      using Lim-transform-eventually[OF tendsto-const ev0'] .
    have ratio-to-2g: ((λu.
       (x-star F − tail-quantile F (u * 2)) / (x-star F − tail-quantile F u))
        ⟶ 2 powr γ) at-top
      using gap-rv[of 2] by simp
    have 2 powr γ = 0
      using tendsto-unique[OF trivial-limit-at-top-linorder ratio-to-2g ratio-to-0] .
    thus False
      by simp
  qed
  define W where W = (λt::real. 1 / (x-star F − tail-quantile F t))
  define s where s = (λw::real. 1 / (1 − F (x-star F − 1 / w)))
  have galois: W t ≤ w ⟷ t ≤ s w if t1: 1 < t for t w
  proof (cases 0 < w)
    case True
    note gpos = gap-pos[OF t1]
    note phipos = phi-pos[OF True]
    have p-bounds: 0 < 1 − 1 / t 1 − 1 / t < 1
      using t1 by simp-all
    have W t ≤ w ⟷ tail-quantile F t ≤ x-star F − 1 / w
      unfolding W-def using gpos True by (simp add: field-simps)
```

```
    also have ... ⟷ 1 − 1 / t ≤ F (x-star F − 1 / w)
      using ginv-le-iff[OF p-bounds, of x-star F − 1 / w] by simp
    also have ... ⟷ t ≤ s w
      using phipos True t1 by (simp add: s-def field-simps)
    finally show ?thesis .
  next
    case False
    have ¬ (W t ≤ w)
    proof −
      note gpos = gap-pos[OF t1]
      hence 0 < W t
        unfolding W-def by simp
      thus ?thesis
        using False by linarith
    qed
    moreover have ¬ (t ≤ s w)
    proof −
      have wle0: w < 0 ∨ w = 0
        using False by linarith
      have 1 / w ≤ 0
        using wle0 by (auto simp: divide-le-0-iff)
      hence x-star F ≤ x-star F − 1 / w
        by simp
      hence F (x-star F) ≤ F (x-star F − 1 / w)
        by (rule monoD[OF mono])
      hence F (x-star F − 1 / w) = 1
        using Fxstar right-continuous-mono.f-le-b[OF F, of x-star F − 1 / w] by
simp
      hence s w = 0
        unfolding s-def by simp
      thus ?thesis
        using t1 by simp
    qed
    ultimately show ?thesis
      by simp
  qed
  have Wtop: filterlim W at-top at-top
  proof −
    have gap-tend0: ((λt. x-star F − tail-quantile F t) ⟶ 0) at-top
      using tendsto-diff[OF tendsto-const[of x-star F] Utend] by simp
    have gap-pos-ev: ∀F t in at-top. 0 < x-star F − tail-quantile F t
      using eventually-gt-at-top[of 1::real] by eventually-elim (rule gap-pos)
    show ?thesis
      unfolding W-def
        using filterlim-inverse-at-top[OF gap-tend0 gap-pos-ev] by (simp add: in-
verse-eq-divide)
  qed
  have s-top: filterlim s at-top at-top
    unfolding filterlim-at-top
```

```
  proof
    fix M :: real
    define M' where M' = max M 2
    have M'1: 1 < M'
      unfolding M'-def by simp
    have WM'pos: 0 < W M'
      unfolding W-def using gap-pos[OF M'1] by simp
    show ∀_F w in at-top. M ≤ s w
      using eventually-ge-at-top[of W M']
      by eventually-elim (use galois[OF M'1] in simp add: M'-def)
  qed
  have Wrv: ((λt. W (t * y) / W t) ⟶ y powr (- γ)) at-top if y: 0 < y for y
  proof -
    have r1: ((λt. (x-star F - tail-quantile F (t * y)) / (x-star F - tail-quantile
F t))
              ⟶ y powr γ) at-top
      using gap-rv[OF y] .
    have ypow-ne: y powr γ ≠ 0
      using y by simp
    note r1inv = tendsto-inverse[OF r1 ypow-ne]
    have ty-top: filterlim (λt::real. t * y) at-top at-top
     using filterlim-tendsto-pos-mult-at-top[OF tendsto-const[of y] y filterlim-ident]
      by(simp add: mult.commute)
    have ev-ty: ∀_F t in at-top. 1 < t * y
      using ty-top[unfolded filterlim-at-top, rule-format, of 2] by eventually-elim
simp
    have eq: ∀_F t in at-top. inverse ((x-star F - tail-quantile F (t * y))
                    / (x-star F - tail-quantile F t)) = W (t * y) / W t
      using eventually-gt-at-top[of 1] ev-ty
      by eventually-elim (use gap-pos in simp add: W-def field-simps)
    have ((λt. W (t * y) / W t) ⟶ inverse (y powr γ)) at-top
      using Lim-transform-eventually[OF r1inv eq] .
    thus ?thesis
      by (simp add: powr-minus)
  qed
  have s-pos: 0 < s w if w: 0 < w for w
    unfolding s-def using phi-pos[OF w] by simp
  have s-ratio: ((λu. s (u * x) / s u) ⟶ x powr (- 1 / γ)) at-top
    if x: 0 < x for x
    using galois-inverse-ratio-tendsto[OF Wtop s-top s-pos galois - Wrv x] gamma
    by simp
  have phi-ratio: ((λu. (1 - F (x-star F - 1 / (u * x))) / (1 - F (x-star F - 1
/ u)))
                   ⟶ x powr (1 / γ)) at-top if x: 0 < x for x
  proof -
    have same: (λu. s (u * x) / s u)
               = (λu. (1 - F (x-star F - 1 / u)) / (1 - F (x-star F - 1 / (u *
x))))
      using phi-pos x by (auto simp: s-def)
```

```
    have main: ((λu. (1 − F (x-star F − 1 / u)) / (1 − F (x-star F − 1 / (u ∗
x))))
                  ⟶ x powr (− 1 / γ)) at-top
      using s-ratio[OF x] unfolding same .
    have xr-ne: x powr (− 1 / γ) ≠ 0
      using x by simp
    have inv: ((λu. inverse ((1 − F (x-star F − 1 / u)) / (1 − F (x-star F − 1
/ (u ∗ x)))))
                  ⟶ inverse (x powr (− 1 / γ))) at-top
      by (intro tendsto-inverse main xr-ne)
    have eq2: (λu. inverse ((1 − F (x-star F − 1 / u)) / (1 − F (x-star F − 1
/ (u ∗ x)))))
                = (λu. (1 − F (x-star F − 1 / (u ∗ x))) / (1 − F (x-star F − 1 /
u)))
      by simp
    have lim: ((λu. (1 − F (x-star F − 1 / (u ∗ x))) / (1 − F (x-star F − 1 /
u)))
                  ⟶ inverse (x powr (− 1 / γ))) at-top
      using inv unfolding eq2 .
    have val-eq: inverse (x powr (− 1 / γ)) = x powr (1 / γ)
      using x by (simp add: powr-minus)
    show ?thesis
      using lim unfolding val-eq .
  qed
  show ?thesis
  proof (rule regularly-varyingI)
    have Fmeas: F ∈ borel-measurable borel
      using borel-measurable-mono[OF mono] .
    have cont: continuous-on {0<..} (λs::real. x-star F − 1 / s)
      by (intro continuous-intros) simp
   have meas1: (λs::real. x-star F − 1 / s) ∈ borel-measurable (restrict-space borel
{0<..})
      using cont by (rule borel-measurable-continuous-on-restrict)
    show (λs. 1 − F (x-star F − 1 / s)) ∈ borel-measurable (restrict-space borel
{0<..})
      using measurable-compose[OF meas1 Fmeas] by simp
  qed(use phi-pos phi-ratio in auto)
qed

lemma tail-quantile-pos-power-ratio:
  fixes F :: real ⇒ real and γ :: real and a :: real ⇒ real
  assumes F: right-continuous-mono F 0 1
    and apos: ⋀t. 0 < a t
    and gamma: 0 < γ
    and quant: ⋀x. 0 < x ⟹
        ((λt. (tail-quantile F (t ∗ x) − tail-quantile F t) / a t)
          ⟶ (x powr γ − 1) / γ) at-top
    and Uinf: filterlim (tail-quantile F) at-top at-top
    and Z: 1 < Z and t0: 0 < t0
```

shows (λn. *tail-quantile F* (*t0* $*$ *Z* $\hat{}$ *Suc n*) / *tail-quantile F* (*t0* $*$ *Z* $\hat{}$ *n*)) $\longrightarrow$ *Z powr* γ
proof −
have *gamma-ne*: $\gamma \neq 0$
using *gamma* **by** *simp*
have *Zg-gt1*: *1* $<$ *Z powr* γ
using *powr-less-mono2*[*OF gamma zero-le-one Z*] **by** *simp*
have *s-top*: *filterlim* (λn. *t0* $*$ *Z* $\hat{}$ *n*) *at-top sequentially*
using *filterlim-tendsto-pos-mult-at-top*[*OF tendsto-const*[*of t0*] *t0* *LIMSEQ-power-at-top*[*OF Z*]] **.**
obtain *N* **where**
N-pos: $\bigwedge k$. *tail-quantile F* (*t0* $*$ *Z* $\hat{}$ (*N* + *k*)) $<$ *tail-quantile F* (*t0* $*$ *Z* $\hat{}$ *Suc* (*N* + *k*))
and *N-ratio*: (λk. (*tail-quantile F* (*t0* $*$ *Z* $\hat{}$ *Suc* (*N* + *Suc k*)) − *tail-quantile F* (*t0* $*$ *Z* $\hat{}$ (*N* + *Suc k*))) / (*tail-quantile F* (*t0* $*$ *Z* $\hat{}$ *Suc* (*N* + *k*)) − *tail-quantile F* (*t0* $*$ *Z* $\hat{}$ (*N* + *k*)))) $\longrightarrow$ *Z powr* γ
using *tail-quantile-diff-geometric-seq*[*OF apos gamma-ne quant Z t0*] **by** *blast*
define *d* **where** *d* = (λk. *tail-quantile F* (*t0* $*$ *Z* $\hat{}$ *Suc* (*N* + *k*)) − *tail-quantile F* (*t0* $*$ *Z* $\hat{}$ (*N* + *k*)))
have *d-pos*: *0* $<$ *d k* **for** *k*
using *N-pos* **by** (*simp add*: *d-def*)
have *d-eq*: *d k* = *tail-quantile F* (*t0* $*$ *Z* $\hat{}$ *Suc* (*N* + *k*)) − *tail-quantile F* (*t0* $*$ *Z* $\hat{}$ (*N* + *k*))
for *k*
by (*simp add*: *d-def*)
have *d-ratio*: (λk. *d* (*Suc k*) / *d k*) $\longrightarrow$ *Z powr* γ
using *N-ratio* **by** (*simp add*: *d-def*)
note *d-sum-ratio* = *geometric-ratio-sum-ratio*[*OF d-pos Zg-gt1 d-ratio*]
have *d-sum-eq*: ($\sum j \leq k$. *d j*) = *tail-quantile F* (*t0* $*$ *Z* $\hat{}$ (*N* + *Suc k*)) − *tail-quantile F* (*t0* $*$ *Z* $\hat{}$ *N*) **for** *k*
by (*induction k*) (*auto simp*: *d-eq*)
define *C* **where** *C* = *tail-quantile F* (*t0* $*$ *Z* $\hat{}$ *N*)
have *PQ-ratio*: (λk. (*tail-quantile F* (*t0* $*$ *Z* $\hat{}$ (*N* + *Suc* (*Suc k*))) − *C*) / (*tail-quantile F* (*t0* $*$ *Z* $\hat{}$ (*N* + *Suc k*)) − *C*)) $\longrightarrow$ *Z powr* γ
proof −
have (λk. (*tail-quantile F* (*t0* $*$ *Z* $\hat{}$ (*N* + *Suc k*)) − *tail-quantile F* (*t0* $*$ *Z* $\hat{}$ *N*) + *d* (*Suc k*)) / (*tail-quantile F* (*t0* $*$ *Z* $\hat{}$ (*N* + *Suc k*)) − *tail-quantile F* (*t0* $*$ *Z* $\hat{}$ *N*))) $\longrightarrow$ *Z powr* γ
using *d-sum-ratio* **by** (*simp add*: *d-sum-eq*)
thus *?thesis*
by (*simp add*: *d-def C-def algebra-simps*)
qed
have *shift1-at-top*: *filterlim* (λk::*nat*. *N* + *Suc k*) *at-top sequentially*

by (*intro filterlim-subseq strict-monoI*) *simp*
note *P-seq-at-top* = *filterlim-compose*[*OF s-top shift1-at-top*]
note *P-at-top* = *filterlim-compose*[*OF Uinf P-seq-at-top*]
note *invP0* = *tendsto-inverse-0-at-top*[*OF P-at-top*]
have *C-over-P*: (λ*k*. *C* / *tail-quantile F* (*t0* ∗ *Z* ^ (*N* + *Suc k*))) ⟶ *0*
using *tendsto-mult*[*OF tendsto-const*[*of C*] *invP0*] **by** (*simp add*: *divide-inverse*)
have *one-minus-C-over-P*: (λ*k*. *1* − *C* / *tail-quantile F* (*t0* ∗ *Z* ^ (*N* + *Suc k*))) ⟶ *1*
using *tendsto-diff*[*OF tendsto-const*[*of 1*] *C-over-P*] **by** *simp*
have *combo*: (λ*k*. (*tail-quantile F* (*t0* ∗ *Z* ^ (*N* + *Suc* (*Suc k*))) − *C*)
/ (*tail-quantile F* (*t0* ∗ *Z* ^ (*N* + *Suc k*)) − *C*)
∗ (*1* − *C* / *tail-quantile F* (*t0* ∗ *Z* ^ (*N* + *Suc k*)))
+ *C* / *tail-quantile F* (*t0* ∗ *Z* ^ (*N* + *Suc k*))) ⟶ *Z powr* γ
using *tendsto-add*[*OF tendsto-mult*[*OF PQ-ratio one-minus-C-over-P*] *C-over-P*] **by** *simp*
have *P-ge*: ∀ *F k in sequentially*. |*C*| + *1* ≤ *tail-quantile F* (*t0* ∗ *Z* ^ (*N* + *Suc k*))
using *P-at-top*[*unfolded filterlim-at-top*, *rule-format*, *of* |*C*| + *1*] **by** *simp*
have *combo-eq*: ∀ *F k in sequentially*. (*tail-quantile F* (*t0* ∗ *Z* ^ (*N* + *Suc* (*Suc k*))) − *C*)
/ (*tail-quantile F* (*t0* ∗ *Z* ^ (*N* + *Suc k*)) − *C*)
∗ (*1* − *C* / *tail-quantile F* (*t0* ∗ *Z* ^ (*N* + *Suc k*)))
+ *C* / *tail-quantile F* (*t0* ∗ *Z* ^ (*N* + *Suc k*))
= *tail-quantile F* (*t0* ∗ *Z* ^ (*N* + *Suc* (*Suc k*)))
/ *tail-quantile F* (*t0* ∗ *Z* ^ (*N* + *Suc k*))
using *P-ge*
proof *eventually-elim*
case (*elim k*)
have *C-lt*: *C* < *tail-quantile F* (*t0* ∗ *Z* ^ (*N* + *Suc k*))
using *elim abs-ge-self*[*of C*] **by** *linarith*
have *P-pos*: *0* < *tail-quantile F* (*t0* ∗ *Z* ^ (*N* + *Suc k*))
using *elim abs-ge-zero*[*of C*] **by** *linarith*
have *p-ne*: *tail-quantile F* (*t0* ∗ *Z* ^ (*N* + *Suc k*)) ≠ *0*
using *P-pos* **by** *simp*
have *pc-ne*: *tail-quantile F* (*t0* ∗ *Z* ^ (*N* + *Suc k*)) − *C* ≠ *0*
using *C-lt* **by** *simp*
show *?case*
using *p-ne pc-ne* **by** (*simp add*: *field-simps*)
qed
have *shifted-target*: (λ*k*. *tail-quantile F* (*t0* ∗ *Z* ^ (*N* + *Suc* (*Suc k*)))
/ *tail-quantile F* (*t0* ∗ *Z* ^ (*N* + *Suc k*))) ⟶ *Z powr* γ
using *Lim-transform-eventually*[*OF combo combo-eq*] .
have *offset-eq*: *tail-quantile F* (*t0* ∗ *Z* ^ *Suc* (*k* + (*N* + *1*)))
/ *tail-quantile F* (*t0* ∗ *Z* ^ (*k* + (*N* + *1*)))
= *tail-quantile F* (*t0* ∗ *Z* ^ (*N* + *Suc* (*Suc k*)))
/ *tail-quantile F* (*t0* ∗ *Z* ^ (*N* + *Suc k*)) **for** *k*
by (*simp add*: *add.commute*)
have *offset-target*: (λ*k*. *tail-quantile F* (*t0* ∗ *Z* ^ *Suc* (*k* + (*N* + *1*)))
/ *tail-quantile F* (*t0* ∗ *Z* ^ (*k* + (*N* + *1*)))) ⟶ *Z powr*

γ
  unfolding *offset-eq* **using** *shifted-target* **.**
 show *?thesis*
  using *offset-target* **by** (*rule LIMSEQ-offset*)
qed

lemma *tail-quantile-pos-ratio-of-power-ratio*:
 fixes F :: *real* ⇒ *real* **and** γ :: *real*
 assumes *F*: *right-continuous-mono F 0 1* **and** *gamma*: $0 < \gamma$
  and *Uinf*: *filterlim* (*tail-quantile F*) *at-top at-top*
  and *power-ratio*: $\bigwedge Z\ t0.\ 1 < Z \Longrightarrow 0 < t0 \Longrightarrow$
    (λ*n*. *tail-quantile F* (*t0* ∗ *Z* ^ *Suc n*) / *tail-quantile F* (*t0* ∗ *Z* ^ *n*)) ⟶
Z powr γ
 shows $\bigwedge y.\ 0 < y \Longrightarrow$
    ((λ*t*. *tail-quantile F* (*t* ∗ *y*) / *tail-quantile F t*) ⟶ *y powr* γ) *at-top*
proof −
 have *Upos*: ∀ *F t in at-top*. *0* < *tail-quantile F t*
  using *Uinf* **unfolding** *filterlim-at-top-dense* **by** *blast*
 fix *y* :: *real*
 assume *y*: *0* < *y*
 show ((λ*t*. *tail-quantile F* (*t* ∗ *y*) / *tail-quantile F t*) ⟶ *y powr* γ) *at-top*
  using *power-ratio y*
  by(*intro monotone-geometric-ratio-imp-ratio*[*OF - Upos gamma*])
   (*auto intro*: *tail-quantile-mono*[*OF F*])
qed

lemma *tail-quantile-pos-regularly-varying*:
 fixes *F* :: *real* ⇒ *real* **and** γ :: *real* **and** *a* :: *real* ⇒ *real*
 assumes *F*: *right-continuous-mono F 0 1*
  and *apos*: $\bigwedge t.\ 0 < a\ t$
  and *gamma*: $0 < \gamma$
  and *quant*: $\bigwedge x.\ 0 < x \Longrightarrow$
    ((λ*t*. (*tail-quantile F* (*t* ∗ *x*) − *tail-quantile F t*) / *a t*)
     ⟶ (*x powr* γ − *1*) / γ) *at-top*
 shows *tail-quantile-pos-at-top*: *filterlim* (*tail-quantile F*) *at-top at-top*
  and *tail-quantile-pos-ratio*: $\bigwedge y.\ 0 < y \Longrightarrow$
    ((λ*t*. *tail-quantile F* (*t* ∗ *y*) / *tail-quantile F t*) ⟶ *y powr* γ) *at-top*
proof −
 interpret *right-continuous-mono F 0 1*
  by *fact*
 have *gamma-ne0*: $\gamma \neq 0$
  using *gamma* **by** *simp*
 have *a-rv*: ((λ*t*. *a* (*t* ∗ *y*) / *a t*) ⟶ *y powr* γ) *at-top* **if** *y*: *0* < *y* **for** *y*
  by (*rule tail-quantile-diff-scale-ratio*[**where** *U* = *tail-quantile F*,
      OF apos gamma-ne0 quant y])

 show *U-at-top*: *filterlim* (*tail-quantile F*) *at-top at-top*
 proof −
  have *a-rv2*: ((λ*t*. *a* (*t* ∗ *2*) / *a t*) ⟶ *2 powr* γ) *at-top*

using *a-rv*[*of 2*] **by** *simp*
have *quant2*: ((λ*t*. (*tail-quantile F* (*t* ∗ *2*) − *tail-quantile F t*) / *a t*)
⟶ (*2 powr* γ − *1*) / γ) *at-top*
using *quant*[*of 2*] **by** *simp*
define *r0* :: *real* **where** *r0* = (*1* + *2 powr* γ) / *2*
have *r0-bounds*: *1* < *r0 r0* < *2 powr* γ
using *gamma* **by** (*simp-all add*: *r0-def*)
define *c* :: *real* **where** *c* = ((*2 powr* γ − *1*) / γ) / *2*
have *c-pos*: *0* < *c*
using *gamma* **by** (*simp add*: *c-def*)
have *c-lt-D2*: *c* < (*2 powr* γ − *1*) / γ
using *gamma* **by**(*auto simp*: *c-def intro*!: *divide-strict-left-mono*)
have *a-ev*: ∀ *F t in at-top*. *r0* ∗ *a t* < *a* (*t* ∗ *2*)
using *order-tendstoD*(*1*)[*OF a-rv2 r0-bounds*(*2*)]
by *eventually-elim* (*use apos* **in** *simp add*: *field-simps*)
have *U-ev*: ∀ *F t in at-top*. *c* ∗ *a t* < *tail-quantile F* (*t* ∗ *2*) − *tail-quantile F t*
using *order-tendstoD*(*1*)[*OF quant2 c-lt-D2*]
by *eventually-elim* (*use apos* **in** *simp add*: *field-simps*)
obtain *T1* **where** *T1*: ⋀*t*. *T1* ≤ *t* ⟹ *r0* ∗ *a t* < *a* (*t* ∗ *2*)
using *a-ev* **unfolding** *eventually-at-top-linorder* **by** *blast*
obtain *T2* **where** *T2*: ⋀*t*. *T2* ≤ *t* ⟹ *c* ∗ *a t* < *tail-quantile F* (*t* ∗ *2*) − *tail-quantile F t*
using *U-ev* **unfolding** *eventually-at-top-linorder* **by** *blast*
define *T* **where** *T* = *max T1 T2*
have *T*: ⋀*t*. *T* ≤ *t* ⟹ *r0* ∗ *a t* < *a* (*t* ∗ *2*)
⋀*t*. *T* ≤ *t* ⟹ *c* ∗ *a t* < *tail-quantile F* (*t* ∗ *2*) − *tail-quantile F t*
using *T1 T2* **by** (*auto simp*: *T-def*)
define *t0* :: *real* **where** *t0* = *max T 2*
have *t0-ge*: *T* ≤ *t0 2* ≤ *t0*
by (*simp-all add*: *t0-def*)
define *tn* :: *nat* ⇒ *real* **where** *tn n* = *t0* ∗ *2* ^ *n* **for** *n*
have *two-pow-ge1*: (*1*::*real*) ≤ *2* ^ *n* **for** *n*
by *simp*
have *t0-le-tn*: *t0* ≤ *tn n* **for** *n*
using *two-pow-ge1*[*of n*] *t0-ge*(*2*) **by** (*simp add*: *tn-def mult-left-mono*)
have *tn-ge-T*: *T* ≤ *tn n* **for** *n*
using *t0-ge*(*1*) *t0-le-tn*[*of n*] **by** *simp*
have *tn-gt1*: *1* < *tn n* **for** *n*
using *t0-ge*(*2*) *t0-le-tn*[*of n*] **by** *simp*
have *tn-Suc*: *tn* (*Suc n*) = *tn n* ∗ *2* **for** *n*
by (*simp add*: *tn-def*)
have *tn-mono*: *m* ≤ *n* ⟹ *tn m* ≤ *tn n* **for** *m n*
using *t0-ge*(*2*) **by** (*simp add*: *tn-def mult-left-mono power-increasing*)
have *a-grow*: *r0* ^ *n* ∗ *a t0* ≤ *a* (*tn n*) **for** *n*
proof (*induction n*)
case *0*
show *?case*
by (*simp add*: *tn-def*)
next

```
    case (Suc n)
    have r0 * a (tn n) < a (tn n * 2)
      using T(1)[OF tn-ge-T[of n]] by simp
    hence step: r0 * a (tn n) < a (tn (Suc n))
      by (simp add: tn-Suc)
    have r0 ^ Suc n * a t0 = r0 * (r0 ^ n * a t0)
      by simp
    also have ... ≤ r0 * a (tn n)
      using Suc.IH r0-bounds(1) by (intro mult-left-mono) auto
    also have ... < a (tn (Suc n))
      by (rule step)
    finally show ?case
      by simp
  qed
  note r0n-at-top = LIMSEQ-power-at-top[OF r0-bounds(1)]
  have a-tn-at-top: filterlim (λn. a (tn n)) at-top sequentially
  proof (rule filterlim-at-top-mono)
   from filterlim-tendsto-pos-mult-at-top[OF tendsto-const apos[of t0] r0n-at-top]
    have filterlim (λn. a t0 * r0 ^ n) at-top sequentially .
    thus filterlim (λn. r0 ^ n * a t0) at-top sequentially
      by (simp add: mult.commute)
    show ∀F n in sequentially. r0 ^ n * a t0 ≤ a (tn n)
      using a-grow by (intro always-eventually) simp
  qed
  have U-mono-seq: m ≤ n ⟹ tail-quantile F (tn m) ≤ tail-quantile F (tn n)
for m n
    using tn-gt1[of m] tn-mono[of m n]
    by (intro tail-quantile-mono[OF ‹right-continuous-mono F 0 1›]) auto
  have not-bdd: ¬ bdd-above (range (λn. tail-quantile F (tn n)))
  proof
    assume bdd: bdd-above (range (λn. tail-quantile F (tn n)))
    have incseq: incseq (λn. tail-quantile F (tn n))
      using U-mono-seq by (intro incseq-SucI) simp
    obtain B where B: ⋀n. tail-quantile F (tn n) ≤ B
      using bdd unfolding bdd-above-def by auto
    obtain L where L: (λn. tail-quantile F (tn n)) ⟶ L
      using incseq-convergent[OF incseq, of B] B by meson
    have diff0: (λn. tail-quantile F (tn (Suc n)) - tail-quantile F (tn n)) ⟶
0
      using tendsto-diff[OF LIMSEQ-Suc[OF L] L] by simp
    have diff-gt: ∀F n in sequentially. c * a (tn n)
                          < tail-quantile F (tn (Suc n)) - tail-quantile F (tn n)
      using T(2)[OF tn-ge-T] by (intro always-eventually) (simp add: tn-Suc)
    have ge1: ∀F n in sequentially. 1 ≤ c * a (tn n)
      using a-tn-at-top[unfolded filterlim-at-top, rule-format, of 1 / c]
      by eventually-elim (use c-pos in simp add: field-simps)
    have gt1-ev: ∀F n in sequentially. 1
                       < tail-quantile F (tn (Suc n)) - tail-quantile F (tn n)
      using ge1 diff-gt
```

```
      by eventually-elim simp
    have lt1-ev:
     ∀F n in sequentially. tail-quantile F (tn (Suc n)) − tail-quantile F (tn n) <
1
      using order-tendstoD(2)[OF diff0, of 1] by simp
    have ∀F n::nat in sequentially. False
      using gt1-ev lt1-ev by eventually-elim linarith
    thus False
      by simp
  qed
  have tn-ex: ⋀M. ∃n. M < tail-quantile F (tn n)
    using not-bdd unfolding bdd-above-def by (auto simp: not-le)
  show ?thesis
    unfolding filterlim-at-top
  proof
    fix M :: real
    obtain n where n: M < tail-quantile F (tn n)
      using tn-ex by blast
    show ∀F t in at-top. M ≤ tail-quantile F t
      using eventually-ge-at-top[of tn n]
    proof eventually-elim
      case (elim t)
      have tail-quantile F (tn n) ≤ tail-quantile F t
        using tn-gt1[of n] elim
        by (intro tail-quantile-mono[OF ‹right-continuous-mono F 0 1›]) auto
      with n show ?case
        by simp
    qed
  qed
 qed
 show ⋀y. 0 < y ⟹
    ((λt. tail-quantile F (t * y) / tail-quantile F t) ⟶ y powr γ) at-top
  by (intro tail-quantile-pos-ratio-of-power-ratio[OF F gamma U-at-top]
      tail-quantile-pos-power-ratio[OF F apos gamma quant U-at-top])
qed

lemma tail-quantile-ratio-pos-imp-tail-regularly-varying:
 fixes F :: real ⇒ real and γ :: real
 assumes F: right-continuous-mono F 0 1 and gamma: 0 < γ
  and Uinf: filterlim (tail-quantile F) at-top at-top
  and Urv: ⋀y. 0 < y ⟹
      ((λt. tail-quantile F (t * y) / tail-quantile F t) ⟶ y powr γ) at-top
 shows (∀x. F x < 1) ∧ regularly-varying (λx. 1 − F x) (− 1 / γ)
proof −
 interpret right-continuous-mono F 0 1
  by fact
 have unbdd: ¬ bdd-support F
 proof
  assume bdd: bdd-support F
```

```
  have Fx1: F (x-star F + 1) = 1
    using bdd cSup-upper[of x-star F + 1 {x. F x < 1}] f-le-b[of x-star F + 1]
    unfolding x-star-def by fastforce
  have le: tail-quantile F t ≤ x-star F + 1 if t1: 1 < t for t
    using Fx1 t1 by (simp add: ginv-le-iff)
  have ∀F t in at-top. x-star F + 1 < tail-quantile F t
    using Uinf[unfolded filterlim-at-top, rule-format, of x-star F + 2]
    by eventually-elim simp
  from eventually-gt-at-top[of 1::real] this
  have ∀F t in at-top. 1 < t ∧ x-star F + 1 < tail-quantile F t
    by (rule eventually-conj)
  then obtain t where 1 < t x-star F + 1 < tail-quantile F t
    using eventually-happens'[OF trivial-limit-at-top-linorder] by blast
  with le[of t] show False
    by simp
 qed
 have Flt1: ∀x. F x < 1
  using F-lt-1-unbdd-support[OF ‹right-continuous-mono F 0 1› unbdd] by blast
 define s where s = (λt::real. 1 / (1 - F t))
 have Ftop: (F ⟶ 1) at-top
  using top by simp
 have s-top: filterlim s at-top at-top
 proof -
  have Ftend: ((λt. 1 - F t) ⟶ 0) at-top
    using tendsto-diff[OF tendsto-const[of 1] Ftop] by simp
  have Fpos: ∀F t in at-top. 0 < 1 - F t
    using Flt1 by (auto intro!: always-eventually)
  from filterlim-inverse-at-top[OF Ftend Fpos]
  show ?thesis
    by (simp add: s-def inverse-eq-divide)
 qed
 have galois: tail-quantile F t ≤ z ⟷ t ≤ s z if t1: 1 < t for t z
 proof -
  have p-bounds: 0 < 1 - 1 / t 1 - 1 / t < 1
    using t1 by simp-all
  have tail-quantile F t ≤ z ⟷ 1 - 1 / t ≤ F z
    using ginv-le-iff[OF p-bounds, of z] by simp
  also have ... ⟷ t ≤ s z
    using Flt1 t1 by (simp add: s-def field-simps)
  finally show ?thesis .
 qed
 have s-pos: 0 < s w if 0 < w for w
  using Flt1 by (simp add: s-def)
 have s-ratio: ((λu. s (u * x) / s u) ⟶ x powr (1 / γ)) at-top
  if x: 0 < x for x
  using galois-inverse-ratio-tendsto[OF Uinf s-top s-pos galois gamma Urv x] .
 have tail-ratio: ((λu. (1 - F (u * x)) / (1 - F u)) ⟶ x powr (- 1 / γ))
at-top
  if x: 0 < x for x
```

```
  proof −
    have same: (λu. s (u ∗ x) / s u) = (λu. (1 − F u) / (1 − F (u ∗ x)))
      using Flt1 by (auto simp: s-def)
    have main: ((λu. (1 − F u) / (1 − F (u ∗ x))) ⟶ x powr (1 / γ)) at-top
      using s-ratio[OF x] unfolding same .
    have xr-ne: x powr (1 / γ) ≠ 0
      using x by simp
    have inv: ((λu. inverse ((1 − F u) / (1 − F (u ∗ x))))
                ⟶ inverse (x powr (1 / γ))) at-top
      by (intro tendsto-inverse main xr-ne)
    have eq2: (λu. inverse ((1 − F u) / (1 − F (u ∗ x)))) = (λu. (1 − F (u ∗
x)) / (1 − F u))
      by simp
    have lim: ((λu. (1 − F (u ∗ x)) / (1 − F u)) ⟶ inverse (x powr (1 / γ)))
at-top
      using inv unfolding eq2 .
    have val-eq: inverse (x powr (1 / γ)) = x powr (− 1 / γ)
      using x by (simp add: powr-minus)
    show ?thesis
      using lim unfolding val-eq .
  qed
  have final-rv: regularly-varying (λx. 1 − F x) (− 1 / γ)
    using borel-measurable-mono[OF mono] Flt1 tail-ratio
    by(auto intro!: regularly-varyingI measurable-restrict-space1)
  show ?thesis
    using Flt1 final-rv by blast
qed

theorem MDA-regularly-varying-pos:
  fixes F :: real ⇒ real and γ :: real
  assumes F: right-continuous-mono F 0 1 and gamma[arith]: 0 < γ
  shows domain-of-attraction F (EVD-cdf γ 0 1)
        ⟷ (∀ x. F x < 1) ∧ regularly-varying (λx. 1 − F x) (− 1 / γ)
proof
  assume dom: domain-of-attraction F (EVD-cdf γ 0 1)
  obtain a where apos: ⋀t. 0 < a t
    and quant: ⋀x. 0 < x ⟹
        ((λt. (tail-quantile F (t ∗ x) − tail-quantile F t) / a t) ⟶ (x powr γ −
1) / γ) at-top
    using dom MDA-iff-tail-quantile'[OF F, of γ] gamma by auto
  show (∀ x. F x < 1) ∧ regularly-varying (λx. 1 − F x) (− 1 / γ)
    by (intro tail-quantile-ratio-pos-imp-tail-regularly-varying[OF F gamma]
        tail-quantile-pos-ratio[OF F apos gamma quant]
        tail-quantile-pos-at-top[OF F apos gamma quant])
next
  assume rv: (∀ x. F x < 1) ∧ regularly-varying (λx. 1 − F x) (− 1 / γ)
  hence unbdd: ⋀x. F x < 1 and rv': regularly-varying (λx. 1 − F x) (− (1 / γ))
    by auto
  have not-bdd: ¬ bdd-support F
```

```
    by (rule not-bdd-supportI'[OF unbdd])
  have endpoint-eq: endpoint-filter F = at-top
    using not-bdd by (simp add: endpoint-filter-def)
  define f where f = (λt::real. γ * max t 1)
  have f-pos: ⋀t. 0 < f t
    unfolding f-def by simp
  have ratio: ((λt. (1 − F (t + x * f t)) / (1 − F t)) ⟶ − ln (EVD-cdf γ 0 1
x)) at-top
    if supp: 1 + γ * x > 0 for x
  proof −
    have eq1: ∀F t in at-top. (1 − F (t * (1 + γ * x))) / (1 − F t)
                = (1 − F (t + x * f t)) / (1 − F t)
      using eventually-ge-at-top[of 1::real]
      by eventually-elim (simp add: f-def field-simps)
    have main: ((λt. (1 − F (t * (1 + γ * x))) / (1 − F t))
                ⟶ (1 + γ * x) powr (− (1 / γ))) at-top
      using regularly-varying-lim[OF rv', of 1 + γ * x] supp by simp
    thus ?thesis
      using Lim-transform-eventually[OF main eq1]
        ln-EVD-cdf-1[OF zero-less-one, of γ x 0] supp gamma
      by simp
  qed
  have ∃f. (∀t. 0 < f t) ∧ (∀x. 1 + γ * x > 0 ⟶
            ((λt. (1 − F (t + x * f t)) / (1 − F t))
              ⟶ − ln (EVD-cdf γ 0 1 x)) (endpoint-filter F))
    using f-pos ratio endpoint-eq by (intro exI[of - f]) simp
  thus domain-of-attraction F (EVD-cdf γ 0 1)
    using MDA-iff-divide-tail[OF F, of γ] by simp
qed

theorem MDA-regularly-varying-neg:
  fixes F :: real ⇒ real and γ :: real
  assumes F: right-continuous-mono F 0 1 and gamma[arith]: γ < 0
  shows domain-of-attraction F (EVD-cdf γ 0 1)
        ⟷ bdd-support F ∧
          regularly-varying (λs. 1 − F (x-star F − 1 / s)) (1 / γ)
proof
  assume dom: domain-of-attraction F (EVD-cdf γ 0 1)
  obtain a where apos: ⋀t. 0 < a t
    and quant: ⋀x. 0 < x ⟹
      ((λt. (tail-quantile F (t * x) − tail-quantile F t) / a t)
        ⟶ (x powr γ − 1) / γ) at-top
    using dom MDA-iff-tail-quantile'[OF F, of γ] gamma by auto
  have bdd: bdd-support F
    by (rule tail-quantile-neg-bdd[OF F apos gamma quant])
  have gap-rv: ⋀y. 0 < y ⟹
    ((λt. (x-star F − tail-quantile F (t * y)) / (x-star F − tail-quantile F t))
      ⟶ y powr γ) at-top
    by (rule tail-quantile-neg-ratio[OF F apos gamma quant])
```

```
  show bdd-support F ∧ regularly-varying (λs. 1 − F (x-star F − 1 / s)) (1 / γ)
    using bdd tail-quantile-gap-ratio-neg-imp-tail-regularly-varying[OF F gamma
bdd gap-rv]
    by blast
next
  assume rv: bdd-support F ∧ regularly-varying (λs. 1 − F (x-star F − 1 / s))
(1 / γ)
  hence bdd: bdd-support F
    and rv': regularly-varying (λs. 1 − F (x-star F − 1 / s)) (1 / γ)
    by auto
  have lim:filterlim (λt. 1 / (c − t)) at-top (at-left c) for c :: real
  proof −
    have tendsto0: ((λt. c − t) ⟶ 0) (at-left c)
      by (auto intro!: tendsto-eq-intros)
    show ?thesis
      using filterlim-inverse-at-top[OF tendsto0 eventually-at-leftI[of c − 1]]
      by(auto simp: inverse-eq-divide)
  qed
  note Slim = lim[of x-star F]
  have endpoint-eq: endpoint-filter F = at-left (x-star F)
    using bdd by (simp add: endpoint-filter-def)
  define f where f = (λt::real. if t < x-star F then − γ ∗ (x-star F − t) else 1)
  have f-pos: ⋀t. 0 < f t
    unfolding f-def using gamma by (auto intro: mult-neg-pos)
  have ratio: ((λt. (1 − F (t + x ∗ f t)) / (1 − F t))
                ⟶ − ln (EVD-cdf γ 0 1 x)) (endpoint-filter F)
    if supp: 1 + γ ∗ x > 0 for x
  proof −
    have main: ((λs. (1 − F (x-star F − 1 / (s ∗ (1 / (1 + γ ∗ x))))))
                  / (1 − F (x-star F − 1 / s))) ⟶ (1 / (1 + γ ∗ x)) powr (1 /
γ)) at-top
      using regularly-varying-lim[OF rv', of 1 / (1 + γ ∗ x)] supp by simp
    have comp: ((λt. (1 − F (x-star F − 1 / ((1 / (x-star F − t)) ∗ (1 / (1 + γ
∗ x))))))
                  / (1 − F (x-star F − 1 / (1 / (x-star F − t)))))
                ⟶ (1 / (1 + γ ∗ x)) powr (1 / γ)) (at-left (x-star F))
      by (rule filterlim-compose[OF main Slim])
    have eq1: ∀F t in at-left (x-star F).
              (1 − F (x-star F − 1 / ((1 / (x-star F − t)) ∗ (1 / (1 + γ ∗ x))))))
                / (1 − F (x-star F − 1 / (1 / (x-star F − t))))
              = (1 − F (t + x ∗ f t)) / (1 − F t)
    proof (rule eventually-at-leftI[of x-star F − 1])
      fix t
      assume t ∈ {x-star F − 1<..<x-star F}
      hence tlt: t < x-star F
        by simp
      have a1: t + x ∗ f t = x-star F − (x-star F − t) ∗ (1 + γ ∗ x)
        using tlt by (simp add: f-def field-simps)
      have a2: 1 / (1 / (x-star F − t) ∗ (1 / (1 + γ ∗ x))) = (x-star F − t) ∗ (1
```

```
+ γ * x)
      using tlt supp by (simp add: field-simps)
    have a3: 1 / (1 / (x-star F - t)) = x-star F - t
      using tlt by simp
    show (1 - F (x-star F - 1 / (1 / (x-star F - t) * (1 / (1 + γ * x)))))
          / (1 - F (x-star F - 1 / (1 / (x-star F - t))))
        = (1 - F (t + x * f t)) / (1 - F t)
      unfolding a2 a3 a1 by simp
  qed simp
  have val-eq: (1 / (1 + γ * x)) powr (1 / γ) = - ln (EVD-cdf γ 0 1 x)
    using ln-EVD-cdf-1[OF zero-less-one, of γ x 0] supp gamma
    by (simp add: powr-divide powr-minus-divide)
  show ?thesis
    unfolding endpoint-eq
    using Lim-transform-eventually[OF comp eq1] val-eq
    by simp
 qed
 have ∃f. (∀t. 0 < f t) ∧ (∀x. 1 + γ * x > 0 ⟶
          ((λt. (1 - F (t + x * f t)) / (1 - F t))
          ⟶⟶ - ln (EVD-cdf γ 0 1 x)) (endpoint-filter F))
  using f-pos ratio by (intro exI[of - f]) simp
 thus domain-of-attraction F (EVD-cdf γ 0 1)
  using MDA-iff-divide-tail[OF F, of γ] by simp
qed
```

Corollary 1.2.4

```
lemma MDA-aux-fun-cong:
 fixes F a a' :: real ⇒ real and γ :: real
 assumes a: ⋀t. 0 < a t
  and quant: ⋀x. 0 < x ⟹
      ((λt. (tail-quantile F (t * x) - tail-quantile F t) / a t)
        ⟶⟶ (if γ = 0 then ln x else (x powr γ - 1) / γ)) at-top
  and equiv: ((λt. a' t / a t) ⟶⟶ 1) at-top
  and x: 0 < x
 shows ((λt. (tail-quantile F (t * x) - tail-quantile F t) / a' t)
        ⟶⟶ (if γ = 0 then ln x else (x powr γ - 1) / γ)) at-top
proof -
 have anz: a t ≠ 0 for t
  using a[of t] by simp
 show ?thesis
  using equiv
  by(intro asymp-equiv-tendsto-transfer[OF - quant[OF x]])
    (simp add: asymp-equivI' asymp-equiv-divide asymp-equiv-symI)
qed

lemma MDA-canonical-normalization:
 fixes F :: real ⇒ real and γ :: real
 assumes F: right-continuous-mono F 0 1
  and dom: domain-of-attraction F (EVD-cdf γ 0 1)
```

obtains $a :: real \Rightarrow real$
where $\bigwedge t.\ 0 < a\ t$
and $\bigwedge x.\ 0 < x \Longrightarrow$
$((\lambda t.\ (tail\text{-}quantile\ F\ (t * x) - tail\text{-}quantile\ F\ t)\ /\ a\ t)$
$\longrightarrow (if\ \gamma = 0\ then\ ln\ x\ else\ (x\ powr\ \gamma - 1)\ /\ \gamma))\ at\text{-}top$
and $\bigwedge x.\ (\lambda n.\ F\ (a\ (real\ n) * x + tail\text{-}quantile\ F\ (real\ n))\ \widehat{}\ n) \longrightarrow EVD\text{-}cdf\ \gamma\ 0\ 1\ x$
proof −
obtain a **where** $apos$: $\bigwedge t.\ 0 < a\ t$
and $quant$: $\bigwedge x.\ 0 < x \Longrightarrow$
$((\lambda t.\ (tail\text{-}quantile\ F\ (t * x) - tail\text{-}quantile\ F\ t)\ /\ a\ t)$
$\longrightarrow (if\ \gamma = 0\ then\ ln\ x\ else\ (x\ powr\ \gamma - 1)\ /\ \gamma))\ at\text{-}top$
using $dom\ MDA\text{-}iff\text{-}tail\text{-}quantile'[OF\ F,\ of\ \gamma]$ **by** $auto$
have wit: $(\lambda n.\ F\ (a\ (real\ n) * x + tail\text{-}quantile\ F\ (real\ n))\ \widehat{}\ n) \longrightarrow EVD\text{-}cdf\ \gamma\ 0\ 1\ x$ **for** x
proof $(rule\ MDA\text{-}tail\text{-}quantile\text{-}witness[$**where** $b = tail\text{-}quantile\ F,\ OF\ F\ apos\ zero\text{-}less\text{-}one])$
fix $y :: real$
assume y: $0 < y$
show $((\lambda t.\ (tail\text{-}quantile\ F\ (t * y) - tail\text{-}quantile\ F\ t)\ /\ a\ t)$
$\longrightarrow (if\ \gamma = 0\ then\ 1 * ln\ y + 0\ else\ 1 * (y\ powr\ \gamma - 1)\ /\ \gamma + 0))\ at\text{-}top$
using $quant[OF\ y]$ **by** $(cases\ \gamma = 0)\ simp\text{-}all$
qed
show $thesis$
by $(rule\ that[of\ a])\ (use\ apos\ quant\ wit$ **in** $blast)+$
qed

lemma $tail\text{-}quantile\text{-}pos\text{-}aux$:
fixes $F :: real \Rightarrow real$ **and** $\gamma :: real$
assumes F: $right\text{-}continuous\text{-}mono\ F\ 0\ 1$ **and** $gamma$: $0 < \gamma$
and dom: $domain\text{-}of\text{-}attraction\ F\ (EVD\text{-}cdf\ \gamma\ 0\ 1)$
and x: $0 < x$
shows $((\lambda t.\ (tail\text{-}quantile\ F\ (t * x) - tail\text{-}quantile\ F\ t)\ /\ (\gamma * tail\text{-}quantile\ F\ t))$
$\longrightarrow (x\ powr\ \gamma - 1)\ /\ \gamma)\ at\text{-}top$
proof −
obtain a **where** $apos$: $\bigwedge t.\ 0 < a\ t$
and $quant0$: $\bigwedge y.\ 0 < y \Longrightarrow$
$((\lambda t.\ (tail\text{-}quantile\ F\ (t * y) - tail\text{-}quantile\ F\ t)\ /\ a\ t)$
$\longrightarrow (if\ \gamma = 0\ then\ ln\ y\ else\ (y\ powr\ \gamma - 1)\ /\ \gamma))\ at\text{-}top$
using $dom\ MDA\text{-}iff\text{-}tail\text{-}quantile'[OF\ F,\ of\ \gamma]$ **by** $auto$
have $quant$: $\bigwedge y.\ 0 < y \Longrightarrow$
$((\lambda t.\ (tail\text{-}quantile\ F\ (t * y) - tail\text{-}quantile\ F\ t)\ /\ a\ t)$
$\longrightarrow (y\ powr\ \gamma - 1)\ /\ \gamma)\ at\text{-}top$
using $quant0\ gamma$ **by** $simp$
have pos: $\forall_F\ t\ in\ at\text{-}top.\ 0 < tail\text{-}quantile\ F\ t$
using $tail\text{-}quantile\text{-}pos\text{-}at\text{-}top[OF\ F\ apos\ gamma\ quant]$
unfolding $filterlim\text{-}at\text{-}top\text{-}dense$ **by** $blast$
have gnz: $\gamma \neq 0$

```
    using gamma by simp
  have ((λt. (tail-quantile F (t * x) / tail-quantile F t - 1) / γ)
        ⟶ (x powr γ - 1) / γ) at-top
    by (intro tendsto-intros tail-quantile-pos-ratio[OF F apos gamma quant x] gnz)
  moreover have ∀F t in at-top.
      (tail-quantile F (t * x) / tail-quantile F t - 1) / γ
        = (tail-quantile F (t * x) - tail-quantile F t) / (γ * tail-quantile F t)
    using pos by eventually-elim (simp add: field-simps)
  ultimately show ?thesis
    by (rule Lim-transform-eventually)
qed

lemma tail-quantile-neg-gap:
  fixes F :: real ⇒ real and γ :: real
  assumes F: right-continuous-mono F 0 1 and gamma: γ < 0
    and dom: domain-of-attraction F (EVD-cdf γ 0 1)
  shows tail-quantile-neg-bdd-support: bdd-support F
    and tail-quantile-neg-gap-pos: ∀F t in at-top. 0 < x-star F - tail-quantile F t
proof -
  have gnz: γ ≠ 0
    using gamma by simp
  obtain a where apos: ⋀t. 0 < a t
    and quant0: ⋀y. 0 < y ⟹
      ((λt. (tail-quantile F (t * y) - tail-quantile F t) / a t)
        ⟶ (if γ = 0 then ln y else (y powr γ - 1) / γ)) at-top
    using dom MDA-iff-tail-quantile'[OF F, of γ] by auto
  have quant: ⋀y. 0 < y ⟹
      ((λt. (tail-quantile F (t * y) - tail-quantile F t) / a t)
        ⟶ (y powr γ - 1) / γ) at-top
    using quant0 gnz by simp
  show bdd: bdd-support F
    by (rule tail-quantile-neg-bdd[OF F apos gamma quant])
  obtain f where fpos: ⋀t. 0 < f t
    and flim: ⋀y. 0 < 1 + γ * y ⟹
      ((λt. (1 - F (t + y * f t)) / (1 - F t))
        ⟶ - ln (EVD-cdf γ 0 1 y)) (endpoint-filter F)
    using dom MDA-iff-divide-tail[OF F, of γ] by auto
  have lim': ⋀y. 0 < 1 + γ * (y - 0) / 1 ⟹
     ((λt. (1 - F (t + (y - 0) * f t)) / (1 - F t))
      ⟶ - ln (EVD-cdf γ 0 1 y)) (endpoint-filter F)
    using flim by simp
  have Flim: (F ⟶ 1) (endpoint-filter F)
    by (intro F-tendsto-1-endpoint-of-lim[OF F - - zero-less-one lim']) (use fpos in
auto)
  have gap: 0 < x-star F - tail-quantile F t if t1: 1 < t for t
    using tail-quantile-lt-x-star[OF F bdd Flim t1] by simp
  show ∀F t in at-top. 0 < x-star F - tail-quantile F t
    using eventually-gt-at-top[of 1] by eventually-elim (rule gap)
qed
```

lemma *tail-quantile-neg-aux*:
fixes $F :: real \Rightarrow real$ **and** $\gamma :: real$
assumes *F*: *right-continuous-mono F 0 1* **and** *gamma*: $\gamma < 0$
and *dom*: *domain-of-attraction F (EVD-cdf γ 0 1)*
and *x*: $0 < x$
shows $((\lambda t.\ (tail\text{-}quantile\ F\ (t * x) - tail\text{-}quantile\ F\ t)$
$/\ (-\gamma * (x\text{-}star\ F - tail\text{-}quantile\ F\ t)))$
$\longrightarrow (x\ powr\ \gamma - 1)\ /\ \gamma)\ at\text{-}top$
proof −
have *gnz*: $\gamma \neq 0$ **and** *gnz'*: $-\gamma \neq 0$
using *gamma* **by** *simp-all*
obtain *a* **where** *apos*: $\bigwedge t.\ 0 < a\ t$
and *quant0*: $\bigwedge y.\ 0 < y \Longrightarrow$
$((\lambda t.\ (tail\text{-}quantile\ F\ (t * y) - tail\text{-}quantile\ F\ t)\ /\ a\ t)$
$\longrightarrow (if\ \gamma = 0\ then\ ln\ y\ else\ (y\ powr\ \gamma - 1)\ /\ \gamma))\ at\text{-}top$
using *dom MDA-iff-tail-quantile'*[*OF F*, *of γ*] **by** *auto*
have *quant*: $\bigwedge y.\ 0 < y \Longrightarrow$
$((\lambda t.\ (tail\text{-}quantile\ F\ (t * y) - tail\text{-}quantile\ F\ t)\ /\ a\ t)$
$\longrightarrow (y\ powr\ \gamma - 1)\ /\ \gamma)\ at\text{-}top$
using *quant0 gnz* **by** *simp*
note *gapev* = *tail-quantile-neg-gap-pos*[*OF F gamma dom*]
have $((\lambda t.\ (1 - (x\text{-}star\ F - tail\text{-}quantile\ F\ (t * x)))\ /\ (x\text{-}star\ F - tail\text{-}quantile\ F\ t))$
$/\ (-\gamma))$
$\longrightarrow (1 - x\ powr\ \gamma)\ /\ (-\gamma))\ at\text{-}top$
by (*intro tendsto-divide tendsto-diff tendsto-const gnz'*
tail-quantile-neg-ratio[*OF F apos gamma quant x*])
moreover have $(1 - x\ powr\ \gamma)\ /\ (-\gamma) = (x\ powr\ \gamma - 1)\ /\ \gamma$
by (*simp add*: *diff-divide-distrib*)
ultimately have $((\lambda t.\ (1 - (x\text{-}star\ F - tail\text{-}quantile\ F\ (t * x)))$
$/\ (x\text{-}star\ F - tail\text{-}quantile\ F\ t))\ /\ (-\gamma))$
$\longrightarrow (x\ powr\ \gamma - 1)\ /\ \gamma)\ at\text{-}top$
by *simp*
moreover have $\forall_F\ t\ in\ at\text{-}top.$
$(1 - (x\text{-}star\ F - tail\text{-}quantile\ F\ (t * x)))\ /\ (x\text{-}star\ F - tail\text{-}quantile\ F\ t))\ /\ (-\gamma)$
$= (tail\text{-}quantile\ F\ (t * x) - tail\text{-}quantile\ F\ t)$
$/\ (-\gamma * (x\text{-}star\ F - tail\text{-}quantile\ F\ t))$
using *gapev*
proof *eventually-elim*
case (*elim t*)
hence *ne*: $x\text{-}star\ F - tail\text{-}quantile\ F\ t \neq 0$
by *simp*
have $1 - (x\text{-}star\ F - tail\text{-}quantile\ F\ (t * x))\ /\ (x\text{-}star\ F - tail\text{-}quantile\ F\ t)$
$= (tail\text{-}quantile\ F\ (t * x) - tail\text{-}quantile\ F\ t)\ /\ (x\text{-}star\ F - tail\text{-}quantile\ F\ t)$
using *ne* **by** (*simp add*: *field-simps*)
thus *?case*

```
    by (simp add: mult.commute)
  qed
  ultimately show ?thesis
    by (rule Lim-transform-eventually)
qed

theorem MDA-normalization-pos:
  fixes F :: real ⇒ real and γ x :: real
  assumes F: right-continuous-mono F 0 1 and gamma: 0 < γ
    and dom: domain-of-attraction F (EVD-cdf γ 0 1)
  shows (λn. F (tail-quantile F (real n) * x) ^ n)
        ⟶ (if 0 < x then exp (− (x powr − (1 / γ))) else 0)
proof −
  have gnz: γ ≠ 0
    using gamma by simp
  obtain a where apos: ⋀t. 0 < a t
    and quant0: ⋀y. 0 < y ⟹
        ((λt. (tail-quantile F (t * y) − tail-quantile F t) / a t)
          ⟶ (if γ = 0 then ln y else (y powr γ − 1) / γ)) at-top
    using dom MDA-iff-tail-quantile'[OF F, of γ] by auto
  have quant: ⋀y. 0 < y ⟹
        ((λt. (tail-quantile F (t * y) − tail-quantile F t) / a t)
          ⟶ (y powr γ − 1) / γ) at-top
    using quant0 gnz by simp
  define a' where a' = (λt. γ * max (tail-quantile F t) 1)
  have a'pos: ⋀t. 0 < a' t
    using gamma by (simp add: a'-def)
  have Uev: ∀F t in at-top. 1 ≤ tail-quantile F t
    using tail-quantile-pos-at-top[OF F apos gamma quant]
    unfolding filterlim-at-top by blast
  have a'eq: ∀F t in at-top. a' t = γ * tail-quantile F t
    using Uev by eventually-elim (simp add: a'-def)
  have quant': ((λt. (tail-quantile F (t * y) − tail-quantile F t) / a' t)
                ⟶ (if γ = 0 then 1 * ln y + 0 else 1 * (y powr γ − 1) / γ +
0)) at-top
    if y: 0 < y for y
  proof −
    have ((λt. (tail-quantile F (t * y) − tail-quantile F t) / (γ * tail-quantile F t))
          ⟶ (y powr γ − 1) / γ) at-top
      by (rule tail-quantile-pos-aux[OF F gamma dom y])
    moreover have ∀F t in at-top.
        (tail-quantile F (t * y) − tail-quantile F t) / (γ * tail-quantile F t)
          = (tail-quantile F (t * y) − tail-quantile F t) / a' t
      using a'eq by eventually-elim simp
    ultimately show ?thesis
      using gnz by (simp add: Lim-transform-eventually)
  qed
  have wit: (λn. F (a' (real n) * y + tail-quantile F (real n)) ^ n) ⟶ EVD-cdf
γ 0 1 y for y
```

```
    by (rule MDA-tail-quantile-witness[where b = tail-quantile F,
          OF F a'pos zero-less-one quant'])
  have ∀F n in sequentially.
     F (a' (real n) * ((x - 1) / γ) + tail-quantile F (real n)) ^ n
       = F (tail-quantile F (real n) * x) ^ n
    using eventually-compose-filterlim[OF a'eq filterlim-real-sequentially]
    by eventually-elim (simp add: gnz field-simps)
  with wit[of (x - 1) / γ]
  have (λn. F (tail-quantile F (real n) * x) ^ n) ⟶ EVD-cdf γ 0 1 ((x - 1)
/ γ)
    by (rule Lim-transform-eventually)
  thus ?thesis
    using gamma by (auto simp: EVD-cdf-def)
qed

theorem MDA-normalization-neg:
  fixes F :: real ⇒ real and γ x :: real
  assumes F: right-continuous-mono F 0 1 and gamma: γ < 0
    and dom: domain-of-attraction F (EVD-cdf γ 0 1)
  shows (λn. F ((x-star F - tail-quantile F (real n)) * x + x-star F) ^ n)
          ⟶ (if x < 0 then exp (- ((- x) powr - (1 / γ))) else 1)
proof -
  have gnz: γ ≠ 0 and gpos: 0 < - γ
    using gamma by simp-all
  note gapev = tail-quantile-neg-gap-pos[OF F gamma dom]
  define a' where
    a' = (λt. if 0 < x-star F - tail-quantile F t
            then - γ * (x-star F - tail-quantile F t) else 1)
  have a'pos: 0 < a' t for t
    using gamma by (auto simp: a'-def intro!: mult-neg-pos)
  have a'eq: ∀F t in at-top. a' t = - γ * (x-star F - tail-quantile F t)
    using gapev by eventually-elim (simp add: a'-def)
  have quant': ((λt. (tail-quantile F (t * y) - tail-quantile F t) / a' t)
                  ⟶ (if γ = 0 then 1 * ln y + 0 else 1 * (y powr γ - 1) / γ +
0)) at-top
    if y: 0 < y for y
  proof -
    have ((λt. (tail-quantile F (t * y) - tail-quantile F t)
                / (- γ * (x-star F - tail-quantile F t)))
           ⟶ (y powr γ - 1) / γ) at-top
      by (rule tail-quantile-neg-aux[OF F gamma dom y])
    moreover have ∀F t in at-top.
      (tail-quantile F (t * y) - tail-quantile F t) / (- γ * (x-star F - tail-quantile
F t))
        = (tail-quantile F (t * y) - tail-quantile F t) / a' t
      using a'eq by eventually-elim simp
    ultimately show ?thesis
      using gnz by (simp add: Lim-transform-eventually)
  qed
```

```
  have wit: (λn. F (a' (real n) * y + tail-quantile F (real n)) ^ n) ⟶ EVD-cdf
γ 0 1 y
    for y
    by (rule MDA-tail-quantile-witness[where b = tail-quantile F,
          OF F a'pos zero-less-one quant'])
  have ∀F n in sequentially.
     F (a' (real n) * (− (x + 1) / γ) + tail-quantile F (real n)) ^ n
       = F ((x-star F − tail-quantile F (real n)) * x + x-star F) ^ n
    using eventually-compose-filterlim[OF a'eq filterlim-real-sequentially]
  proof eventually-elim
    case (elim n)
    have a' (real n) * (− (x + 1) / γ) + tail-quantile F (real n)
        = (x-star F − tail-quantile F (real n)) * x + x-star F
      unfolding elim using gnz by (simp add: field-simps)
    thus ?case
      by simp
  qed
  with wit[of − (x + 1) / γ]
  have (λn. F ((x-star F − tail-quantile F (real n)) * x + x-star F) ^ n)
        ⟶ EVD-cdf γ 0 1 (− (x + 1) / γ)
    by (rule Lim-transform-eventually)
  thus ?thesis
    using gamma by(auto simp: EVD-cdf-def)
qed

lemma tail-quantile-spacing-aux:
  fixes F :: real ⇒ real and γ c :: real
  assumes F: right-continuous-mono F 0 1
    and dom: domain-of-attraction F (EVD-cdf γ 0 1)
    and c: 1 < c and x: 0 < x
  shows ((λt. (tail-quantile F (t * x) − tail-quantile F t)
              / ((tail-quantile F (t * c) − tail-quantile F t)
                 / (if γ = 0 then ln c else (c powr γ − 1) / γ)))
         ⟶ (if γ = 0 then ln x else (x powr γ − 1) / γ)) at-top
proof −
  have c0: 0 < c
    using c by simp
  define D where D = (if γ = 0 then ln c else (c powr γ − 1) / γ)
  have D-pos: 0 < D
    using powr-less-mono[of 0 γ c] powr-less-mono[of γ 0 c] c
    by(cases γ = 0; cases 0 < γ)
      (auto simp: D-def intro!: divide-pos-pos divide-neg-neg)
  obtain a where apos: ⋀t. 0 < a t
    and quant: ⋀y. 0 < y ⟹
      ((λt. (tail-quantile F (t * y) − tail-quantile F t) / a t)
        ⟶ (if γ = 0 then ln y else (y powr γ − 1) / γ)) at-top
    using dom MDA-iff-tail-quantile'[OF F, of γ] by auto
  have quantc: ((λt. (tail-quantile F (t * c) − tail-quantile F t) / a t) ⟶ D)
at-top
```

```
    using quant[OF c0] by(simp add: D-def)
  define a' where a' t = (tail-quantile F (t * c) - tail-quantile F t) / D for t
  have equiv: ((λt. a' t / a t) ⟶ 1) at-top
  proof -
    have eq: (tail-quantile F (t * c) - tail-quantile F t) / a t / D = a' t / a t for t
      by(simp add: a'-def mult.commute)
    have ((λt. (tail-quantile F (t * c) - tail-quantile F t) / a t / D) ⟶ D / D)
at-top
      using D-pos by(intro tendsto-divide[OF quantc tendsto-const]) simp
    thus ?thesis
      using D-pos unfolding eq by simp
  qed
  show ?thesis
    using MDA-aux-fun-cong[OF apos quant equiv x, unfolded a'-def D-def] .
qed

theorem MDA-normalization-zero-spacing:
  fixes F :: real ⇒ real and c x :: real
  assumes F: right-continuous-mono F 0 1 and c: 1 < c
    and dom: domain-of-attraction F (EVD-cdf 0 0 1)
  shows (λn. F ((tail-quantile F (real n * c) - tail-quantile F (real n)) / ln c * x
              + tail-quantile F (real n)) ^ n)
        ⟶ EVD-cdf 0 0 1 x
proof -
  have c0: 0 < c
    using c by simp
  have lnc: 0 < ln c
    using c by simp
  obtain a where apos: ⋀t. 0 < a t
    and quant: ⋀y. 0 < y ⟹
      ((λt. (tail-quantile F (t * y) - tail-quantile F t) / a t) ⟶ ln y) at-top
    using dom MDA-iff-tail-quantile'[OF F, of 0] by auto
  define a' where a' t = (tail-quantile F (t * c) - tail-quantile F t) / ln c for t
  have spacing: ((λt. (tail-quantile F (t * y) - tail-quantile F t) / a' t) ⟶ ln
y) at-top
    if y: 0 < y for y
    using tail-quantile-spacing-aux[OF F dom c y] by(simp add: a'-def)
  have ev: ∀F t in at-top. 0 < a' t
  proof(safe intro!: eventually-mono[OF order-tendstoD(1)[OF quant[OF c0] lnc]])
    fix t
    assume 0 < (tail-quantile F (t * c) - tail-quantile F t) / a t
    thus 0 < a' t
      using apos[of t] lnc by(simp add: a'-def zero-less-divide-iff)
  qed
  define a'' where a'' t = (if 0 < a' t then a' t else 1) for t
  have a''-pos: 0 < a'' t for t
    by(simp add: a''-def)
  have a''-eq: ∀F t in at-top. a'' t = a' t
    using ev by eventually-elim (simp add: a''-def)
```

have *quant''*: ((λ*t*. (*tail-quantile F* (*t* ∗ *y*) − *tail-quantile F t*) / *a''* *t*) ⟶ *ln y*) *at-top*
if *y*: *0* < *y* **for** *y*
proof −
have ∀ *F* *t* *in* *at-top*. (*tail-quantile F* (*t* ∗ *y*) − *tail-quantile F t*) / *a'* *t*
= (*tail-quantile F* (*t* ∗ *y*) − *tail-quantile F t*) / *a''* *t*
using *a''-eq* **by** *eventually-elim simp*
from *Lim-transform-eventually*[*OF spacing*[*OF y*] *this*]
show *?thesis* **.**
qed
have *wit*: (λ*n*. *F* (*a''* (*real n*) ∗ *x* + *tail-quantile F* (*real n*)) ^ *n*) ⟶ *EVD-cdf 0 0 1 x*
by(*intro MDA-tail-quantile-witness*[**where** γ = *0* **and** μ = *0* **and** σ = *1* **and** *a* = *a''*
and *b* = *tail-quantile F*, *OF F a''-pos zero-less-one*])
(*use quant''* **in** *simp*)
have ∀ *F* *n* *in* *sequentially*.
F (*a''* (*real n*) ∗ *x* + *tail-quantile F* (*real n*)) ^ *n*
= *F* ((*tail-quantile F* (*real n* ∗ *c*) − *tail-quantile F* (*real n*)) / *ln c* ∗ *x*
+ *tail-quantile F* (*real n*)) ^ *n*
using *eventually-compose-filterlim*[*OF a''-eq filterlim-real-sequentially*]
by *eventually-elim* (*simp add*: *a'-def*)
with *wit* **show** *?thesis*
by (*rule Lim-transform-eventually*)
qed

end

4 Peaks Over Threshold

theory *Peaks-Over-Threshold*
imports
Domain-of-Attraction
begin

In this section, we formalize the Pickands–Balkema–de Haan theorem. Our formalization is stated in the form of Theorem 3.4.13(b) in the book by Embrechts et al. [3]. The original sources of the theorem are Theorems 3 and 4 in the paper by Balkema and de Haan [1] and Theorem 7 in the paper by Pickands [6].

4.1 Excess (Peaks-Over-Threshold) Distribution Function

definition *excess-cdf* :: (*real* ⇒ *real*) ⇒ *real* ⇒ *real* ⇒ *real* **where**
excess-cdf F u x = *1* − (*1* − *F* (*u* + *x*)) / (*1* − *F u*)

lemma *excess-cdf-def2*:*F u* < *1* ⟹ *excess-cdf F u x* = (*F* (*u* + *x*) − *F u*) / (*1* − *F u*)

by (*smt* (*verit*, *ccfv-threshold*) *add-divide-distrib divide-self-if excess-cdf-def*)

lemma *excess-cdf-0*[*simp*]: *F u* < *1* ⟹ *excess-cdf F u 0* = *0*
by (*simp add*: *excess-cdf-def*)

lemma *excess-cdf-eq-1-of-F-eq-1*:
assumes *F u* = *1*
shows *excess-cdf F u x* = *1*
using *assms* **by** (*simp add*: *excess-cdf-def*)

lemma *excess-cdf-mono*:
assumes *F*: *right-continuous-mono F 0 1* **and** *xy*: *x* ≤ *y*
shows *excess-cdf F u x* ≤ *excess-cdf F u y*
proof (*cases F u* < *1*)
assume *Fu*: *F u* < *1*
have *mono*: *F* (*u* + *x*) ≤ *F* (*u* + *y*)
using *xy* **by** (*intro monoD*[*OF right-continuous-mono.mono*[*OF F*]]) *simp*
have (*1* − *F* (*u* + *y*)) / (*1* − *F u*) ≤ (*1* − *F* (*u* + *x*)) / (*1* − *F u*)
using *mono Fu* **by** (*intro divide-right-mono*) *auto*
thus *?thesis*
by (*simp add*: *excess-cdf-def*)
next
case *False*
hence *F u* = *1*
using *right-continuous-mono.f-le-b*[*OF F*, *of u*] **by** *linarith*
thus *?thesis*
by (*simp add*: *excess-cdf-eq-1-of-F-eq-1*)
qed

lemma *excess-cdf-bounds*:
assumes *F*: *right-continuous-mono F 0 1* **and** *x*: *0* ≤ *x*
shows *0* ≤ *excess-cdf F u x excess-cdf F u x* ≤ *1*
proof −
have *0* ≤ *excess-cdf F u x* ∧ *excess-cdf F u x* ≤ *1*
proof (*cases F u* < *1*)
case *Fu*: *True*
have *le1*: *F* (*u* + *x*) ≤ *1*
using *right-continuous-mono.f-le-b*[*OF F*] **.**
have *ge*: *F u* ≤ *F* (*u* + *x*)
using *x* **by** (*intro monoD*[*OF right-continuous-mono.mono*[*OF F*]]) *simp*
have (*1* − *F* (*u* + *x*)) / (*1* − *F u*) ≤ *1*
using *ge Fu* **by** (*simp add*: *divide-le-eq*)
moreover have *0* ≤ (*1* − *F* (*u* + *x*)) / (*1* − *F u*)
using *le1 Fu* **by** (*simp add*: *divide-nonneg-pos*)
ultimately show *?thesis*
by (*simp add*: *excess-cdf-def*)
next
case *False*
hence *F u* = *1*

```
    using right-continuous-mono.f-le-b[OF F, of u] by linarith
  thus ?thesis
    by (simp add: excess-cdf-eq-1-of-F-eq-1)
 qed
 thus 0 ≤ excess-cdf F u x excess-cdf F u x ≤ 1
  by auto
qed

lemma excess-cdf-eq-1-beyond-support:
 assumes F: right-continuous-mono F 0 1 and bdd: bdd-support F
  and xu: x-star F ≤ u + x
 shows excess-cdf F u x = 1
proof −
 have F (x-star F) = 1
  by (intro F-x-star-eq-1 F bdd)
 moreover have F (x-star F) ≤ F (u + x)
  using xu by (intro monoD[OF right-continuous-mono.mono[OF F]])
 ultimately have F (u + x) = 1
  using right-continuous-mono.f-le-b[OF F, of u + x] by linarith
 thus ?thesis
  by (simp add: excess-cdf-def)
qed

lemma excess-tendsto-of-divide-tail:
 fixes F f :: real ⇒ real and γ x :: real
 assumes F: right-continuous-mono F 0 1
  and fpos: ⋀t. 0 < f t
  and lim: ⋀x. 1 + γ * x > 0 ⟹
            ((λt. (1 − F (t + x * f t)) / (1 − F t))
              ⟶ − ln (EVD-cdf γ 0 1 x)) (endpoint-filter F)
  and x: 0 ≤ x
 shows ((λt. excess-cdf F t (f t * x)) ⟶ GPD-cdf γ 0 1 x) (endpoint-filter F)
proof −
 have main: ((λt. excess-cdf F t (f t * y)) ⟶ GPD-cdf γ 0 1 y) (endpoint-filter
F)
  if y: 0 ≤ y and supp: 1 + γ * y > 0 for y
 proof −
  have ratio-lim: ((λt. (1 − F (t + y * f t)) / (1 − F t))
                ⟶ 1 − GPD-cdf γ 0 1 y) (endpoint-filter F)
   using lim[OF supp] EVD-tail-eq-GPD-cdf[OF supp y] by simp
  have ((λt. 1 − (1 − F (t + y * f t)) / (1 − F t))
        ⟶ 1 − (1 − GPD-cdf γ 0 1 y)) (endpoint-filter F)
   by (intro tendsto-diff tendsto-const ratio-lim)
  thus ?thesis
   by (simp add: excess-cdf-def mult.commute)
 qed
 show ?thesis
 proof (cases 1 + γ * x > 0)
  case True
```

```
    show ?thesis
      by (rule main[OF x True])
  next
    case False
    hence notsupp: 1 + γ * x ≤ 0
      by simp
    have gneg: γ < 0
    proof (rule ccontr)
      assume ¬ γ < 0
      hence 0 ≤ γ * x
        using x by (intro mult-nonneg-nonneg) auto
      with notsupp show False
        by simp
    qed
    have gpdx1: GPD-cdf γ 0 1 x = 1
      using x gneg notsupp by (auto simp: GPD-cdf-def[OF zero-less-one] Let-def)
    have gpdbdry: GPD-cdf γ 0 1 (− 1 / γ) = 1
      using gneg by (simp add: GPD-cdf-def[OF zero-less-one] Let-def)
    have gpd-cont: (GPD-cdf γ 0 1 ⟶ 1) (at (− 1 / γ))
      using isCont-GPD-cdf[OF zero-less-one, where ξ = γ and μ = 0 and x =
− 1 / γ] gpdbdry
      by (simp add: isCont-def)
    have gpd-cont-left: (GPD-cdf γ 0 1 ⟶ 1) (at-left (− 1 / γ))
      using gpd-cont by (simp add: filterlim-at-split)
    have le1: excess-cdf F t (f t * x) ≤ 1 for t
      using fpos[of t] x by (intro excess-cdf-bounds(2)[OF F]) simp
    have ((λt. excess-cdf F t (f t * x)) ⟶ 1) (endpoint-filter F)
      unfolding tendsto-iff dist-real-def
    proof safe
      fix e :: real
      assume e: 0 < e
      have ∀F y in at-left (− 1 / γ). 1 − e < GPD-cdf γ 0 1 y
        using order-tendstoD(1)[OF gpd-cont-left, of 1 − e] e by simp
      moreover have ∀F y in at-left (− 1 / γ). y < − 1 / γ ∧ 0 ≤ y
        using gneg by (intro eventually-at-leftI[of 0]) auto
      ultimately obtain x0 where x0: 1 − e < GPD-cdf γ 0 1 x0 x0 < − 1 / γ
0 ≤ x0
        using eventually-happens'[OF trivial-limit-at-left-real eventually-conj] by
blast
      have supp0: 1 + γ * x0 > 0
        using mult-strict-left-mono-neg[OF x0(2) gneg] gneg by simp
      have x0x: x0 < x
      proof (rule ccontr)
        assume ¬ x0 < x
        hence 1 + γ * x0 ≤ 1 + γ * x
          using gneg by (intro add-left-mono mult-left-mono-neg) auto
        with supp0 notsupp show False
          by simp
      qed
```

```
    have ∀F t in endpoint-filter F. 1 − e < excess-cdf F t (f t ∗ x0)
      using order-tendstoD(1)[OF main[OF x0(3) supp0] x0(1)] .
    thus ∀F t in endpoint-filter F. |excess-cdf F t (f t ∗ x) − 1| < e
    proof eventually-elim
      case (elim t)
      have excess-cdf F t (f t ∗ x0) ≤ excess-cdf F t (f t ∗ x)
        using fpos[of t] x0x
        by (intro excess-cdf-mono[OF F] mult-left-mono less-imp-le) auto
      with elim le1[of t] show ?case
        by (simp add: abs-less-iff)
    qed
  qed
  thus ?thesis
    using gpdx1 by simp
 qed
qed

lemma tail-probability-u-form:
 fixes F :: real ⇒ real and γ :: real
 assumes F: right-continuous-mono F 0 1
   and dom: domain-of-attraction F (EVD-cdf γ 0 1)
 shows ∃β. (∀u. 0 < β u) ∧
          (∀x≥0. ((λu. excess-cdf F u (β u ∗ x) − GPD-cdf γ 0 1 x)
                   ⟶ 0) (endpoint-filter F))
proof −
 obtain f where fpos: ⋀t. 0 < f t
   and lim: ⋀x. 1 + γ ∗ x > 0 ⟹
              ((λt. (1 − F (t + x ∗ f t)) / (1 − F t))
                ⟶ − ln (EVD-cdf γ 0 1 x)) (endpoint-filter F)
   using MDA-iff-divide-tail[OF F] dom by blast
 show ?thesis
 proof (safe intro!: exI[of - f])
   show ⋀u. 0 < f u
     using fpos by blast
 next
   fix x :: real
   assume x: 0 ≤ x
  from tendsto-diff[OF excess-tendsto-of-divide-tail[OF F fpos lim x] tendsto-const,
                    of GPD-cdf γ 0 1 x]
  show ((λu. excess-cdf F u (f u ∗ x) − GPD-cdf γ 0 1 x) ⟶ 0) (endpoint-filter
F)
     by simp
 qed
qed

lemma excess-pointwise-imp-uniform:
 fixes F :: real ⇒ real and β :: real ⇒ real and FF :: real filter
 assumes F: right-continuous-mono F 0 1 and βpos: ⋀u. 0 < β u
   and pconv: ⋀x. 0 ≤ x ⟹ ((λu. excess-cdf F u (β u ∗ x)) ⟶ GPD-cdf γ 0
```

1 x) *FF*
shows *uniform-limit* {*0*..} (λ*u x*. *excess-cdf F u* (*β u* ∗ *x*)) (*GPD-cdf γ 0 1*) *FF*
proof −
interpret *GPD*: *right-continuous-mono GPD-cdf γ 0 1 0 1*
by (*intro right-continuous-mono-GPD-cdf zero-less-one*)
interpret *F*: *right-continuous-mono F 0 1*
by *fact*
show *?thesis*
proof (*rule mono-on-UCT-atLeast*[*OF eventuallyI*[*OF mono-onI*] *eventuallyI mono-onI*])
fix *u* :: *real* **and** *x y* :: *real*
assume *x* ∈{*0*..} *x* ≤ *y*
thus *excess-cdf F u* (*β u* ∗ *x*) ≤ *excess-cdf F u* (*β u* ∗ *y*)
using *βpos*[*of u*] **by** (*intro excess-cdf-mono*[*OF F*] *mult-left-mono*) *auto*
next
fix *u* :: *real*
show ∀ *x*≥*0*. *excess-cdf F u* (*β u* ∗ *x*) ≤ *1*
using *βpos*[*of u*] *excess-cdf-bounds*[*OF F*] **by** *simp*
next
fix *x y* :: *real*
assume *x* ∈{*0*..} *x* ≤ *y*
thus *GPD-cdf γ 0 1 x* ≤ *GPD-cdf γ 0 1 y*
by (*intro monoD*[*OF GPD.mono*])
next
show *continuous-on* {*0*..} (*GPD-cdf γ 0 1*)
using *isCont-GPD-cdf* **by**(*auto intro*!: *continuous-at-imp-continuous-on*)
next
show (*GPD-cdf γ 0 1* ⟶ *1*) *at-top*
by (*intro GPD-cdf-at-top-1 zero-less-one*)
qed *fact*
qed

lemma *tail-probability-u-form-uniform*:
fixes *F* :: *real* ⇒ *real* **and** *γ* :: *real*
assumes *F*: *right-continuous-mono F 0 1*
and *dom*: *domain-of-attraction F* (*EVD-cdf γ 0 1*)
shows ∃ *β*. (∀ *u*. *0* < *β u*) ∧
uniform-limit {*0*..} (λ*u x*. *excess-cdf F u* (*β u* ∗ *x*)) (*GPD-cdf γ 0 1*)
(*endpoint-filter F*)
proof −
obtain *β* **where** *βpos*: ⋀*u*. *0* < *β u*
and *pconv0*: ⋀*x*. *0* ≤ *x* ⟹
((λ*u*. *excess-cdf F u* (*β u* ∗ *x*) − *GPD-cdf γ 0 1 x*) ⟶ *0*)
(*endpoint-filter F*)
using *tail-probability-u-form*[*OF F dom*] **by** *blast*
have *pconv*: ((λ*u*. *excess-cdf F u* (*β u* ∗ *x*)) ⟶ *GPD-cdf γ 0 1 x*) (*endpoint-filter F*)
if *x*: *0* ≤ *x* **for** *x*
using *tendsto-add*[*OF pconv0*[*OF x*] *tendsto-const*, *of GPD-cdf γ 0 1 x*] **by**

```
simp
  have uniform-limit {0..} (λu x. excess-cdf F u (β u * x)) (GPD-cdf γ 0 1)
(endpoint-filter F)
    by (intro excess-pointwise-imp-uniform[OF F βpos] pconv)
  thus ?thesis
    using βpos by blast
qed
```

4.2 Pickands–Balkema–de Haan Theorem

The proof of the direct half follows the proof of Theorem 3.4.13(b) in the same book: the pointwise statement is Theorem 1.1.6(4) of the book by de Haan and Ferreira [2], and uniformity follows since the limit is continuous. The converse half follows the route sketched in the proof of Theorem 3.4.5 of Embrechts et al. [3].

4.2.1 Direct Half

```
theorem Pickands-Balkema-de-Haan-forward:
  fixes F :: real ⇒ real and γ :: real
  assumes F: right-continuous-mono F 0 1
    and dom: domain-of-attraction F (EVD-cdf γ 0 1)
  shows ∃β. (∀u. 0 < β u) ∧
            uniform-limit {0..} (λu x. excess-cdf F u x − GPD-cdf γ 0 (β u) x)
              (λx. 0) (endpoint-filter F)
proof −
  obtain β where βpos: ⋀u. 0 < β u
    and ul: uniform-limit {0..} (λu x. excess-cdf F u (β u * x)) (GPD-cdf γ 0 1)
              (endpoint-filter F)
    using tail-probability-u-form-uniform[OF F dom] by blast
  have uniform-limit {0..} (λu x. excess-cdf F u x − GPD-cdf γ 0 (β u) x) (λx.
0)
         (endpoint-filter F)
  proof (rule uniform-limitI)
    fix ε :: real
    assume ε: 0 < ε
    show ∀F u in endpoint-filter F.
           ∀x∈{0..}. dist (excess-cdf F u x − GPD-cdf γ 0 (β u) x) 0 < ε
      using uniform-limitD[OF ul ε]
    proof eventually-elim
      case (elim u)
      show ?case
      proof
        fix x :: real
        assume x ∈ {0..}
        hence x: 0 ≤ x
          by simp
        have y: x / β u ∈ {0..}
```

using x $\beta pos[of\ u]$ **by** *simp*
have *dist-lt*: *dist* (*excess-cdf F u* (β u * (x / β u))) (*GPD-cdf* γ *0 1* (x / β u)) < ε
using *elim y* **by** *blast*
have *rescale*: β u * (x / β u) = x
using $\beta pos[of\ u]$ **by** *simp*
have *target*: *GPD-cdf* γ *0 1* (x / β u) = *GPD-cdf* γ *0* (β u) x
using *GPD-cdf-location-scale*[*OF* $\beta pos[of\ u]$, *of* γ x] **by** *simp*
show *dist* (*excess-cdf F u x* − *GPD-cdf* γ *0* (β u) x) *0* < ε
using *dist-lt* **unfolding** *rescale target* **by** (*simp add*: *dist-real-def*)
qed
qed
qed
thus *?thesis*
using βpos **by** *blast*
qed

4.2.2 Converse Half

lemma *GPD-level-nonneg*:
fixes γ x :: *real*
assumes *1* ≤ x
shows *0* ≤ (*if* γ = *0 then ln x else* (*x powr* γ − *1*) / γ)
by (*smt* (*verit*) *assms ge-one-powr-ge-zero ln-ge-zero nle-le powr-less-one powr-one-eq-one zero-le-divide-iff*)

lemma *GPD-level-pos*:
fixes γ x :: *real*
assumes *1* < x
shows *0* < (*if* γ = *0 then ln x else* (*x powr* γ − *1*) / γ)
by (*smt* (*verit*, *ccfv-SIG*) *assms divide-neg-neg divide-pos-pos gr-one-powr ln-one ln-strict-mono powr-less-one*)

lemma *GPD-level-supp*:
fixes γ x :: *real*
assumes *1* ≤ x
shows *1* + γ * (*if* γ = *0 then ln x else* (*x powr* γ − *1*) / γ) = (*if* γ = *0 then 1 else x powr* γ)
using *assms* **by** *simp*

lemma *GPD-tail-eq*:
assumes x: *0* ≤ x
shows *1* − *GPD-cdf* γ *0 1 x*
= (*if* γ = *0 then exp* (− x)
else if 0 ≤ *1* + γ * x *then* (*1* + γ * x) *powr* (− *1* / γ) *else 0*)
using x **by** (*simp add*: *GPD-cdf-def*[*OF zero-less-one*] *Let-def*)

lemma *GPD-tail-level*:
assumes x: *1* ≤ x

```
  shows 1 − GPD-cdf γ 0 1 ((if γ = 0 then ln x else (x powr γ − 1) / γ)) = 1
/ x
proof (cases γ = 0)
  case True
  thus ?thesis
   using x by (simp add: GPD-tail-eq GPD-level-nonneg exp-minus inverse-eq-divide)
next
  case False
  have x0: 0 < x
    using x by simp
  have supp: 1 + γ * (if γ = 0 then ln x else (x powr γ − 1) / γ) = x powr γ
    using GPD-level-supp[OF x, of γ] False by simp
  have 0 < x powr γ
    using x0 by simp
  hence 1 − GPD-cdf γ 0 1 (if γ = 0 then ln x else (x powr γ − 1) / γ)
        = (x powr γ) powr (− 1 / γ)
    using GPD-level-nonneg[OF x, of γ] False supp by (simp add: GPD-tail-eq)
  also have ... = x powr (γ * (− 1 / γ))
    using x0 by (simp add: powr-powr)
  also have ... = 1 / x
    using False x0 by (simp add: powr-minus-divide field-simps)
  finally show ?thesis .
qed

lemma GPD-tail-strict-antimono:
  assumes y: 0 ≤ y and yz: y < z and supp: 0 < 1 + γ * y
  shows 1 − GPD-cdf γ 0 1 z < 1 − GPD-cdf γ 0 1 y
proof −
  have z: 0 ≤ z
    using y yz by simp
  show ?thesis
  proof (cases γ = 0)
    case True
    thus ?thesis
      using y z yz by (simp add: GPD-tail-eq)
  next
    case nonzero: False
    show ?thesis
    proof (cases 0 < 1 + γ * z)
      case True
      show ?thesis
      proof (cases 0 < γ)
        case gpos: True
        have 1 + γ * y < 1 + γ * z
          using yz gpos by simp
        hence (1 + γ * z) powr (− 1 / γ) < (1 + γ * y) powr (− 1 / γ)
          using supp gpos by (intro powr-less-mono2-neg) auto
        thus ?thesis
          using y z supp True nonzero by (simp add: GPD-tail-eq)
```

```
    next
      case False
      hence gneg: γ < 0
        using nonzero by simp
      have 1 + γ * z < 1 + γ * y
        using yz gneg by simp
      hence (1 + γ * z) powr (− 1 / γ) < (1 + γ * y) powr (− 1 / γ)
        using True gneg by (intro powr-less-mono2) auto
      thus ?thesis
        using y z supp True nonzero by (simp add: GPD-tail-eq)
    qed
  next
    case False
    hence 1 − GPD-cdf γ 0 1 z ≤ 0
      using z nonzero by (auto simp: GPD-tail-eq)
    moreover have 0 < 1 − GPD-cdf γ 0 1 y
      using y supp by (simp add: GPD-tail-eq)
    ultimately show ?thesis
      by simp
  qed
 qed
qed

lemma GPD-tail-gt-level:
 assumes x: 1 < x and y: 0 ≤ y and lt: y < (if γ = 0 then ln x else (x powr γ
− 1) / γ)
 shows 1 / x < 1 − GPD-cdf γ 0 1 y
proof −
 have x1: 1 ≤ x
   using x by simp
 have supp: 0 < 1 + γ * y
 proof (cases γ = 0)
   case True
   thus ?thesis
     by simp
 next
   case False
   hence sup': 1 + γ * (if γ = 0 then ln x else (x powr γ − 1) / γ) = x powr γ
     using GPD-level-supp[OF x1, of γ] by simp
   have pos: 0 < x powr γ
     using x by simp
   show ?thesis
   proof (cases 0 < γ)
     case True
     hence 0 ≤ γ * y
       using y by simp
     thus ?thesis
       by simp
   next
```

```
    case False
    hence gneg: γ < 0
      using ‹γ ≠ 0› by simp
    have γ * (if γ = 0 then ln x else (x powr γ − 1) / γ) < γ * y
      by (rule mult-strict-left-mono-neg[OF lt gneg])
    thus ?thesis
      using sup' pos by linarith
  qed
 qed
 have 1 − GPD-cdf γ 0 1 ((if γ = 0 then ln x else (x powr γ − 1) / γ)) < 1 −
GPD-cdf γ 0 1 y
  by (intro GPD-tail-strict-antimono y lt supp)
 thus ?thesis
  using GPD-tail-level[OF x1, of γ] by simp
qed

lemma GPD-tail-lt-level:
 assumes x: 1 < x and gt: (if γ = 0 then ln x else (x powr γ − 1) / γ) < y
 shows 1 − GPD-cdf γ 0 1 y < 1 / x
proof −
 have x1: 1 ≤ x
  using x by simp
 note L0 = GPD-level-nonneg[OF x1]
 have supp: 0 < 1 + γ * (if γ = 0 then ln x else (x powr γ − 1) / γ)
  using GPD-level-supp[OF x1, of γ] x by (cases γ = 0) auto
 have 1 − GPD-cdf γ 0 1 y < 1 − GPD-cdf γ 0 1 ((if γ = 0 then ln x else (x
powr γ − 1) / γ))
  by (intro GPD-tail-strict-antimono L0 gt supp)
 thus ?thesis
  using GPD-tail-level[OF x1, of γ] by simp
qed
lemma endpoint-filter-eventually-at-left:
 fixes F :: real ⇒ real
 assumes ev: eventually P (endpoint-filter F)
 shows ∀F t in endpoint-filter F. eventually P (at-left t)
proof (cases bdd-support F)
 case True
 hence eq: endpoint-filter F = at-left (x-star F)
  by (simp add: endpoint-filter)
 obtain c where c: c < x-star F
  and P: ⋀z. c < z ⟹ z < x-star F ⟹ P z
  by (metis assms eq eventually-at-left-field)
 show ?thesis
  unfolding eq
 proof (rule eventually-at-leftI[where a = c])
  fix t :: real
  assume t ∈ {c<..<x-star F}
  hence t: c < t t < x-star F
   by auto
```

```
    show eventually P (at-left t)
    proof (rule eventually-at-leftI[where a = c])
      fix z :: real
      assume z ∈ {c<..<t}
      thus P z
        using t by (auto intro!: P)
    next
      show c < t
        using t by simp
    qed
  next
    show c < x-star F
      by (rule c)
  qed
next
  case False
  hence eq: endpoint-filter F = at-top
    by (simp add: endpoint-filter)
  obtain d where d: ⋀z. d ≤ z ⟹ P z
    using ev[unfolded eq] by (auto simp: eventually-at-top-linorder)
  show ?thesis
    unfolding eq
  proof (rule eventually-at-top-linorderI[where c = d + 1])
    fix t :: real
    assume t: d + 1 ≤ t
    show eventually P (at-left t)
    proof (rule eventually-at-leftI[where a = d])
      fix z :: real
      assume z ∈ {d<..<t}
      thus P z
        by (auto intro!: d)
    next
      show d < t
        using t by simp
    qed
  qed
qed

lemma excess-uniform-scaled-iff:
  fixes F :: real ⇒ real and β :: real ⇒ real and FF :: real filter
  assumes βpos: ⋀u. 0 < β u
  shows uniform-limit {0..} (λu x. excess-cdf F u x − GPD-cdf γ 0 (β u) x) (λx.
0) FF
        ⟷ uniform-limit {0..} (λu x. excess-cdf F u (β u ∗ x)) (GPD-cdf γ 0 1)
FF
proof
  assume ul: uniform-limit {0..} (λu x. excess-cdf F u x − GPD-cdf γ 0 (β u) x)
(λx. 0) FF
  show uniform-limit {0..} (λu x. excess-cdf F u (β u ∗ x)) (GPD-cdf γ 0 1) FF
```

```
  proof (rule uniform-limitI)
    fix ε :: real
    assume ε: 0 < ε
    show ∀F u in FF. ∀y∈{0..}. dist (excess-cdf F u (β u * y)) (GPD-cdf γ 0 1
y) < ε
    proof(safe intro!: eventually-mono[OF uniform-limitD[OF ul ε]])
      fix u y :: real
      assume h:∀z∈{0..}. dist (excess-cdf F u z − GPD-cdf γ 0 (β u) z) 0 < ε
        and y:0 ≤ y
      have mem: β u * y ∈ {0..}
        using βpos[of u] y by simp
      have target: GPD-cdf γ 0 (β u) (β u * y) = GPD-cdf γ 0 1 y
         using GPD-cdf-location-scale[OF βpos[of u], of γ β u * y] βpos[of u] by
simp
      have |excess-cdf F u (β u * y) − GPD-cdf γ 0 (β u) (β u * y)| < ε
        using h mem by (simp add: dist-real-def)
      thus dist (excess-cdf F u (β u * y)) (GPD-cdf γ 0 1 y) < ε
        using target by (simp add: dist-real-def)
    qed
  qed
next
  assume ul: uniform-limit {0..} (λu x. excess-cdf F u (β u * x)) (GPD-cdf γ 0
1) FF
  show uniform-limit {0..} (λu x. excess-cdf F u x − GPD-cdf γ 0 (β u) x) (λx.
0) FF
  proof (rule uniform-limitI)
    fix ε :: real
    assume ε: 0 < ε
    show ∀F u in FF. ∀x∈{0..}. dist (excess-cdf F u x − GPD-cdf γ 0 (β u) x)
0 < ε
    proof(safe intro!: eventually-mono[OF uniform-limitD[OF ul ε]])
      fix u x :: real
      assume h:∀y∈{0..}. dist (excess-cdf F u (β u * y)) (GPD-cdf γ 0 1 y) < ε
        and x:0 ≤ x
      hence x: 0 ≤ x
        by simp
      have mem: x / β u ∈ {0..}
        using βpos[of u] x by simp
      have rescale: excess-cdf F u (β u * (x / β u)) = excess-cdf F u x
        using βpos[of u] by simp
      have target: GPD-cdf γ 0 1 (x / β u) = GPD-cdf γ 0 (β u) x
        using GPD-cdf-location-scale[OF βpos[of u], of γ x] by simp
      have dist (excess-cdf F u (β u * (x / β u))) (GPD-cdf γ 0 1 (x / β u)) < ε
        using h mem by blast
      hence |excess-cdf F u x − GPD-cdf γ 0 (β u) x| < ε
        using rescale target by (simp add: dist-real-def)
      thus dist (excess-cdf F u x − GPD-cdf γ 0 (β u) x) 0 < ε
        by (simp add: dist-real-def)
    qed
```

```
  qed
qed

lemma excess-uniform-eventually:
  fixes F :: real ⇒ real and β :: real ⇒ real
  assumes F: right-continuous-mono F 0 1
    and ul: uniform-limit {0..} (λu x. excess-cdf F u (β u ∗ x)) (GPD-cdf γ 0 1)
          (endpoint-filter F)
    and δ: 0 < δ
  shows ∀ F t in endpoint-filter F. F t < 1 ∧
                        (∀ x≥0. |excess-cdf F t (β t ∗ x) − GPD-cdf γ 0 1 x| < δ)
  by(auto intro!: eventually-mono[OF eventually-conj
       [OF F-lt-1-endpoint[OF F] uniform-limitD[OF ul δ]]] simp: dist-real-def)

lemma excess-iterate-tail-small:
  fixes F :: real ⇒ real and β :: real ⇒ real
  assumes βpos: ⋀u. 0 < β u
    and Y0: 0 < Y0 and KY0: 1 / 2 ≤ GPD-cdf γ 0 1 Y0
    and cT: c < T
    and lt1: ⋀z. c < z ⟹ z < T ⟹ F z < 1
    and near: ⋀z. c < z ⟹ z < T ⟹
          |excess-cdf F z (β z ∗ Y0) − GPD-cdf γ 0 1 Y0| < 1 / 4
    and step: ⋀z. c < z ⟹ z < T ⟹ z + β z ∗ Y0 < T
    and δ: 0 < δ
  shows ∃ z. c < z ∧ z < T ∧ 1 − F z < δ
proof −
  define w :: nat ⇒ real where w = rec-nat ((c + T) / 2) (λn wn. wn + β wn
∗ Y0)
  have w0: w 0 = (c + T) / 2
    by (simp add: w-def)
  have wSuc: w (Suc n) = w n + β (w n) ∗ Y0 for n
    by (simp add: w-def)
  have inv: c < w n ∧ w n < T for n
  proof (induction n)
    case 0
    thus ?case
      using cT by (simp add: w0)
  next
    case (Suc n)
    have bY: 0 < β (w n) ∗ Y0
      using βpos[of w n] Y0 by simp
    have lo: c < w n + β (w n) ∗ Y0
      using Suc[THEN conjunct1] bY by linarith
    have hi: w n + β (w n) ∗ Y0 < T
      using Suc by (intro step) auto
    show ?case
      using lo hi by (simp add: wSuc)
  qed
  have wlo: c < w n and whi: w n < T for n
```

```
    using inv by blast+
  have pos: 0 < 1 − F (w n) for n
    using lt1[OF wlo whi] by simp
  have decay: 1 − F (w (Suc n)) ≤ 3 / 4 ∗ (1 − F (w n)) for n
  proof −
    have |excess-cdf F (w n) (β (w n) ∗ Y0) − GPD-cdf γ 0 1 Y0| < 1 / 4
      by (intro near wlo whi)
    hence e: 1 − excess-cdf F (w n) (β (w n) ∗ Y0) ≤ 3 / 4
      using KY0 by linarith
    have 1 − F (w n + β (w n) ∗ Y0)
          = (1 − excess-cdf F (w n) (β (w n) ∗ Y0)) ∗ (1 − F (w n))
      using lt1[OF wlo whi, of n] by (simp add: excess-cdf-def)
    also have ... ≤ 3 / 4 ∗ (1 − F (w n))
      using e pos[of n] by (intro mult-right-mono) auto
    finally show ?thesis
      by (simp add: wSuc)
  qed
  have geom: 1 − F (w n) ≤ (3 / 4) ^ n ∗ (1 − F (w 0)) for n
  proof (induction n)
    case 0
    thus ?case
      by simp
  next
    case (Suc n)
    have 1 − F (w (Suc n)) ≤ 3 / 4 ∗ (1 − F (w n))
      by (rule decay)
    also have ... ≤ 3 / 4 ∗ ((3 / 4) ^ n ∗ (1 − F (w 0)))
      using Suc by simp
    finally show ?case
      by simp
  qed
  have pow0: (λn. (3 / 4:: real) ^ n) ⟶ 0
    by (rule LIMSEQ-realpow-zero) auto
  have (λn. (3 / 4:: real) ^ n ∗ (1 − F (w 0))) ⟶ 0
    using tendsto-mult[OF pow0 tendsto-const[of 1 − F (w 0)]] by simp
  from order-tendstoD(2)[OF this δ] obtain n
    where (3 / 4 :: real) ^ n ∗ (1 − F (w 0)) < δ
    by (auto simp: eventually-sequentially)
  hence 1 − F (w n) < δ
    using geom[of n] by simp
  thus ?thesis
    using wlo whi by blast
qed

lemma excess-tail-small-below-x-star:
  fixes F :: real ⇒ real and β :: real ⇒ real
  assumes F: right-continuous-mono F 0 1 and βpos: ⋀u. 0 < β u
    and ul: uniform-limit {0..} (λu x. excess-cdf F u (β u ∗ x)) (GPD-cdf γ 0 1)
            (endpoint-filter F)
```

```
    and bdd: bdd-support F and δ: 0 < δ
  shows ∃z. z < x-star F ∧ 1 − F z < δ
proof −
  define Y0 where Y0 = (if γ = 0 then ln 2 else (2 powr γ − 1) / γ)
  have Y0pos: 0 < Y0
    unfolding Y0-def by (intro GPD-level-pos) simp
  have KY0: GPD-cdf γ 0 1 Y0 = 1 / 2
    unfolding Y0-def using GPD-tail-level[of 2 γ] by simp
  have eq: endpoint-filter F = at-left (x-star F)
    using bdd by (simp add: endpoint-filter)
  have ∃c<x-star F. ∀z>c. z < x-star F ⟶
        F z < 1 ∧ (∀x≥0. |excess-cdf F z (β z ∗ x) − GPD-cdf γ 0 1 x| < 1 / 4)
    using excess-uniform-eventually[OF F ul, of 1 / 4]
    unfolding eq eventually-at-left-field by simp
  then obtain c where c: c < x-star F
    and P: ⋀z. c < z ⟹ z < x-star F ⟹
          F z < 1 ∧ (∀x≥0. |excess-cdf F z (β z ∗ x) − GPD-cdf γ 0 1 x| < 1 /
4)
    by blast
  have near: |excess-cdf F z (β z ∗ Y0) − GPD-cdf γ 0 1 Y0| < 1 / 4
    if z: c < z z < x-star F for z
    using P[OF z] Y0pos by simp
  have step: z + β z ∗ Y0 < x-star F if z: c < z z < x-star F for z
  proof (rule ccontr)
    assume ¬ z + β z ∗ Y0 < x-star F
    hence ge: x-star F ≤ z + β z ∗ Y0
      by simp
    have excess-cdf F z (β z ∗ Y0) = 1
      by (rule excess-cdf-eq-1-beyond-support[OF F bdd ge])
    hence (1::real) / 2 < 1 / 4
      using near[OF z] KY0 by simp
    thus False
      by simp
  qed
  have ∃z. c < z ∧ z < x-star F ∧ 1 − F z < δ
  proof (rule excess-iterate-tail-small[OF βpos Y0pos])
    show 1 / 2 ≤ GPD-cdf γ 0 1 Y0
      by (simp add: KY0)
    show c < x-star F
      by (rule c)
    show ⋀z. c < z ⟹ z < x-star F ⟹ F z < 1
      using P by blast
    show ⋀z. c < z ⟹ z < x-star F ⟹ |excess-cdf F z (β z ∗ Y0) − GPD-cdf
γ 0 1 Y0| < 1 / 4
      by (rule near)
    show ⋀z. c < z ⟹ z < x-star F ⟹ z + β z ∗ Y0 < x-star F
      by (rule step)
    show 0 < δ
      by (rule δ)
```

```
  qed
  thus ?thesis
    by blast
qed

lemma F-tendsto-1-endpoint-of-excess:
  fixes F :: real ⇒ real and β :: real ⇒ real
  assumes F: right-continuous-mono F 0 1 and βpos: ⋀u. 0 < β u
    and ul: uniform-limit {0..} (λu x. excess-cdf F u (β u ∗ x)) (GPD-cdf γ 0 1)
            (endpoint-filter F)
  shows (F ⟶⟶ 1) (endpoint-filter F)
proof (cases bdd-support F)
  case False
  thus ?thesis
    using right-continuous-mono.top[OF F] by (simp add: endpoint-filter)
next
  case bdd: True
  interpret right-continuous-mono F 0 1
    by fact
  show ?thesis
    unfolding endpoint-filter(1)[OF bdd] order-tendsto-iff
  proof safe
    fix a :: real
    assume a: a < 1
    obtain z where z: z < x-star F 1 − F z < 1 − a
      using excess-tail-small-below-x-star[OF F βpos ul bdd, of 1 − a] a by auto
    show ∀F t in at-left (x-star F). a < F t
    proof (intro eventually-at-leftI[OF - z(1)])
      fix t
      assume t ∈ {z<..<x-star F}
      hence F z ≤ F t
        by (intro monoD[OF mono]) simp
      thus a < F t
        using z(2) by simp
    qed
  next
    fix a :: real
    assume 1 < a
    thus ∀F t in at-left (x-star F). F t < a
      using f-le-b[THEN order-le-less-trans]
      by (auto intro!: eventually-at-leftI[of x-star F − 1])
  qed
qed

lemma tail-quantile-asymp-inverse-of-excess:
  fixes F :: real ⇒ real and β :: real ⇒ real
  assumes F: right-continuous-mono F 0 1 and βpos: ⋀u. 0 < β u
    and ul: uniform-limit {0..} (λu x. excess-cdf F u (β u ∗ x)) (GPD-cdf γ 0 1)
            (endpoint-filter F)
```

```
  shows ((λs. s * (1 - F (tail-quantile F s))) ⟶ 1) at-top
proof -
  define Y0 where Y0 = (if γ = 0 then ln 2 else (2 powr γ - 1) / γ)
  have Y0pos: 0 < Y0
    unfolding Y0-def by (intro GPD-level-pos) simp
  have KY0: GPD-cdf γ 0 1 Y0 = 1 / 2
    unfolding Y0-def using GPD-tail-level[of 2 γ] by simp
  note Flim = F-tendsto-1-endpoint-of-excess[OF F βpos ul]
  note Uendpoint = tail-quantile-tendsto-endpoint[OF F Flim]
  show ?thesis
    unfolding tendsto-iff dist-real-def
  proof safe
    fix e :: real
    assume e: 0 < e
    define δ where δ = min (e / 16) (1 / 8)
    define η where η = min (e / 8) (1 / 8)
    have δpos: 0 < δ and δle: δ ≤ 1 / 8 and δe: δ ≤ e / 16
      using e by (simp-all add: δ-def)
    have ηpos: 0 < η and ηle: η ≤ 1 / 8 and ηe: η ≤ e / 8
      using e by (simp-all add: η-def)
    define Q where
      Q = (λz. F z < 1 ∧ (∀x≥0. |excess-cdf F z (β z * x) - GPD-cdf γ 0 1 x| <
δ))
    have evP: ∀F t in endpoint-filter F. eventually Q (at-left t)
      unfolding Q-def
      by (intro endpoint-filter-eventually-at-left excess-uniform-eventually[OF F ul
δpos])
    note evU = eventually-compose-filterlim[OF evP Uendpoint]
    show ∀F s in at-top. |s * (1 - F (tail-quantile F s)) - 1| < e
      using evU eventually-gt-at-top[of 1]
    proof eventually-elim
      case (elim s)
      define t0 where t0 = tail-quantile F s
      have s1: 1 < s
        using elim(2) .
      have spos: 0 < s
        using s1 by simp
      have fA: 1 - F t0 ≤ 1 / s
        using lt-tail-quantile-iff[OF F s1, of t0] by (simp add: t0-def)
      have fB: 1 / s < 1 - F w if w < t0 for w
        using lt-tail-quantile-iff[OF F s1] that by (simp add: t0-def)
      obtain c where c: c < t0 and Q: ⋀z. c < z ⟹ z < t0 ⟹ Q z
        using elim(1) eventually-at-left-field t0-def by blast
      have Qlt1: F z < 1 if c < z z < t0 for z
        using Q[OF that] by (simp add: Q-def)
      have Qnear: |excess-cdf F z (β z * x) - GPD-cdf γ 0 1 x| < δ
        if z: c < z z < t0 and x: 0 ≤ x for z x
        using Q[OF z] x by (simp add: Q-def)
      have ∃z. c < z ∧ z < t0 ∧ t0 ≤ z + β z * Y0
```

```
    proof (rule ccontr)
      assume ¬ (∃ z. c < z ∧ z < t0 ∧ t0 ≤ z + β z * Y0)
      hence step: ⋀z. c < z ⟹ z < t0 ⟹ z + β z * Y0 < t0
        by (auto simp: not-le)
      have ∃ z. c < z ∧ z < t0 ∧ 1 − F z < 1 / s
      proof (rule excess-iterate-tail-small[OF βpos Y0pos])
        show |excess-cdf F z' (β z' * Y0) − GPD-cdf γ 0 1 Y0| < 1 / 4
          if c < z' z' < t0 for z'
          using Qnear[OF that less-imp-le[OF Y0pos]] δle by simp
      qed(simp-all add: KY0 c Qlt1 step spos)
      then obtain z where z: c < z z < t0 1 − F z < 1 / s
        by blast
      show False
        using fB[OF z(2)] z(3) by simp
    qed
    then obtain z where z: c < z z < t0 and zA: t0 ≤ z + β z * Y0
      by blast
    define y0 where y0 = (t0 − z) / β z
    have βz: 0 < β z
      by (rule βpos)
    have y0pos: 0 < y0
      using z(2) βz by (simp add: y0-def)
    have y0-le: y0 ≤ Y0
      using zA βz by (simp add: y0-def field-simps)
    have y0-eq: z + β z * y0 = t0
      using βz by (simp add: y0-def)
    define B where B = 1 − GPD-cdf γ 0 1 y0
    have Bge: 1 / 2 ≤ B
    using monoD[OF right-continuous-mono.mono[OF right-continuous-mono-GPD-cdf[of
1]]
          y0-le,of γ 0] KY0
      by(simp add: B-def)
    have Bpos: 0 < B
      using Bge by simp
    have Fzpos: 0 < 1 − F z
      using Qlt1[OF z(1,2)] by simp
    have factor: 1 − F (z + h) = (1 − excess-cdf F z h) * (1 − F z) for h
      using Qlt1[OF z(1,2)] by (simp add: excess-cdf-def)
    have lower1: (B − δ) * (1 − F z) < 1 − F t0
    proof −
      have |excess-cdf F z (β z * y0) − GPD-cdf γ 0 1 y0| < δ
        using Qnear[OF z(1,2) less-imp-le[OF y0pos]] .
      hence B − δ < 1 − excess-cdf F z (β z * y0)
        by (simp add: B-def)
      moreover have 1 − F t0 = (1 − excess-cdf F z (β z * y0)) * (1 − F z)
        using factor[of β z * y0] y0-eq by simp
      ultimately show ?thesis
        using Fzpos by (simp add: mult-strict-right-mono)
    qed
```

```
    have upper1: 1 / s < (B + η + δ) * (1 - F z)
    proof -
      have cont: isCont (GPD-cdf γ 0 1) y0
        by (intro isCont-GPD-cdf zero-less-one)
      obtain ϱ where ϱ: 0 < ϱ
        and ϱnear0: ⋀y. y ≠ y0 ⟹ |y - y0| < ϱ ⟹
                    |GPD-cdf γ 0 1 y - GPD-cdf γ 0 1 y0| < η
        using LIM-D[OF cont[unfolded isCont-def] ηpos] by auto
      have ϱnear: |GPD-cdf γ 0 1 y - GPD-cdf γ 0 1 y0| < η if |y - y0| < ϱ
for y
        using ϱnear0[of y] that ηpos by (cases y = y0) auto
      define ϑ where ϑ = min (β z * ϱ / 2) ((t0 - z) / 2)
      have ϑpos: 0 < ϑ
        using βz ϱ z(2) by (simp add: ϑ-def)
      have ϑ1: ϑ ≤ β z * ϱ / 2
        unfolding ϑ-def by (rule min.cobounded1)
      have ϑ2: ϑ ≤ (t0 - z) / 2
        unfolding ϑ-def by (rule min.cobounded2)
      define w where w = t0 - ϑ
      have wlt: w < t0
        using ϑpos by (simp add: w-def)
      have wgt: z < w
        using ϑ2 z(2) by (simp add: w-def)
      define yw where yw = (w - z) / β z
      have yw-eq: z + β z * yw = w
        using βz by (simp add: yw-def)
      have yw0: 0 ≤ yw
        using wgt βz by (simp add: yw-def)
      have yw-diff: yw - y0 = - (ϑ / β z)
        using βz by (simp add: yw-def y0-def w-def field-simps)
      have |yw - y0| < ϱ
        using ϑ1 βz ϱ ϑpos by (simp add: yw-diff field-simps)
      hence |GPD-cdf γ 0 1 yw - GPD-cdf γ 0 1 y0| < η
        by (rule ϱnear)
      hence Byw: 1 - GPD-cdf γ 0 1 yw < B + η
        by (simp add: B-def)
      have |excess-cdf F z (β z * yw) - GPD-cdf γ 0 1 yw| < δ
        using Qnear[OF z(1,2) yw0] .
      hence 1 - excess-cdf F z (β z * yw) < B + η + δ
        using Byw by simp
      moreover have 1 - F w = (1 - excess-cdf F z (β z * yw)) * (1 - F z)
        using factor[of β z * yw] yw-eq by simp
      ultimately have 1 - F w < (B + η + δ) * (1 - F z)
        using Fzpos by (simp add: mult-strict-right-mono)
      thus ?thesis
        using fB[OF wlt] by simp
    qed
    have Bden: 0 < B + η + δ
      using Bpos ηpos δpos by simp
```

```
    have Bnum: 0 < B − δ
      using Bge δle by simp
    have (B − δ) / (B + η + δ) < s ∗ (1 − F t0)
    proof −
      have 1 / s < (B + η + δ) ∗ (1 − F z)
        by (rule upper1)
      hence (B − δ) ∗ (1 / s) < (B − δ) ∗ ((B + η + δ) ∗ (1 − F z))
        using Bnum by (intro mult-strict-left-mono) auto
      also have ... = (B + η + δ) ∗ ((B − δ) ∗ (1 − F z))
        by (simp add: algebra-simps)
      also have ... < (B + η + δ) ∗ (1 − F t0)
        using lower1 Bden by (simp add: mult-strict-left-mono)
      finally have (B − δ) ∗ (1 / s) < (B + η + δ) ∗ (1 − F t0) .
      hence (B − δ) / (B + η + δ) < s ∗ (1 − F t0)
        using Bden spos by (simp add: field-simps)
      thus ?thesis .
    qed
    moreover have 1 − 2 ∗ (η + 2 ∗ δ) ≤ (B − δ) / (B + η + δ)
    proof −
      have (B − δ) / (B + η + δ) = 1 − (η + 2 ∗ δ) / (B + η + δ)
        using Bden by (simp add: field-simps)
      moreover have (η + 2 ∗ δ) / (B + η + δ) ≤ 2 ∗ (η + 2 ∗ δ)
      proof −
        have 1 / 2 ≤ B + η + δ
          using Bge ηpos δpos by simp
        hence (η + 2 ∗ δ) / (B + η + δ) ≤ (η + 2 ∗ δ) / (1 / 2)
          using ηpos δpos Bden by (intro divide-left-mono) auto
        thus ?thesis
          by simp
      qed
      ultimately show ?thesis
        by simp
    qed
    moreover have s ∗ (1 − F t0) ≤ 1
      using fA spos by (simp add: field-simps)
    ultimately show ?case
      using ηe δe e by (simp add: t0-def abs-real-def)
  qed
 qed
qed

lemma tail-quantile-gap-of-excess:
  fixes F :: real ⇒ real and β :: real ⇒ real
  assumes F: right-continuous-mono F 0 1 and βpos: ⋀u. 0 < β u
    and ul: uniform-limit {0..} (λu x. excess-cdf F u (β u ∗ x)) (GPD-cdf γ 0 1)
            (endpoint-filter F)
    and x: 1 < x
  shows ((λs. (tail-quantile F (s ∗ x) − tail-quantile F s) / β (tail-quantile F s))
         ⟶ (if γ = 0 then ln x else (x powr γ − 1) / γ)) at-top
```

```
proof −
  note Flim = F-tendsto-1-endpoint-of-excess[OF F βpos ul]
  note Uendpoint = tail-quantile-tendsto-endpoint[OF F Flim]
  note P1 = tail-quantile-asymp-inverse-of-excess[OF F βpos ul]
  have xpos: 0 < x
    using x by simp
  define q where q = 1 / x
  have qpos: 0 < q
    using xpos by (simp add: q-def)
  show ?thesis
    unfolding tendsto-iff dist-real-def
  proof safe
    fix e :: real
    assume e: 0 < e
    define L where L = (if γ = 0 then ln x else (x powr γ − 1) / γ)
    have Lpos: 0 < L
      unfolding L-def by (intro GPD-level-pos x)
    define ε where ε = min (e / 2) (L / 2)
    have εpos: 0 < ε and εe: ε < e and εL: ε < L
      using e Lpos by (simp-all add: ε-def)
    define ym yp where ym = L − ε and yp = L + ε
    have ym0: 0 < ym
      using εL by (simp add: ym-def)
    have ymL: ym < L
      using εpos by (simp add: ym-def)
    have Lyp: L < yp
      using εpos by (simp add: yp-def)
    define Km Kp where Km = 1 − GPD-cdf γ 0 1 ym and Kp = 1 − GPD-cdf
γ 0 1 yp
    have Km-gt: q < Km
      unfolding Km-def q-def
      using ym0 ymL by (intro GPD-tail-gt-level[OF x]) (auto simp: L-def)
    have Kp-lt: Kp < q
      unfolding Kp-def q-def
      using Lyp by (intro GPD-tail-lt-level[OF x]) (simp add: L-def)
    have Kp0: 0 ≤ Kp
      using GPD-cdf-le-1
      unfolding Kp-def by simp
    define δ where δ = min ((q − Kp) / 2) ((Km − q) / 2)
    have δ1: δ ≤ (q − Kp) / 2
      unfolding δ-def by (rule min.cobounded1)
    have δ2: δ ≤ (Km − q) / 2
      unfolding δ-def by (rule min.cobounded2)
    have δpos: 0 < δ
      using Kp-lt Km-gt by (simp add: δ-def)
    have Kpδ: Kp + δ ≤ q
    proof −
      have 2 ∗ (Kp + δ) ≤ 2 ∗ Kp + (q − Kp)
        using δ1 by simp
```

```
    also have ... ≤ 2 * q
      using Kp-lt by simp
    finally show ?thesis
      by simp
  qed
  have Kmδ: q < Km - δ
  proof -
    have 2 * δ ≤ Km - q
      using δ2 by simp
    hence 2 * q < 2 * (Km - δ)
      using Km-gt by simp
    thus ?thesis
      by simp
  qed
  have Kmδpos: 0 < Km - δ
    using Kmδ qpos by linarith
  define r where r = q / (Km - δ)
  have r-lt1: r < 1
    unfolding r-def using Kmδ Kmδpos by simp
  define ζ where ζ = (1 - r) / 2
  have ζpos: 0 < ζ
    using r-lt1 by (simp add: ζ-def)
  have rζ: r < 1 - ζ
  proof -
    have 1 - ζ = (1 + r) / 2
      by (simp add: ζ-def field-simps)
    moreover have r < (1 + r) / 2
      using r-lt1 by simp
    ultimately show ?thesis
      by simp
  qed
  have key: q < (Km - δ) * (1 - ζ)
  proof -
    have (Km - δ) * r = q
      using Kmδpos by (simp add: r-def)
    moreover have (Km - δ) * r < (Km - δ) * (1 - ζ)
      using rζ Kmδpos by (intro mult-strict-left-mono) auto
    ultimately show ?thesis
      by simp
  qed
  define Q where
    Q = (λt. F t < 1 ∧ (∀ y≥0. |excess-cdf F t (β t * y) - GPD-cdf γ 0 1 y| <
δ))
  have evQ: ∀ F s in at-top. Q (tail-quantile F s)
    unfolding Q-def
    by (intro eventually-compose-filterlim[OF excess-uniform-eventually[OF F ul
δpos]]
        Uendpoint)
  have evP1: ∀ F s in at-top. 1 - ζ < s * (1 - F (tail-quantile F s))
```

```
    using order-tendstoD(1)[OF P1, of 1 − ζ] ζpos by simp
  have evx: ∀F s::real in at-top. 1 < s ∗ x
    using x by real-asymp
  show ∀F s in at-top. |(tail-quantile F (s ∗ x) − tail-quantile F s)
                          / β (tail-quantile F s)
                        − (if γ = 0 then ln x else (x powr γ − 1) / γ)| < e
    using evQ evP1 evx eventually-gt-at-top[of 1]
  proof eventually-elim
    case (elim s)
    define t0 where t0 = tail-quantile F s
    have s1: 1 < s and sx1: 1 < s ∗ x
      using elim(3,4) by simp-all
    have spos: 0 < s
      using s1 by simp
    have qs: q / s = 1 / (s ∗ x)
      using xpos spos by (simp add: q-def field-simps)
    have βt: 0 < β t0
      by (rule βpos)
    have Ft0: F t0 < 1
      using elim(1) by (simp add: Q-def t0-def)
    have Ft0pos: 0 < 1 − F t0
      using Ft0 by simp
    have near: |excess-cdf F t0 (β t0 ∗ y) − GPD-cdf γ 0 1 y| < δ if 0 ≤ y for
y
      using elim(1) that by (simp add: Q-def t0-def)
    have factor: 1 − F (t0 + h) = (1 − excess-cdf F t0 h) ∗ (1 − F t0) for h
      using Ft0 by (simp add: excess-cdf-def)
    have upper-tail: 1 − F t0 ≤ 1 / s
     using lt-tail-quantile-iff[OF F s1, of tail-quantile F s] by (simp add: t0-def)
    have lower-tail: 1 − ζ < s ∗ (1 − F t0)
      using elim(2) by (simp add: t0-def)
    have p-bounds: 0 < 1 − 1 / (s ∗ x) 1 − 1 / (s ∗ x) < 1
      using sx1 by simp-all
    have Uup: tail-quantile F (s ∗ x) ≤ t0 + β t0 ∗ yp
    proof −
      have |excess-cdf F t0 (β t0 ∗ yp) − GPD-cdf γ 0 1 yp| < δ
        using ym0 εpos by (intro near) (simp add: yp-def ym-def)
      hence 1 − excess-cdf F t0 (β t0 ∗ yp) < Kp + δ
        by (simp add: Kp-def)
      hence step1: 1 − F (t0 + β t0 ∗ yp) < (Kp + δ) ∗ (1 − F t0)
        using Ft0pos factor[of β t0 ∗ yp] by (simp add: mult-strict-right-mono)
      have step2: (Kp + δ) ∗ (1 − F t0) ≤ q ∗ (1 / s)
        using Kpδ upper-tail Ft0pos qpos by (intro mult-mono) auto
      have step3: q ∗ (1 / s) = 1 / (s ∗ x)
        using qs by simp
      have 1 − F (t0 + β t0 ∗ yp) ≤ 1 / (s ∗ x)
        using step1 step2 step3 by simp
      hence 1 − 1 / (s ∗ x) ≤ F (t0 + β t0 ∗ yp)
        by simp
```

```
      thus ?thesis
       using right-continuous-mono.ginv-le-iff[OF F p-bounds, of t0 + β t0 * yp]
by simp
    qed
    have Ulo: t0 + β t0 * ym < tail-quantile F (s * x)
    proof −
      have |excess-cdf F t0 (β t0 * ym) − GPD-cdf γ 0 1 ym| < δ
        using ym0 by (intro near) simp
      hence Km − δ < 1 − excess-cdf F t0 (β t0 * ym)
        by (simp add: Km-def)
      hence (Km − δ) * (1 − F t0) < 1 − F (t0 + β t0 * ym)
        using Ft0pos factor[of β t0 * ym] by (simp add: mult-strict-right-mono)
      hence s * ((Km − δ) * (1 − F t0)) < s * (1 − F (t0 + β t0 * ym))
        using spos by (intro mult-strict-left-mono) auto
      moreover have q < s * ((Km − δ) * (1 − F t0))
          using key mult-strict-left-mono[OF lower-tail Kmδpos] by (simp add:
algebra-simps)
      ultimately have q < s * (1 − F (t0 + β t0 * ym))
        by simp
      hence q / s < 1 − F (t0 + β t0 * ym)
        using spos by (simp add: divide-less-eq mult.commute)
      hence 1 / (s * x) < 1 − F (t0 + β t0 * ym)
        using qs by simp
      thus ?thesis
        using right-continuous-mono.ginv-le-iff[OF F p-bounds, of t0 + β t0 *
ym]
        by linarith
    qed
    have ym < (tail-quantile F (s * x) − t0) / β t0
      using Ulo βt by (simp add: field-simps)
    moreover have (tail-quantile F (s * x) − t0) / β t0 ≤ yp
      using Uup βt by (simp add: field-simps)
    ultimately show ?case
      using εe by (simp add: t0-def ym-def yp-def L-def abs-real-def)
  qed
 qed
qed

lemma GPD-level-mult:
 fixes γ y :: real
 assumes y: 0 < y
 shows (if γ = 0 then ln (2 * y) else ((2 * y) powr γ − 1) / γ)
        − (if γ = 0 then ln y else (y powr γ − 1) / γ)
       = y powr γ * (if γ = 0 then ln 2 else (2 powr γ − 1) / γ)
proof (cases γ = 0)
next
 case False
 have (2 * y) powr γ = 2 powr γ * y powr γ
   using y by (simp add: powr-mult)
```

```
  thus ?thesis
    using False by (simp add: field-simps)
qed(use y in simp add: ln-mult)

lemma GPD-level-inverse:
  fixes γ x :: real
  assumes x: 0 < x
  shows - ((if γ = 0 then ln (1 / x) else ((1 / x) powr γ - 1) / γ) * x powr γ)
        = (if γ = 0 then ln x else (x powr γ - 1) / γ)
proof (cases γ = 0)
  case True
  thus ?thesis
    using x by (simp add: ln-div)
next
  case False
  have xpowr: 0 < x powr γ
    using x by simp
  have - ((if γ = 0 then ln (1 / x) else ((1 / x) powr γ - 1) / γ) * x powr γ)
         = - ((1 / x powr γ - 1) / γ * x powr γ)
    using False by (simp add: divide-inverse inverse-powr)
  also have ... = - ((1 - x powr γ) / γ)
    using xpowr False by (simp add: field-simps)
  also have ... = (if γ = 0 then ln x else (x powr γ - 1) / γ)
    using False by (simp add: field-simps)
  finally show ?thesis .
qed

lemma tail-quantile-gap-of-excess-all:
  fixes F :: real ⇒ real and β :: real ⇒ real
  assumes F: right-continuous-mono F 0 1 and βpos: ⋀u. 0 < β u
    and ul: uniform-limit {0..} (λu x. excess-cdf F u (β u * x)) (GPD-cdf γ 0 1)
              (endpoint-filter F)
    and x: 0 < x
  shows ((λs. (tail-quantile F (s * x) - tail-quantile F s) / β (tail-quantile F s))
           ⟶ (if γ = 0 then ln x else (x powr γ - 1) / γ)) at-top
proof -
  define U where U = tail-quantile F
  define a where a = (λs. β (tail-quantile F s))
  have apos: 0 < a s for s
    by (simp add: a-def βpos)
  have anz: a s ≠ 0 for s
    using apos[of s] by simp
  have Q: ((λs. (U (s * y) - U s) / a s)
            ⟶ (if γ = 0 then ln y else (y powr γ - 1) / γ)) at-top if 1 < y for
y
    using tail-quantile-gap-of-excess[OF F βpos ul that] by (simp add: U-def a-def)
  have L2pos: 0 < (if γ = 0 then ln 2 else (2 powr γ - 1) / γ)
    by (intro GPD-level-pos) simp
  have shift: filterlim (λs::real. s * y) at-top at-top if 0 < y for y
```

```
    using that by real-asymp
  have RV: ((λs. a (s * y) / a s) ⟶ y powr γ) at-top if y: 1 < y for y
  proof −
    have ypos: 0 < y
      using y by simp
    have Q2y: ((λs. (U (s * (2 * y)) − U s) / a s)
                  ⟶ (if γ = 0 then ln (2 * y) else ((2 * y) powr γ − 1) / γ))
at-top
      using y by (intro Q) simp
    have eq: (U (s * (2 * y)) − U s) / a s − (U (s * y) − U s) / a s
            = (U (s * y * 2) − U (s * y)) / a s for s
    proof −
      have arg: s * (2 * y) = s * y * 2
        by (simp add: algebra-simps)
      have (U (s * (2 * y)) − U s) / a s − (U (s * y) − U s) / a s
          = ((U (s * (2 * y)) − U s) − (U (s * y) − U s)) / a s
        by (simp add: diff-divide-distrib)
      also have ... = (U (s * (2 * y)) − U (s * y)) / a s
        by simp
      finally show ?thesis
        by (simp add: arg)
    qed
    have Ndiff: ((λs. (U (s * y * 2) − U (s * y)) / a s)
                  ⟶ (if γ = 0 then ln (2 * y) else ((2 * y) powr γ − 1) / γ)
                    − (if γ = 0 then ln y else (y powr γ − 1) / γ)) at-top
    proof −
      have ((λs. (U (s * (2 * y)) − U s) / a s − (U (s * y) − U s) / a s)
          ⟶ (if γ = 0 then ln (2 * y) else ((2 * y) powr γ − 1) / γ)
            − (if γ = 0 then ln y else (y powr γ − 1) / γ)) at-top
        by (intro tendsto-diff Q2y Q y)
      thus ?thesis
        by (simp add: eq)
    qed
    have D: ((λs. (U (s * y * 2) − U (s * y)) / a (s * y))
          ⟶ (if γ = 0 then ln 2 else (2 powr γ − 1) / γ)) at-top
    proof −
      have ((λr. (U (r * 2) − U r) / a r)
          ⟶ (if γ = 0 then ln 2 else (2 powr γ − 1) / γ)) at-top
        by (intro Q) simp
      from filterlim-compose[OF this shift[OF ypos]] show ?thesis
        by (simp add: comp-def)
    qed
    have Dpos: ∀F s in at-top. 0 < (U (s * y * 2) − U (s * y)) / a (s * y)
      by (rule order-tendstoD(1)[OF D L2pos])
    have ((λs. ((U (s * y * 2) − U (s * y)) / a s)
              / ((U (s * y * 2) − U (s * y)) / a (s * y)))
          ⟶ ((if γ = 0 then ln (2 * y) else ((2 * y) powr γ − 1) / γ)
              − (if γ = 0 then ln y else (y powr γ − 1) / γ))
            / (if γ = 0 then ln 2 else (2 powr γ − 1) / γ)) at-top
```

```
    using L2pos by (intro tendsto-divide Ndiff D) simp
  moreover have ∀ F s in at-top. ((U (s * y * 2) - U (s * y)) / a s)
                        / ((U (s * y * 2) - U (s * y)) / a (s * y)) = a (s * y) / a s
    using apos anz
  by (auto intro!: eventually-mono[OF Dpos] simp: zero-less-divide-iff field-simps)
  ultimately have ((λs. a (s * y) / a s)
                  ⟶ ((if γ = 0 then ln (2 * y) else ((2 * y) powr γ - 1) / γ)
                      - (if γ = 0 then ln y else (y powr γ - 1) / γ))
                     / (if γ = 0 then ln 2 else (2 powr γ - 1) / γ)) at-top
    by (rule Lim-transform-eventually)
  thus ?thesis
    using GPD-level-mult[OF ypos, of γ] L2pos by simp
qed
have RVall: ((λs. a (s * y) / a s) ⟶ y powr γ) at-top if y: 0 < y for y
proof (cases 1 < y)
  case True
  thus ?thesis
    by (rule RV)
next
  case notgt: False
  show ?thesis
  proof (cases y = 1)
    case True
    thus ?thesis
      using anz by simp
  next
    case ne: False
    have ylt: y < 1
      using notgt ne by simp
    define μ where μ = 1 / y
    have μ1: 1 < μ
      using y ylt by (simp add: μ-def)
    have ((λr. a (r * μ) / a r) ⟶ μ powr γ) at-top
      by (rule RV[OF μ1])
    from filterlim-compose[OF this shift[OF y]]
    have ((λs. a (s * y * μ) / a (s * y)) ⟶ μ powr γ) at-top
      by (simp add: comp-def)
    moreover have s * y * μ = s for s
      using y by (simp add: μ-def)
    ultimately have inv: ((λs. a s / a (s * y)) ⟶ μ powr γ) at-top
      by simp
    have ypowr: 0 < y powr γ
      using y by simp
    have μpowr: μ powr γ = 1 / y powr γ
      using y by (simp add: μ-def divide-inverse inverse-powr)
    have μnz: μ powr γ ≠ 0
      using μ1 by simp
    from tendsto-inverse[OF inv μnz]
    have ((λs. inverse (a s / a (s * y))) ⟶ inverse (μ powr γ)) at-top .
```

```
    moreover have inverse (a s / a (s * y)) = a (s * y) / a s for s
      by (simp add: inverse-eq-divide)
    moreover have inverse (μ powr γ) = y powr γ
      using ypowr by (simp add: μpowr)
    ultimately show ?thesis
      by simp
  qed
qed
have res: ((λs. (U (s * x) - U s) / a s)
          ⟶ (if γ = 0 then ln x else (x powr γ - 1) / γ)) at-top
proof (cases 1 < x)
  case True
  thus ?thesis
    by (rule Q)
next
  case notgt: False
  show ?thesis
  proof (cases x = 1)
    case True
    thus ?thesis
      by simp
  next
    case ne: False
    have xlt: x < 1
      using notgt ne by simp
    define μ where μ = 1 / x
    have μ1: 1 < μ
      using x xlt by (simp add: μ-def)
    have ((λr. (U (r * μ) - U r) / a r)
          ⟶ (if γ = 0 then ln μ else (μ powr γ - 1) / γ)) at-top
      by (rule Q[OF μ1])
    from filterlim-compose[OF this shift[OF x]]
    have ((λs. (U (s * x * μ) - U (s * x)) / a (s * x))
          ⟶ (if γ = 0 then ln μ else (μ powr γ - 1) / γ)) at-top
      by (simp add: comp-def)
    moreover have s * x * μ = s for s
      using x by (simp add: μ-def)
    ultimately have A: ((λs. (U s - U (s * x)) / a (s * x))
                        ⟶ (if γ = 0 then ln μ else (μ powr γ - 1) / γ)) at-top
      by simp
    note B = RVall[OF x]
    from tendsto-mult[OF A B]
    have ((λs. ((U s - U (s * x)) / a (s * x)) * (a (s * x) / a s))
          ⟶ (if γ = 0 then ln μ else (μ powr γ - 1) / γ) * x powr γ) at-top .
    moreover have ((U s - U (s * x)) / a (s * x)) * (a (s * x) / a s)
                  = - ((U (s * x) - U s) / a s) for s
      using anz by (simp add: field-simps)
    ultimately have ((λs. - ((U (s * x) - U s) / a s))
                     ⟶ (if γ = 0 then ln μ else (μ powr γ - 1) / γ) * x powr
```

```
γ) at-top
      by simp
    from tendsto-minus[OF this]
    have ((λs. (U (s * x) - U s) / a s)
          ⟶ - ((if γ = 0 then ln μ else (μ powr γ - 1) / γ) * x powr γ))
at-top
      by simp
   moreover have - ((if γ = 0 then ln μ else (μ powr γ - 1) / γ) * x powr γ)
                 = (if γ = 0 then ln x else (x powr γ - 1) / γ)
     unfolding μ-def by (rule GPD-level-inverse[OF x])
    ultimately show ?thesis
      by simp
  qed
 qed
 thus ?thesis
  by (simp add: U-def a-def)
qed

theorem Pickands-Balkema-de-Haan-converse:
 fixes F :: real ⇒ real and γ :: real
 assumes F: right-continuous-mono F 0 1
  and hyp: ∃β. (∀u. 0 < β u) ∧
          uniform-limit {0..} (λu x. excess-cdf F u x - GPD-cdf γ 0 (β u) x)
           (λx. 0) (endpoint-filter F)
 shows domain-of-attraction F (EVD-cdf γ 0 1)
proof -
 obtain β where βpos: ⋀u. 0 < β u
  and ul0: uniform-limit {0..} (λu x. excess-cdf F u x - GPD-cdf γ 0 (β u) x)
(λx. 0)
           (endpoint-filter F)
  using hyp by blast
 have iff: uniform-limit {0..} (λu x. excess-cdf F u x - GPD-cdf γ 0 (β u) x)
(λx. 0)
           (endpoint-filter F)
         ⟷ uniform-limit {0..} (λu x. excess-cdf F u (β u * x)) (GPD-cdf γ 0
1)
            (endpoint-filter F)
  by (intro excess-uniform-scaled-iff βpos)
 have ul: uniform-limit {0..} (λu x. excess-cdf F u (β u * x)) (GPD-cdf γ 0 1)
          (endpoint-filter F)
  using iff ul0 by simp
 have ∃a. (∀t. 0 < a t) ∧
         (∀x>0. ((λt. (tail-quantile F (t * x) - tail-quantile F t) / a t)
                 ⟶ (if γ = 0 then ln x else (x powr γ - 1) / γ)) at-top)
 proof (intro exI[of - λs. β (tail-quantile F s)] conjI allI impI)
  show ⋀t. 0 < β (tail-quantile F t)
   by (rule βpos)
 next
  fix x :: real
```

assume *xpos*: $0 < x$
show $((\lambda t.\ (\textit{tail-quantile}\ F\ (t * x) - \textit{tail-quantile}\ F\ t) / \beta\ (\textit{tail-quantile}\ F\ t))$
$\longrightarrow (\textit{if}\ \gamma = 0\ \textit{then}\ \ln x\ \textit{else}\ (x\ \textit{powr}\ \gamma - 1) / \gamma))\ \textit{at-top}$
by (*rule tail-quantile-gap-of-excess-all*[*OF F βpos ul xpos*])
qed
thus *?thesis*
using *MDA-iff-tail-quantile'*[*OF F, of γ*] **by** *simp*
qed

theorem *Pickands-Balkema-de-Haan*:
fixes $F :: \textit{real} \Rightarrow \textit{real}$ **and** $\gamma :: \textit{real}$
assumes *F*: *right-continuous-mono F 0 1*
shows *domain-of-attraction F* (*EVD-cdf γ 0 1*)
$\longleftrightarrow (\exists \beta.\ (\forall u.\ 0 < \beta\ u) \wedge$
uniform-limit $\{0..\}$ $(\lambda u\ x.\ \textit{excess-cdf}\ F\ u\ x - \textit{GPD-cdf}\ \gamma\ 0\ (\beta\ u)\ x)$
$(\lambda x.\ 0)$ (*endpoint-filter F*))
using *Pickands-Balkema-de-Haan-forward*[*OF F*] *Pickands-Balkema-de-Haan-converse*[*OF F*]
by *blast*

4.2.3 Explicit Choice of β

lemma *excess-uniform-beta-cong*:
fixes $F :: \textit{real} \Rightarrow \textit{real}$ **and** $\beta\ \beta' :: \textit{real} \Rightarrow \textit{real}$ **and** *FF* :: *real filter*
assumes *ul*: *uniform-limit* $\{0..\}$ $(\lambda u\ x.\ \textit{excess-cdf}\ F\ u\ x - \textit{GPD-cdf}\ \gamma\ 0\ (\beta\ u)\ x)$ $(\lambda x.\ 0)$ *FF*
and *ev*: $\forall_F\ u\ \textit{in}\ FF.\ \beta'\ u = \beta\ u$
shows *uniform-limit* $\{0..\}$ $(\lambda u\ x.\ \textit{excess-cdf}\ F\ u\ x - \textit{GPD-cdf}\ \gamma\ 0\ (\beta'\ u)\ x)$ $(\lambda x.\ 0)$ *FF*
proof (*rule uniform-limitI*)
fix $\varepsilon :: \textit{real}$
assume ε: $0 < \varepsilon$
show $\forall_F\ u\ \textit{in}\ FF.\ \forall x \in \{0..\}.\ \textit{dist}\ (\textit{excess-cdf}\ F\ u\ x - \textit{GPD-cdf}\ \gamma\ 0\ (\beta'\ u)\ x)\ 0 < \varepsilon$
using *uniform-limitD*[*OF ul* ε] *ev* **by** *eventually-elim simp*
qed

Case: $\gamma > 0$

theorem *Pickands-Balkema-de-Haan-pos*:
fixes $F :: \textit{real} \Rightarrow \textit{real}$ **and** $\gamma :: \textit{real}$
assumes *F*: *right-continuous-mono F 0 1* **and** γ: $0 < \gamma$
and *dom*: *domain-of-attraction F* (*EVD-cdf γ 0 1*)
shows *uniform-limit* $\{0..\}$ $(\lambda u\ x.\ \textit{excess-cdf}\ F\ u\ x - \textit{GPD-cdf}\ \gamma\ 0\ (\gamma * u)\ x)$ $(\lambda x.\ 0)$
(*endpoint-filter F*)
proof −
have *unbdd*: $\forall x.\ F\ x < 1$ **and** *rv*: *regularly-varying* $(\lambda x.\ 1 - F\ x)\ (-\ 1\ /\ \gamma)$
using *MDA-regularly-varying-pos*[*OF F γ*] *dom* **by** *auto*
have *nbdd*: $\neg$ *bdd-support F*

by (*rule not-bdd-supportI'*) (*use unbdd* **in** *simp*)
note *ef* = *endpoint-filter*(*2*)[*OF nbdd*]
define β **where** β = (λ*u*::*real*. γ * *max u 1*)
have β*pos*: *0* < β *u* **for** *u*
using γ **by** (*simp add*: β-*def*)
have β*eq*: $\forall_F$ *u in at-top*. β *u* = γ * *u*
using *eventually-ge-at-top*[*of 1*::*real*] **by** *eventually-elim* (*simp add*: β-*def*)
have *pconv*: ((*λu. excess-cdf F u* (β *u* * *x*)) ⟶ *GPD-cdf* γ *0 1 x*) *at-top* **if** *x*: *0* ≤ *x* **for** *x*
proof −
have *xnn*: *0* ≤ γ * *x*
using γ *x* **by** *simp*
have *cpos*: *0* < *1* + γ * *x*
using *xnn* **by** *simp*
have ((*λu. (1 − F* (*u* * (*1* + γ * *x*))) / (*1* − *F u*)) ⟶ (*1* + γ * *x*) *powr* (− *1* / γ)) *at-top*
by (*rule regularly-varying-lim*[*OF rv cpos*])
from *tendsto-diff*[*OF tendsto-const*[*of 1*] *this*]
have *lim*: ((*λu. 1* − (*1* − *F* (*u* * (*1* + γ * *x*))) / (*1* − *F u*))
⟶ *1* − (*1* + γ * *x*) *powr* (− *1* / γ)) *at-top* .
have *eq*: $\forall_F$ *u in at-top*. *1* − (*1* − *F* (*u* * (*1* + γ * *x*))) / (*1* − *F u*)
= *excess-cdf F u* (β *u* * *x*)
using β*eq* **by** *eventually-elim* (*simp add*: *excess-cdf-def field-simps*)
have *m1*: − *1* ≤ γ * *x*
using *xnn* **by** *simp*
have *target*: *GPD-cdf* γ *0 1 x* = *1* − (*1* + γ * *x*) *powr* (− *1* / γ)
using *x* γ *m1* **by** (*simp add*: *GPD-cdf-def*)
show *?thesis*
using *Lim-transform-eventually*[*OF lim eq*] *target* **by** *simp*
qed
have *uniform-limit* {*0*..} (*λu x. excess-cdf F u* (β *u* * *x*)) (*GPD-cdf* γ *0 1*) *at-top*
by (*intro excess-pointwise-imp-uniform*[*OF F* β*pos*] *pconv*)
hence *uniform-limit* {*0*..} (*λu x. excess-cdf F u x* − *GPD-cdf* γ *0* (β *u*) *x*) (*λx.* *0*) *at-top*
using *excess-uniform-scaled-iff*[*OF* β*pos*] **by** *simp*
moreover have β*eq'*: $\forall_F$ *u in at-top*. γ * *u* = β *u*
using β*eq* **by** *eventually-elim simp*
ultimately show *?thesis*
using *excess-uniform-beta-cong* **by** (*simp add*: *ef*)
qed

Case: $\gamma < 0$

theorem *Pickands-Balkema-de-Haan-neg*:
fixes *F* :: *real* ⇒ *real* **and** γ :: *real*
assumes *F*: *right-continuous-mono F 0 1* **and** γ: γ < *0*
and *dom*: *domain-of-attraction F* (*EVD-cdf* γ *0 1*)
shows *uniform-limit* {*0*..} (*λu x. excess-cdf F u x* − *GPD-cdf* γ *0* (− γ * (*x-star F* − *u*)) *x*)
(*λx. 0*) (*endpoint-filter F*)

proof −
have *bdd*: *bdd-support F* **and** *rv*: *regularly-varying* (λ*s*. *1* − *F* (*x-star F* − *1* / *s*)) (*1* / γ)
using *MDA-regularly-varying-neg*[*OF F* γ] *dom* **by** *auto*
note *ef* = *endpoint-filter*(*1*)[*OF bdd*]
have *evlt*: ∀ *F u in at-left* (*x-star F*). *u* < *x-star F*
by (*auto simp*: *eventually-at-filter*)
define β **where** β = (λ*u*::*real*. *if u* < *x-star F then* − γ ∗ (*x-star F* − *u*) *else 1*)
have β*pos*: *0* < β *u* **for** *u*
using γ **by** (*auto simp*: β*-def intro*!: *mult-neg-pos*)
have β*eq*: ∀ *F u in at-left* (*x-star F*). β *u* = − γ ∗ (*x-star F* − *u*)
using *evlt* **by** *eventually-elim* (*simp add*: β*-def*)
have *stop*: *filterlim* (λ*u*. *1* / (*x-star F* − *u*)) *at-top* (*at-left* (*x-star F*))
proof −
have *tend*: ((λ*u*::*real*. *x-star F* − *u*) ⟶ *0*) (*at-left* (*x-star F*))
by (*auto intro*!: *tendsto-eq-intros*)
have *filterlim* (λ*u*. *inverse* (*x-star F* − *u*)) *at-top* (*at-left* (*x-star F*))
by (*intro filterlim-inverse-at-top*[*OF tend*] *eventually-mono*[*OF evlt*]) *simp*
thus *?thesis*
by (*simp add*: *inverse-eq-divide*)
qed
have *pconv*: ((λ*u*. *excess-cdf F u* (β *u* ∗ *x*)) ⟶ *GPD-cdf* γ *0 1 x*) (*at-left* (*x-star F*))
if *x*: *0* ≤ *x* **for** *x*
proof (*cases 0* < *1* + γ ∗ *x*)
case *supp*: *True*
hence *m1*: − *1* ≤ γ ∗ *x*
by *simp*
have *cpos*: *0* < *1* / (*1* + γ ∗ *x*)
using *supp* **by** *simp*
have ((λ*s*. (*1* − *F* (*x-star F* − *1* / (*s* ∗ (*1* / (*1* + γ ∗ *x*))))))
/ (*1* − *F* (*x-star F* − *1* / *s*))) ⟶ (*1* / (*1* + γ ∗ *x*)) *powr* (*1* / γ)) *at-top*
by (*rule regularly-varying-lim*[*OF rv cpos*])
from *filterlim-compose*[*OF this stop*]
have *lim0*: ((λ*u*. (*1* − *F* (*x-star F* − *1* / (*1* / (*x-star F* − *u*) ∗ (*1* / (*1* + γ ∗ *x*)))))
/ (*1* − *F* (*x-star F* − *1* / (*1* / (*x-star F* − *u*))))) ⟶ (*1* / (*1* + γ ∗ *x*)) *powr* (*1* / γ)) (*at-left* (*x-star F*))
by *simp*
have *eq0*: ∀ *F u in at-left* (*x-star F*).
(*1* − *F* (*x-star F* − *1* / (*1* / (*x-star F* − *u*) ∗ (*1* / (*1* + γ ∗ *x*)))))
/ (*1* − *F* (*x-star F* − *1* / (*1* / (*x-star F* − *u*))))
= (*1* − *F* (*u* + β *u* ∗ *x*)) / (*1* − *F u*)
using *evlt* β*eq*
proof *eventually-elim*
case (*elim u*)
hence *gap*: *0* < *x-star F* − *u*
by *simp*

```
    have 1 / (1 / (x-star F − u) * (1 / (1 + γ * x))) = (x-star F − u) * (1 +
γ * x)
      using gap supp by (simp add: field-simps)
    moreover have x-star F − (x-star F − u) * (1 + γ * x) = u + β u * x
      unfolding elim(2) by (simp add: algebra-simps)
    moreover have x-star F − 1 / (1 / (x-star F − u)) = u
      using gap by simp
    ultimately show ?case
      by simp
  qed
  have lim: ((λu. (1 − F (u + β u * x)) / (1 − F u))
            ⟶ (1 / (1 + γ * x)) powr (1 / γ)) (at-left (x-star F))
    by (rule Lim-transform-eventually[OF lim0 eq0])
  have powr-eq: (1 / (1 + γ * x)) powr (1 / γ) = (1 + γ * x) powr (− 1 / γ)
   using supp by (simp add: powr-divide powr-minus divide-inverse inverse-powr)
  from tendsto-diff[OF tendsto-const[of 1] lim]
  have ((λu. 1 − (1 − F (u + β u * x)) / (1 − F u))
        ⟶ 1 − (1 + γ * x) powr (− 1 / γ)) (at-left (x-star F))
    by (simp add: powr-eq)
  moreover have GPD-cdf γ 0 1 x = 1 − (1 + γ * x) powr (− 1 / γ)
    using x γ m1 by (simp add: GPD-cdf-def)
  ultimately show ?thesis
    by (simp add: excess-cdf-def)
next
  case False
  hence supp: 1 + γ * x ≤ 0
    by simp
  have ∀F u in at-left (x-star F). excess-cdf F u (β u * x) = 1
    using evlt βeq
  proof eventually-elim
    case (elim u)
    hence gap: 0 < x-star F − u
      by simp
    have (x-star F − u) * (1 + γ * x) ≤ 0
      using gap supp by (simp add: mult-nonneg-nonpos less-imp-le)
    hence x-star F ≤ u + − γ * (x-star F − u) * x
      by (simp add: algebra-simps)
    hence x-star F ≤ u + β u * x
      unfolding elim(2) by simp
    thus ?case
      by (rule excess-cdf-eq-1-beyond-support[OF F bdd])
  qed
  moreover have GPD-cdf γ 0 1 x = 1
    using x γ supp by (auto simp: GPD-cdf-def)
  ultimately show ?thesis
    by (simp add: tendsto-eventually)
qed
have uniform-limit {0..} (λu x. excess-cdf F u (β u * x)) (GPD-cdf γ 0 1)
      (at-left (x-star F))
```

by (*intro excess-pointwise-imp-uniform*[*OF F βpos*] *pconv*)
hence *uniform-limit* {*0*..} (λ*u x*. *excess-cdf F u x* − *GPD-cdf γ 0* (*β u*) *x*) (λ*x*. *0*)
(*at-left* (*x-star F*))
using *excess-uniform-scaled-iff*[*OF βpos*] **by** *simp*
moreover have ∀$_F$ *u in at-left* (*x-star F*). − *γ* ∗ (*x-star F* − *u*) = *β u*
using *βeq* **by** *eventually-elim simp*
ultimately show *?thesis*
using *excess-uniform-beta-cong* **by** (*simp add*: *ef*)
qed

Case: arbitrary γ

lemma *isCont-GPD-level*:
fixes *γ y* :: *real*
assumes *y*: *0* < *y*
shows *isCont* (λ*z*. *if γ* = *0 then ln z else* (*z powr γ* − *1*) / *γ*) *y*
using *y* **by** (*cases γ* = *0*) (*auto intro*!: *isCont-ln isCont-divide isCont-diff continuous-powr*)

lemma *tail-quantile-witness-of-gap*:
fixes *F a* :: *real* ⇒ *real* **and** *γ* :: *real*
assumes *F*: *right-continuous-mono F 0 1*
and *apos*: ⋀*t*. *0* < *a t*
and *quant*: ⋀*y*. *0* < *y* ⟹ ((λ*t*. (*tail-quantile F* (*t* ∗ *y*) − *tail-quantile F t*) / *a t*)
⟶ (*if γ* = *0 then ln y else* (*y powr γ* − *1*) / *γ*)) *at-top*
shows (λ*n*. *F* (*a* (*real n*) ∗ *x* + *tail-quantile F* (*real n*)) ^ *n*) ⟶ *EVD-cdf γ 0 1 x*
proof (*rule MDA-tail-quantile-witness*[**where** *b* = *tail-quantile F*, *OF F apos zero-less-one*])
fix *y* :: *real*
assume *y*: *0* < *y*
show ((λ*t*. (*tail-quantile F* (*t* ∗ *y*) − *tail-quantile F t*) / *a t*)
⟶ (*if γ* = *0 then 1* ∗ *ln y* + *0 else 1* ∗ (*y powr γ* − *1*) / *γ* + *0*)) *at-top*
using *quant*[*OF y*] **by** (*cases γ* = *0*) *simp-all*
qed

lemma *t-div-a-tendsto-zero-endpoint*:
fixes *F a* :: *real* ⇒ *real* **and** *γ* :: *real*
assumes *F*: *right-continuous-mono F 0 1*
and *apos*: ⋀*t*. *0* < *a t*
and *quant*: ⋀*y*. *0* < *y* ⟹ ((λ*t*. (*tail-quantile F* (*t* ∗ *y*) − *tail-quantile F t*) / *a t*)
⟶ (*if γ* = *0 then ln y else* (*y powr γ* − *1*) / *γ*)) *at-top*
shows ((λ*t*. (*t* − *tail-quantile F* (*1* / (*1* − *F t*))) / *a* (*1* / (*1* − *F t*))) ⟶ *0*)
(*endpoint-filter F*)
proof −
define *s* **where** *s* = (λ*t*::*real*. *1* / (*1* − *F t*))
have *goal-eq*: (λ*t*. (*t* − *tail-quantile F* (*1* / (*1* − *F t*))) / *a* (*1* / (*1* − *F t*)))
= (λ*t*. (*t* − *tail-quantile F* (*s t*)) / *a* (*s t*))

```
  by (simp add: s-def)
have stop: filterlim s at-top (endpoint-filter F)
  unfolding s-def
  by (rule s-tendsto-at-top-endpoint[where a = λn. a (real n)
        and b = λn. tail-quantile F (real n), OF F - zero-less-one
        tail-quantile-witness-of-gap[OF F apos quant]])
     (use apos in simp)
note Flt1 = F-lt-1-endpoint[OF F]
have spos: ∀F t in endpoint-filter F. 0 < s t
  using stop[unfolded filterlim-at-top-dense, rule-format, of 0] by simp
have conv: ((λt. (tail-quantile F (s t * y') − tail-quantile F (s t)) / a (s t))
              ⟶ (if γ = 0 then ln y' else (y' powr γ − 1) / γ))
             (endpoint-filter F) if y': |y' − 1| < 1 for y'
  by (rule filterlim-compose[OF quant stop]) (use y' in simp)
have low: ∀F t in endpoint-filter F.
            (tail-quantile F (s t * y') − tail-quantile F (s t)) / a (s t)
              ≤ (t − tail-quantile F (s t)) / a (s t)
  if y': (1::real) − 1 < y' y' < 1 for y'
  using stop[unfolded filterlim-at-top-dense, rule-format, of 1 / y'] spos Flt1
proof eventually-elim
  case (elim t)
  hence s-gt: 1 / y' < s t and s-pos: 0 < s t and Ft: F t < 1
    by auto
  have u1: 1 < s t * y'
    using mult-strict-right-mono[OF s-gt] y' by force
  have u2: s t * y' < 1 / (1 − F t)
    using mult-strict-left-mono[OF y'(2) s-pos] by (simp add: s-def)
  show ?case
    using apos[of s t] Ft u1 u2
    by (auto intro!: divide-right-mono tail-quantile-le-of-lt-inv[OF F])
qed
have up: ∀F t in endpoint-filter F. (t − tail-quantile F (s t)) / a (s t)
              ≤ (tail-quantile F (s t * y') − tail-quantile F (s t)) / a (s t)
  if y': (1::real) < y' y' < 1 + 1 for y'
  using spos Flt1
proof eventually-elim
  case (elim t)
  hence s-pos: 0 < s t and Ft: F t < 1
    by auto
  have u: 1 / (1 − F t) < s t * y'
    using mult-strict-left-mono[OF - s-pos] y' by (simp add: s-def)
  show ?case
    using apos[of s t]
    by (auto intro!: divide-right-mono
        less-imp-le[OF t-lt-tail-quantile-of-gt-inv[OF F Ft u]])
qed
have one: (if γ = 0 then ln 1 else (1 powr γ − 1) / γ) = 0
  by simp
show ?thesis
```

using *squeeze-tendsto-local*[*OF zero-less-one isCont-GPD-level*[*OF zero-less-one*] *conv low up*]
unfolding *goal-eq one* **.**
qed

lemma *divide-tail-of-gap*:
fixes $F\ a :: real \Rightarrow real$ **and** $\gamma\ x :: real$
assumes *F*: *right-continuous-mono F 0 1*
and *apos*: $\bigwedge t.\ 0 < a\ t$
and *quant*: $\bigwedge y.\ 0 < y \Longrightarrow ((\lambda t.\ (tail\text{-}quantile\ F\ (t * y) - tail\text{-}quantile\ F\ t)\ /\ a\ t)$
$\longrightarrow (if\ \gamma = 0\ then\ ln\ y\ else\ (y\ powr\ \gamma - 1)\ /\ \gamma))\ at\text{-}top$
and *x*: $0 < 1 + \gamma * x$
shows $((\lambda t.\ (1 - F\ (t + x * a\ (1\ /\ (1 - F\ t))))\ /\ (1 - F\ t))$
$\longrightarrow - ln\ (EVD\text{-}cdf\ \gamma\ 0\ 1\ x))\ (endpoint\text{-}filter\ F)$
proof −
define *s* **where** $s = (\lambda t{::}real.\ 1\ /\ (1 - F\ t))$
define *U* **where** $U = tail\text{-}quantile\ F$
have *goal-eq*: $(\lambda t.\ (1 - F\ (t + x * a\ (1\ /\ (1 - F\ t))))\ /\ (1 - F\ t))$
$= (\lambda t.\ (1 - F\ (t + x * a\ (s\ t)))\ /\ (1 - F\ t))$
by (*simp add*: *s-def*)
have *stop*: *filterlim s at-top* (*endpoint-filter F*)
unfolding *s-def*
by (*rule s-tendsto-at-top-endpoint*[**where** $a = \lambda n.\ a\ (real\ n)$
and $b = \lambda n.\ tail\text{-}quantile\ F\ (real\ n)$, *OF F - zero-less-one*
tail-quantile-witness-of-gap[*OF F apos quant*]])
(*use apos* **in** *simp*)
note *Flt1* = *F-lt-1-endpoint*[*OF F*]
have *delta-tendsto*: $((\lambda t.\ (t - U\ (s\ t))\ /\ a\ (s\ t)) \longrightarrow 0)\ (endpoint\text{-}filter\ F)$
using *t-div-a-tendsto-zero-endpoint*[*OF F apos quant*] **by** (*simp add*: *s-def U-def*)
have *op*: *open* $\{x'.\ 0 < 1 + \gamma * x'\}$
by (*auto intro*!: *open-Collect-less continuous-on-add continuous-on-mult*)
have $\forall_F\ x'\ in\ nhds\ x.\ x' \in \{x'.\ 0 < 1 + \gamma * x'\}$
using *x op* **by** (*intro eventually-nhds-in-open*) *auto*
hence *ev*: $\forall_F\ x'\ in\ nhds\ x.\ 0 < 1 + \gamma * x'$
by *eventually-elim simp*
then obtain η **where** *eta*: $0 < \eta$
and *eta-supp*: $\bigwedge x'.\ |x' - x| < \eta \Longrightarrow 0 < 1 + \gamma * x'$
using *eventually-nhds-metric*[*of* $\lambda x'.\ 0 < 1 + \gamma * x'\ x$] **by** (*auto simp*: *dist-real-def*)
have *Dcont*: *isCont* $(\lambda x'.\ ln\ (EVD\text{-}cdf\ \gamma\ 0\ 1\ x'))\ x$
using *EVD-cdf-supp-bounds*[*OF - zero-less-one*, *of* $\gamma\ x\ 0$] *x*
by (*auto intro*!: *isCont-ln' isCont-EVD-cdf*)
have *conv*: $((\lambda t.\ -((1 - F\ (a\ (s\ t) * x' + U\ (s\ t)))\ /\ (1 - F\ t)))$
$\longrightarrow ln\ (EVD\text{-}cdf\ \gamma\ 0\ 1\ x'))\ (endpoint\text{-}filter\ F)$
if x': $|x' - x| < \eta$ **for** x'
proof −
have *supp'*: $0 < 1 + \gamma * x'$

```
    using x' eta-supp by simp
  have quant': ((λt. (tail-quantile F (t * y) - tail-quantile F t) / a t)
                ⟶ (if γ = 0 then 1 * ln y + 0 else 1 * (y powr γ - 1) / γ +
0)) at-top
    if y: 0 < y for y
    using quant[OF y] by (cases γ = 0) simp-all
  have base: ((λt. t * (1 - F (a t * x' + U t))) ⟶ - ln (EVD-cdf γ 0 1 x'))
at-top
    unfolding U-def
    by (intro MDA-tail-quantile-to-tail-probability[where b = tail-quantile F,
        OF F apos zero-less-one quant']) (use supp' in auto)
  show ?thesis
   using tendsto-minus[OF filterlim-compose[OF base stop]] by (simp add: s-def)
 qed
 have low: ∀F t in endpoint-filter F. - ((1 - F (a (s t) * x' + U (s t))) / (1 -
F t))
                                    ≤ - ((1 - F (t + x * a (s t))) / (1 - F t))
  if x': x - η < x' x' < x for x'
 proof -
  have dlt: x' - x < 0
    using x' by simp
  have ev1: ∀F t in endpoint-filter F. x' - x < (t - U (s t)) / a (s t)
    using order-tendstoD(1)[OF delta-tendsto dlt] .
  show ?thesis
    using ev1 Flt1
  proof eventually-elim
    case (elim t)
    hence d: x' - x < (t - U (s t)) / a (s t) and Ft: F t < 1
      by auto
    have apos': 0 < a (s t)
      using apos by simp
    have (x' - x) * a (s t) < ((t - U (s t)) / a (s t)) * a (s t)
      using mult-strict-right-mono[OF d apos'] .
    also have ... = t - U (s t)
      using apos' by simp
    finally have ineq: (x' - x) * a (s t) < t - U (s t) .
    have a (s t) * x' + U (s t) < t + x * a (s t)
      using ineq by (simp add: field-simps)
    hence fmono: F (a (s t) * x' + U (s t)) ≤ F (t + x * a (s t))
      by (intro monoD[OF right-continuous-mono.mono[OF F]]) simp
    have (1 - F (t + x * a (s t))) / (1 - F t)
          ≤ (1 - F (a (s t) * x' + U (s t))) / (1 - F t)
      using fmono Ft by (intro divide-right-mono) auto
    thus ?case
      by simp
  qed
 qed
 have up: ∀F t in endpoint-filter F. - ((1 - F (t + x * a (s t))) / (1 - F t))
                                  ≤ - ((1 - F (a (s t) * x' + U (s t))) / (1 - F t))
```

```
    if x': x < x' x' < x + η for x'
  proof −
    have dpos: 0 < x' − x
      using x' by simp
    have ev1: ∀F t in endpoint-filter F. (t − U (s t)) / a (s t) < x' − x
      using order-tendstoD(2)[OF delta-tendsto dpos] .
    show ?thesis
      using ev1 Flt1
    proof eventually-elim
      case (elim t)
      hence d: (t − U (s t)) / a (s t) < x' − x and Ft: F t < 1
        by auto
      have apos': 0 < a (s t)
        using apos by simp
      have ((t − U (s t)) / a (s t)) * a (s t) < (x' − x) * a (s t)
        using mult-strict-right-mono[OF d apos'] .
      moreover have ((t − U (s t)) / a (s t)) * a (s t) = t − U (s t)
        using apos' by simp
      ultimately have ineq: t − U (s t) < (x' − x) * a (s t)
        by simp
      have t + x * a (s t) < a (s t) * x' + U (s t)
        using ineq by (simp add: field-simps)
      hence fmono: F (t + x * a (s t)) ≤ F (a (s t) * x' + U (s t))
        by (intro monoD[OF right-continuous-mono.mono[OF F]]) simp
      have (1 − F (a (s t) * x' + U (s t))) / (1 − F t)
            ≤ (1 − F (t + x * a (s t))) / (1 − F t)
        using fmono Ft by (intro divide-right-mono) auto
      thus ?case
        by simp
    qed
  qed
  show ?thesis
    unfolding goal-eq
    using tendsto-minus[OF squeeze-tendsto-local[OF eta Dcont conv low up]] by
simp
qed

theorem Pickands-Balkema-de-Haan-normalization:
  fixes F a :: real ⇒ real and γ :: real
  assumes F: right-continuous-mono F 0 1
    and apos: ⋀t. 0 < a t
    and quant: ⋀x. 0 < x ⟹ ((λt. (tail-quantile F (t * x) − tail-quantile F t) /
a t)
                  ⟶ (if γ = 0 then ln x else (x powr γ − 1) / γ)) at-top
  shows uniform-limit {0..} (λu x. excess-cdf F u x − GPD-cdf γ 0 (a (1 / (1 −
F u))) x)
          (λx. 0) (endpoint-filter F)
proof −
  define β where β = (λu. a (1 / (1 − F u)))
```

have βpos: $0 < \beta\ u$ **for** u
using *apos* **by** (*simp add*: β-*def*)
have *pconv*: $((\lambda u.\ excess\text{-}cdf\ F\ u\ (\beta\ u * x)) \longrightarrow GPD\text{-}cdf\ \gamma\ 0\ 1\ x)$ (*endpoint-filter* F)
if x: $0 \leq x$ **for** x
by (*rule excess-tendsto-of-divide-tail*[*OF F* βpos - x])
(*use divide-tail-of-gap*[*OF F apos quant*] **in** *simp add*: β-*def*)
have *uniform-limit* $\{0..\}$ $(\lambda u\ x.\ excess\text{-}cdf\ F\ u\ (\beta\ u * x))$ (*GPD-cdf* γ *0 1*) (*endpoint-filter F*)
by (*intro excess-pointwise-imp-uniform*[*OF F* βpos] *pconv*)
thus *?thesis*
using *excess-uniform-scaled-iff*[*OF* βpos] **by** (*simp add*: β-*def*)
qed

corollary *Pickands-Balkema-de-Haan-normalization'*:
fixes $F :: real \Rightarrow real$ **and** $\gamma :: real$
assumes F: *right-continuous-mono F 0 1*
and *dom*: *domain-of-attraction F* (*EVD-cdf* γ *0 1*)
obtains $a :: real \Rightarrow real$
where $\bigwedge t.\ 0 < a\ t$
and $\bigwedge x.\ 0 < x \Longrightarrow ((\lambda t.\ (tail\text{-}quantile\ F\ (t * x) - tail\text{-}quantile\ F\ t) / a\ t) \longrightarrow (if\ \gamma = 0\ then\ ln\ x\ else\ (x\ powr\ \gamma - 1) / \gamma))\ at\text{-}top$
and $\bigwedge x.\ (\lambda n.\ F\ (a\ (real\ n) * x + tail\text{-}quantile\ F\ (real\ n))\ \hat{}\ n) \longrightarrow EVD\text{-}cdf\ \gamma\ 0\ 1\ x$
and *uniform-limit* $\{0..\}$ $(\lambda u\ x.\ excess\text{-}cdf\ F\ u\ x - GPD\text{-}cdf\ \gamma\ 0\ (a\ (1 / (1 - F\ u)))\ x)$ $(\lambda x.\ 0)$ (*endpoint-filter F*)
proof −
obtain a **where** *apos*: $\bigwedge t.\ 0 < a\ t$
and *quant0*: $\bigwedge x.\ 0 < x \Longrightarrow ((\lambda t.\ (tail\text{-}quantile\ F\ (t * x) - tail\text{-}quantile\ F\ t) / a\ t) \longrightarrow (if\ \gamma = 0\ then\ ln\ x\ else\ (x\ powr\ \gamma - 1) / \gamma))\ at\text{-}top$
using *dom MDA-iff-tail-quantile'*[*OF F*, *of* γ] **by** *auto*
have *quant*: $\bigwedge x.\ 0 < x \Longrightarrow ((\lambda t.\ (tail\text{-}quantile\ F\ (t * x) - tail\text{-}quantile\ F\ t) / a\ t) \longrightarrow (if\ \gamma = 0\ then\ ln\ x\ else\ (x\ powr\ \gamma - 1) / \gamma))\ at\text{-}top$
using *quant0* **by** *simp*
show *thesis*
by (*rule that*[*of a*])
(*use apos quant tail-quantile-witness-of-gap*[*OF F apos quant*]
Pickands-Balkema-de-Haan-normalization[*OF F apos quant*] **in** *blast*)+
qed

end

5 Concrete Examples

theory *Examples*
imports

Peaks-Over-Threshold
begin

5.1 Maximum of Random Variables

In this subsection, we state results using i.i.d. random variables.

context *prob-space*
begin

lemma *sample-max-measurable*[*measurable*]:
  fixes $X :: nat \Rightarrow 'a \Rightarrow real$
  assumes *indep-vars* (λ-::*nat*. *borel*) *X UNIV*
  shows (λw. *MAX* i∈{..<*n*::*nat*}. *X i w*) ∈ *borel-measurable M*
  using *assms* **by** (*auto simp*: *indep-vars-def intro*!: *borel-measurable-Max*)

lemma *prob-sample-max-le*:
  fixes $X :: nat \Rightarrow 'a \Rightarrow real$
  assumes *indep*: *indep-vars* (λ-. *borel*) *X UNIV*
    and *cdf*: $\bigwedge i$. *cdf* (*distr M borel* (*X i*)) = *F*
    and *n*: *0* < *n*
  shows *prob* {*w* ∈ *space M*. (*MAX* i∈{..<*n*::*nat*}. *X i w*) ≤ *x*} = *F x* ^ *n*
proof −
  have *prob* {*w* ∈ *space M*. (*MAX* i∈{..<*n*::*nat*}. *X i w*) ≤ *x*}
      = *prob* {*w* ∈ *space M*. ∀ i∈{..<*n*}. *X i w* ≤ *x*}
    using *n* **by**(*subst linorder-class.Max-le-iff*) *auto*
  also have ... = *prob* ($\bigcap i$∈{..<*n*::*nat*}. *X i* −‘ {..*x*} ∩ *space M*)
    using *n* **by**(*intro arg-cong*[**where** *f*=*prob*]) *auto*
  also have ... = ($\prod i$∈{..<*n*::*nat*}. *prob* (*X i* −‘ {..*x*} ∩ *space M*))
    using *n* **by**(*intro indep-varsD*[*OF indep*]) *auto*
  also have ... = ($\prod i$∈{..<*n*::*nat*}. *cdf* (*distr M borel* (*X i*)) *x*)
    using *indep* **by**(*simp add*: *cdf-def measure-distr indep-vars-def*)
  also have ... = *F x* ^ *n*
    by(*simp add*: *cdf*)
  finally show *?thesis* **.**
qed

corollary *cdf-distr-sample-max*:
  fixes $X :: nat \Rightarrow 'a \Rightarrow real$
  assumes *indep*: *indep-vars* (λ-. *borel*) *X UNIV*
    and *cdf*: $\bigwedge i$. *cdf* (*distr M borel* (*X i*)) = *F*
    and *n*: *0* < *n*
  shows *cdf* (*distr M borel* (λw. (*MAX* i∈{..<*n*::*nat*}. *X i w*))) *x* = *F x* ^ *n*
proof −
  have *cdf* (*distr M borel* (λw. (*MAX* i∈{..<*n*::*nat*}. *X i w*))) *x*
      = *prob* {*w* ∈ *space M*. (*MAX* i∈{..<*n*::*nat*}. *X i w*) ≤ *x*}
   using *indep* **by**(*auto simp*: *cdf-def measure-distr intro*!: *arg-cong*[**where** *f*=*prob*])
  thus *?thesis*
    by(*simp add*: *prob-sample-max-le assms*)
qed

corollary *cdf-distr-sample-max-normalized*:
fixes $X :: nat \Rightarrow 'a \Rightarrow real$ **and** $a\ b :: nat \Rightarrow real$
assumes *indep*: *indep-vars* (λ-. *borel*) X *UNIV*
and *cdf*: $\bigwedge i.\ cdf\ (distr\ M\ borel\ (X\ i)) = F$
and a: $\bigwedge n.\ 0 < a\ n$
and n: $0 < n$
shows *cdf* (*distr M borel* ($\lambda w.$ ((*MAX* i∈{..<n::*nat*}. $X\ i\ w$) − $b\ n$) / $a\ n$)) x
$= F\ (a\ n * x + b\ n)\ \widehat{}\ n$
(**is** *?lhs* = -)
proof −
have *?lhs* = *cdf* (*distr M borel* ($\lambda w.$ (*MAX* i∈{..<n::*nat*}. $X\ i\ w$))) ($a\ n * x + b\ n$)
using *indep a*
by(*auto simp*: *cdf-def measure-distr pos-divide-le-eq*
mult.commute intro!: *arg-cong*[**where** f=*prob*])
thus *?thesis*
by(*simp add*: *assms cdf-distr-sample-max*)
qed

theorem *Fisher-Tippett-Gnedenko-sample-max*:
fixes $X :: nat \Rightarrow 'a \Rightarrow real$ **and** $a\ b :: nat \Rightarrow real$
assumes *indep*: *indep-vars* (λ-. *borel*) X *UNIV*
and *cdf*: $\bigwedge i.\ cdf\ (distr\ M\ borel\ (X\ i)) = F$
and N: *real-distribution* N **and** *deg*: ¬ *degenerate-df* (*cdf* N)
and a: $\bigwedge n.\ 0 < a\ n$
and *wcm*: *weak-conv-m* ($\lambda n.$ *distr M borel* ($\lambda w.$ ((*MAX* i∈{..<n::*nat*}. $X\ i\ w$) − $b\ n$) / $a\ n$)) N
obtains $\gamma\ \mu\ \sigma$ **where** $0 < \sigma$ **and** N = *density lborel* (*EVD-density* $\gamma\ \mu\ \sigma$)
proof −
interpret N: *real-distribution* N
by *fact*
have *wc*:*weak-conv* ($\lambda n\ x.\ F\ (a\ n * x + b\ n)\ \widehat{}\ n$) (*cdf* N)
unfolding *weak-conv-def*
proof *safe*
fix x
assume x:*isCont* (*cdf* N) x
show ($\lambda n.\ F\ (a\ n * x + b\ n)\ \widehat{}\ n$) ⟶ *cdf* $N\ x$
by(*rule LIMSEQ-imp-Suc*,
subst cdf-distr-sample-max-normalized[**where** a=a,*OF indep cdf a*,*symmetric*])
(*use wcm x* **in** *auto simp*: *weak-conv-m-def weak-conv-def dest*: *LIMSEQ-Suc*)
qed
have *right-continuous-mono F 0 1*
unfolding *cdf*[*of 0*,*symmetric*] **using** *indep*
by(*auto intro*!: *right-continuous-mono-cdf simp*: *indep-vars-def*)
from *Fisher-Tippett-Gnedenko*[*OF this N*.*right-continuous-mono-axioms deg a wc*]
obtain $\gamma\ \mu\ \sigma$ **where** $0 < \sigma$ *cdf* N = *EVD-cdf* $\gamma\ \mu\ \sigma$
by *metis*

thus *?thesis*
by(*auto intro*!: *cdf-unique that N real-distribution-EVD*)
qed

end

5.2 Excesses of a Random Variable over a High Threshold

context *prob-space*
begin

lemma *prob-gt-pos-of-lt-x-star*:
fixes $X :: 'a \Rightarrow real$
assumes X[*measurable*]: *random-variable borel X*
and *cdf*: *cdf* (*distr M borel X*) = F
and u: $u < x\text{-}star\ F$
shows $0 < prob\ \{w \in space\ M.\ u < X\ w\}$
proof −
have F: *right-continuous-mono F 0 1*
unfolding *cdf*[*symmetric*]
by(*auto intro*!: *right-continuous-mono-cdf*)
have $F\ u < 1$
using *F-lt-1-unbdd-support*[*OF F*] *lt-x-star-iff-F-lt-1*[*OF F*, *THEN iffD1*, *OF - u*]
by(*cases bdd-support F*) *auto*
moreover have $prob\ \{w \in space\ M.\ u < X\ w\} = 1 - F\ u$
unfolding *cdf*[*symmetric*]
by(*auto simp*: *cdf-def measure-distr prob-compl*[*symmetric*] *intro*!: *arg-cong*[**where** *f*=*prob*])
ultimately show *?thesis*
by *simp*
qed

lemma *AE-le-x-star*:
fixes $X :: 'a \Rightarrow real$
assumes X[*measurable*]: *random-variable borel X*
and *cdf*: *cdf* (*distr M borel X*) = F
and *bdd*: *bdd-support F*
shows $AE\ w\ in\ M.\ X\ w \leq x\text{-}star\ F$
proof −
have F: *right-continuous-mono F 0 1*
unfolding *cdf*[*symmetric*]
by(*auto intro*!: *right-continuous-mono-cdf*)
have $prob\ \{w \in space\ M.\ X\ w \leq x\text{-}star\ F\} = F\ (x\text{-}star\ F)$
by(*auto simp*: *cdf*[*symmetric*] *cdf-def measure-distr intro*!: *arg-cong*[**where** *f*=*prob*])
also have ... = *1*
by(*rule F-x-star-eq-1*[*OF F bdd*])
finally show *?thesis*

by(*simp add*: *prob-Collect-eq-1*)
qed

lemma *cond-prob-excess-cdf*:
fixes $X :: 'a \Rightarrow real$
assumes X[*measurable*]: *random-variable borel X*
and *cdf*: *cdf* (*distr M borel X*) = F
and u: $F\ u < 1$ **and** x:$0 \leq x$
shows $\mathcal{P}(w\ in\ M.\ X\ w - u \leq x \mid u < X\ w) = excess\text{-}cdf\ F\ u\ x$
proof −
have $\mathcal{P}(w\ in\ M.\ X\ w - u \leq x \mid u < X\ w) =$
prob $\{\omega \in space\ M.\ X\ \omega - u \leq x \wedge u < X\ \omega\}$ / *prob* $\{\omega \in space\ M.\ u < X\ \omega\}$
by(*simp add*: *cond-prob-def*)
also have ... = *prob* $(\{\omega \in space\ M.\ X\ \omega \leq x + u\} - \{\omega \in space\ M.\ X\ \omega \leq u\})$
/ *prob* $\{\omega \in space\ M.\ u < X\ \omega\}$
by(*auto intro*!: *arg-cong*[**where** *f*=*prob*])
also have ... = (*prob* $\{\omega \in space\ M.\ X\ \omega \leq x + u\}$
− *prob* $(\{\omega \in space\ M.\ X\ \omega \leq x + u\} \cap \{\omega \in space\ M.\ X\ \omega \leq u\})$)
/ *prob* $\{\omega \in space\ M.\ u < X\ \omega\}$
by(*simp add*: *finite-measure-Diff'*)
also have ... = (*prob* $\{\omega \in space\ M.\ X\ \omega \leq x + u\}$ − *prob* $\{\omega \in space\ M.\ X\ \omega \leq u\}$)
/ *prob* $\{\omega \in space\ M.\ u < X\ \omega\}$
using x **by**(*auto intro*!: *arg-cong*[**where** *f*=*prob*])
also have ... = $(F\ (x + u) - F\ u)$ / *prob* $\{\omega \in space\ M.\ u < X\ \omega\}$
by(*auto simp*: *cdf*[*symmetric*] *cdf-def measure-distr*
intro!: *arg-cong2*[*OF arg-cong*[**where** *f*=*prob*] *arg-cong*[**where** *f*=*prob*]])
also have ... = $(F\ (x + u) - F\ u)$ / $(1 - F\ u)$
by(*auto simp*: *cdf*[*symmetric*] *cdf-def measure-distr prob-compl*[*symmetric*]
intro!: *arg-cong*[**where** *f*=*prob*])
also have ... = *excess-cdf F u x*
using u **by**(*simp add*: *excess-cdf-def2 add.commute*)
finally show *?thesis* **.**
qed

lemma *cdf-distr-excess*:
fixes $X :: 'a \Rightarrow real$
assumes X[*measurable*]: *random-variable borel X*
and *cdf*: *cdf* (*distr M borel X*) = F
and u: $F\ u < 1$ **and** x:$0 \leq x$
shows *cdf* (*distr* (*uniform-measure M* $\{w \in space\ M.\ u < X\ w\}$) *borel* $(\lambda w.\ X\ w - u))\ x$
= *excess-cdf F u x* (**is** *?lhs* = -)
proof −
have *?lhs* = $\mathcal{P}(\omega\ in\ uniform\text{-}measure\ M\ \{w \in space\ M.\ u < X\ w\}.\ X\ \omega - u \leq x)$
by(*auto simp*: *cdf-def measure-distr intro*!: *arg-cong*[**where** *f*=*measure* -])

```
  also have ... = 𝒫(w in M. X w − u ≤ x | u < X w)
    by(subst measure-uniform-measure-eq-cond-prob) auto
  also have ... = excess-cdf F u x
    by(simp add: cond-prob-excess-cdf assms)
  finally show ?thesis .
qed

theorem Pickands-Balkema-de-Haan-cond-prob:
  fixes X :: 'a ⇒ real and F :: real ⇒ real and γ :: real
  assumes X[measurable]: random-variable borel X
    and cdf: cdf (distr M borel X) = F
    and dom: domain-of-attraction F (EVD-cdf γ 0 1)
  obtains β :: real ⇒ real
  where ⋀u. 0 < β u
    and uniform-limit {0..}
          (λu x. 𝒫(w in M. X w − u ≤ x | u < X w) − GPD-cdf γ 0 (β u) x)
          (λx. 0) (endpoint-filter F)
proof −
  have F:right-continuous-mono F 0 1
    unfolding cdf[symmetric]
    by(auto intro!: right-continuous-mono-cdf)
  from Pickands-Balkema-de-Haan[OF this] dom
  obtain β where h:⋀u. 0 < β u
     uniform-limit {0..} (λu x. excess-cdf F u x − GPD-cdf γ 0 (β u) x) (λx. 0)
                   (endpoint-filter F)
    by blast
  have ∀ F u in endpoint-filter F. ∀ x∈{0..}.
        𝒫(w in M. X w − u ≤ x | u < X w) − GPD-cdf γ 0 (β u) x
        = excess-cdf F u x − GPD-cdf γ 0 (β u) x
   by(auto intro!: eventually-mono[OF F-lt-1-endpoint[OF F]] simp: cond-prob-excess-cdf
assms)
  from h uniform-limit-cong[OF this] that
  show ?thesis
    by blast
qed

end

lemma excess-cdf-GPD:
  fixes γ σ u x :: real
  assumes sigma: 0 < σ and u: 0 ≤ u and supp: 0 < σ + γ ∗ u and x: 0 ≤ x
  shows excess-cdf (GPD-cdf γ 0 σ) u x = GPD-cdf γ 0 (σ + γ ∗ u) x
proof (cases γ = 0)
  case True
  have excess-cdf (GPD-cdf γ 0 σ) u x = 1 − exp (− ((u + x) / σ)) / exp (− (u
/ σ))
    using True u x by (simp add: excess-cdf-def GPD-cdf-def[OF sigma])
  also have ... = 1 − exp (− (x / σ))
    using sigma by (simp add: exp-diff[symmetric] field-simps)
```

```
  also have ... = GPD-cdf γ 0 (σ + γ * u) x
    using True x by (simp add: GPD-cdf-def[OF sigma])
  finally show ?thesis .
next
  case γ0: False
  define s where s = σ + γ * u
  have s: 0 < s
    using supp by (simp add: s-def)
  have ru: 1 + γ * (u / σ) = s / σ
    using sigma by (simp add: s-def field-simps)
  have rx: 1 + γ * ((u + x) / σ) = (s + γ * x) / σ
    using sigma by (simp add: s-def field-simps)
  have pu: 0 < s / σ
    using s sigma by simp
  have cond-u: − 1 ≤ γ * (u / σ)
    using ru pu by linarith
  have tail-u: 1 − GPD-cdf γ 0 σ u = (s / σ) powr (− 1 / γ)
    using u γ0 cond-u ru by (simp add: GPD-cdf-def[OF sigma] Let-def)
  show ?thesis
    unfolding s-def[symmetric]
  proof (cases 0 ≤ s + γ * x)
    case True
    have nn: 0 ≤ (s + γ * x) / σ
      using True sigma by simp
    have cond-x: − 1 ≤ γ * ((u + x) / σ)
      using rx nn by linarith
    have tail-x: 1 − GPD-cdf γ 0 σ (u + x) = ((s + γ * x) / σ) powr (− 1 / γ)
      using u x γ0 cond-x rx by (simp add: GPD-cdf-def[OF sigma] Let-def)
    have base: (s + γ * x) / σ / (s / σ) = 1 + γ * (x / s)
      using s sigma by (simp add: field-simps)
    have div: ((s + γ * x) / σ) powr (− 1 / γ) / (s / σ) powr (− 1 / γ)
                = (1 + γ * (x / s)) powr (− 1 / γ)
    proof −
      have ((s + γ * x) / σ) powr (− 1 / γ) / (s / σ) powr (− 1 / γ)
            = ((s + γ * x) / σ / (s / σ)) powr (− 1 / γ)
        using nn pu by (intro powr-divide[symmetric])
      also have ... = (1 + γ * (x / s)) powr (− 1 / γ)
        unfolding base ..
      finally show ?thesis .
    qed
    have excess-cdf (GPD-cdf γ 0 σ) u x = 1 − (1 + γ * (x / s)) powr (− 1 / γ)
      using tail-u tail-x div by (simp add: excess-cdf-def)
    also have ... = GPD-cdf γ 0 s x
      using x γ0 True s by (simp add: GPD-cdf-def[OF s] Let-def field-simps)
    finally show excess-cdf (GPD-cdf γ 0 σ) u x = GPD-cdf γ 0 s x .
  next
    case False
    have neg: (s + γ * x) / σ < 0
      using False sigma by (simp add: divide-neg-pos)
```

```
    have cond-x: ¬ - 1 ≤ γ * ((u + x) / σ)
      using rx neg by linarith
    have GPD-cdf γ 0 σ (u + x) = 1
      using u x γ0 cond-x by (simp add: GPD-cdf-def[OF sigma] Let-def)
    hence excess-cdf (GPD-cdf γ 0 σ) u x = 1
      by (simp add: excess-cdf-def)
    also have ... = GPD-cdf γ 0 s x
      using x γ0 False s by (simp add: GPD-cdf-def[OF s] Let-def field-simps)
    finally show excess-cdf (GPD-cdf γ 0 σ) u x = GPD-cdf γ 0 s x .
  qed
qed

corollary excess-cdf-exponential:
  fixes u x :: real
  assumes 0 ≤ u and 0 ≤ x
  shows excess-cdf (GPD-cdf 0 0 1) u x = GPD-cdf 0 0 1 x
  using excess-cdf-GPD[where γ = 0 and σ = 1 and u = u and x = x] assms
by simp

corollary excess-cdf-Pareto1:
  fixes u x :: real
  assumes u: 1 ≤ u and x: 0 ≤ x
  shows excess-cdf (GPD-cdf 1 1 1) u x = GPD-cdf 1 0 u x
proof -
  have shift: GPD-cdf 1 1 1 y = GPD-cdf 1 0 1 (y - 1) for y
    by (simp add: GPD-cdf-def[OF zero-less-one] Let-def)
  have excess-cdf (GPD-cdf 1 1 1) u x = excess-cdf (GPD-cdf 1 0 1) (u - 1) x
    by (simp add: excess-cdf-def shift algebra-simps)
  also have ... = GPD-cdf 1 0 u x
    using excess-cdf-GPD[where γ = 1 and σ = 1 and u = u - 1 and x = x]
u x by simp
  finally show ?thesis .
qed

corollary excess-cdf-uniform:
  fixes u x :: real
  assumes 0 ≤ u and u < 1 and 0 ≤ x
  shows excess-cdf (GPD-cdf (- 1) 0 1) u x = GPD-cdf (- 1) 0 (1 - u) x
  using excess-cdf-GPD[where γ = - 1 and σ = 1 and u = u and x = x] assms
by simp

lemma excess-cdf-EVD-tendsto-GPD:
  fixes γ :: real
  obtains β :: real ⇒ real
  where ⋀u. 0 < β u
    and uniform-limit {0..}
          (λu x. excess-cdf (EVD-cdf γ 0 1) u x - GPD-cdf γ 0 (β u) x)
          (λx. 0) (endpoint-filter (EVD-cdf γ 0 1))
  using Pickands-Balkema-de-Haan[OF right-continuous-mono-EVD-cdf[OF zero-less-one],
```

```
of γ]
      EVD-cdf-MDA[OF zero-less-one, of γ 0]
  by blast

corollary(in prob-space) cond-prob-excess-GPD:
  fixes X :: 'a ⇒ real and γ σ u x :: real
  assumes X: random-variable borel X
    and X-cdf: cdf (distr M borel X) = GPD-cdf γ 0 σ
    and sigma: 0 < σ and u: 0 ≤ u and supp: 0 < σ + γ * u and x: 0 ≤ x
  shows P(w in M. X w − u ≤ x | u < X w) = GPD-cdf γ 0 (σ + γ * u) x
proof −
  have lt1: GPD-cdf γ 0 σ u < 1
  proof (cases γ = 0)
    case True
    thus ?thesis
      using u by (simp add: GPD-cdf-def[OF sigma])
  next
    case False
    have 0 < 1 + γ * (u / σ)
      using supp sigma by (simp add: field-simps)
    thus ?thesis
      using u False by (simp add: GPD-cdf-def[OF sigma] Let-def)
  qed
  have P(w in M. X w − u ≤ x | u < X w) = excess-cdf (GPD-cdf γ 0 σ) u x
    by (rule cond-prob-excess-cdf[OF X X-cdf lt1 x])
  also have ... = GPD-cdf γ 0 (σ + γ * u) x
    by (rule excess-cdf-GPD[OF sigma u supp x])
  finally show ?thesis .
qed
```

5.3 Domain of Attraction

In this subsection we formalize concrete distributions, and determine the extreme value index γ of the extreme value distribution they are attracted to.

The results cover the following distributions, where $\mathcal{D}(G_\gamma)$ denotes (*in-domain-attraction-of*) for *EVD-cdf γ 0 1*.

- Normal distribution: $F_{\mathrm{N}} \in \mathcal{D}(G_0)$.
- Erlang distribution: $F_{\mathrm{E}} \in \mathcal{D}(G_0)$.
- Uniform distribution on an interval: $F_{\mathrm{U}} \in \mathcal{D}(G_{-1})$.
- Cauchy distribution: $F_{\mathrm{C}} \in \mathcal{D}(G_1)$.

5.3.1 Normal Distribution

lemma *real-distribution-normal*:
  $0 < \sigma \Longrightarrow$ *real-distribution* (*density lborel* (*normal-density* μ σ))
 by(*simp add*: *real-distribution-def real-distribution-axioms-def prob-space-normal-density*)

lemma *normal-density-affine-eq*:
  assumes *sigma*: $0 < \sigma$
  shows *normal-density* μ σ ($\mu + \sigma * y$) = *std-normal-density y* / σ
  using *sigma*
  by(*simp add*: *normal-density-def real-sqrt-mult field-simps*)

lemma *std-normal-tail-measure*:
  1 − *cdf std-normal-distribution t*
    = *enn2real* ($\int^+ y \in \{t..\}$. *ennreal* (*std-normal-density y*) ∂*lborel*)
proof −
  interpret *N*: *real-distribution std-normal-distribution*
    by(*rule real-distribution-normal*) *simp*
 have 1 − *cdf std-normal-distribution t* = *measure std-normal-distribution* (*UNIV* − {*..t*})
    using *N.prob-compl*[*of* {*..t*}] **by**(*simp add*: *cdf-def*)
  also have ... = *measure std-normal-distribution* {*t<..*}
    by(*rule arg-cong*[**where** *f*=*measure std-normal-distribution*]) *auto*
  also have ... = *enn2real* ($\int^+ y \in \{t<..\}$. *ennreal* (*std-normal-density y*) ∂*lborel*)
    by(*simp add*: *measure-def emeasure-density*)
  also have ... = *enn2real* ($\int^+ y \in \{t..\}$. *ennreal* (*std-normal-density y*) ∂*lborel*)
   by(*intro arg-cong*[**where** *f*=*enn2real*] *nn-integral-cong-AE AE-I*[**where** *N*={*t*}])
      (*auto simp*: *indicator-def*)
  finally show *?thesis* **.**
qed

lemma *DERIV-std-normal-density*:
  DERIV std-normal-density y :> − *y* ∗ *std-normal-density y*
  unfolding *std-normal-density-def*
  by(*auto intro*!: *derivative-eq-intros simp*: *field-simps*)

lemma *DERIV-mills-upper*:
  fixes *y* :: *real*
  assumes *y*: $0 < y$
  shows *DERIV* (λz. − (*std-normal-density z* / *z*)) *y* :> *std-normal-density y* ∗ (1 + 1 / y^2)
  using *y*
  by(*auto intro*!: *derivative-eq-intros DERIV-std-normal-density*
      simp: *field-simps power2-eq-square*)

lemma *DERIV-mills-lower*:
  fixes *y* :: *real*
  assumes *y*: $0 < y$
  shows *DERIV* (λz. *std-normal-density z* / *z* ^ 3 − *std-normal-density z* / *z*) *y*
        :> *std-normal-density y* ∗ (1 − 3 / *y* ^ 4)

```
  using y
  by(auto intro!: derivative-eq-intros DERIV-std-normal-density
      simp: field-simps eval-nat-numeral)

lemma tendsto-mills-upper: ((λz::real. − (std-normal-density z / z)) ⟶ 0)
at-top
  unfolding std-normal-density-def by real-asymp

lemma tendsto-mills-lower:
  ((λz. std-normal-density z / z ^ 3 − std-normal-density z / z) ⟶ 0) at-top
  unfolding std-normal-density-def by real-asymp

lemma std-normal-nn-integral-upper:
  fixes t :: real
  assumes t: 0 < t
  shows (∫⁺y∈{t..}. ennreal (std-normal-density y * (1 + 1 / y²)) ∂lborel)
        = ennreal (std-normal-density t / t)
proof −
  have (∫⁺y∈{t..}. ennreal (std-normal-density y * (1 + 1 / y²)) ∂lborel)
        = ennreal (0 − (− (std-normal-density t / t)))
    by(rule nn-integral-FTC-atLeast[where F=λz. − (std-normal-density z / z)])
      (use t in ‹auto intro!: DERIV-mills-upper tendsto-mills-upper›)
  thus ?thesis
    by simp
qed

lemma std-normal-nn-integral-lower:
  fixes t :: real
  assumes t: 2 ≤ t
  shows (∫⁺y∈{t..}. ennreal (std-normal-density y * (1 − 3 / y ^ 4)) ∂lborel)
        = ennreal (std-normal-density t / t − std-normal-density t / t ^ 3)
proof −
  have nonneg: 0 ≤ std-normal-density x * (1 − 3 / x ^ 4) if x: t ≤ x for x
  proof −
    have x2: 2 ≤ x
      using t x by simp
    hence (2::real) ^ 4 ≤ x ^ 4
      by(intro power-mono) auto
    hence 3 / x ^ 4 ≤ 3 / 2 ^ 4
      using x2 by(intro divide-left-mono) auto
    thus ?thesis
      by(intro mult-nonneg-nonneg normal-density-nonneg) simp
  qed
  have (∫⁺y∈{t..}. ennreal (std-normal-density y * (1 − 3 / y ^ 4)) ∂lborel)
        = ennreal (0 − (std-normal-density t / t ^ 3 − std-normal-density t / t))
    by(rule nn-integral-FTC-atLeast
        [where F=λz. std-normal-density z / z ^ 3 − std-normal-density z / z])
      (use t nonneg in ‹auto intro!: DERIV-mills-lower tendsto-mills-lower›)
  thus ?thesis
```

```
    by simp
qed

lemma std-normal-tail-bounds:
  fixes t :: real
  assumes t: 2 ≤ t
 shows std-normal-tail-upper: 1 − cdf std-normal-distribution t ≤ std-normal-density
t / t
    and std-normal-tail-lower:
      std-normal-density t / t − std-normal-density t / t ^ 3
        ≤ 1 − cdf std-normal-distribution t
proof −
  have t0: 0 < t
    using t by simp
  have le: std-normal-density y ≤ std-normal-density y ∗ (1 + 1 / y²) for y :: real
    using mult-left-mono[of 1 1 + 1 / y² std-normal-density y] by simp
  have up: (∫⁺y∈{t..}. ennreal (std-normal-density y) ∂lborel)
            ≤ ennreal (std-normal-density t / t)
    unfolding std-normal-nn-integral-upper[OF t0, symmetric]
    by(intro nn-integral-mono mult-right-mono ennreal-leI le) simp
  have fin: (∫⁺y∈{t..}. ennreal (std-normal-density y) ∂lborel) < ⊤
    using up ennreal-less-top by(rule le-less-trans)
  show 1 − cdf std-normal-distribution t ≤ std-normal-density t / t
    unfolding std-normal-tail-measure
    using enn2real-mono[OF up] t0 by simp
  have cube: std-normal-density t / t ^ 3 ≤ std-normal-density t / t
    using t0 t power-increasing[of 1 3 t]
    by(intro divide-left-mono) auto
  have low: ennreal (std-normal-density t / t − std-normal-density t / t ^ 3)
             ≤ (∫⁺y∈{t..}. ennreal (std-normal-density y) ∂lborel)
    unfolding std-normal-nn-integral-lower[OF t, symmetric]
    by(intro nn-integral-mono mult-right-mono ennreal-leI mult-left-le) auto
  show std-normal-density t / t − std-normal-density t / t ^ 3
        ≤ 1 − cdf std-normal-distribution t
    unfolding std-normal-tail-measure
    using enn2real-mono[OF low fin] cube t0 by simp
qed

lemma std-normal-tail-asymp-equiv:
  (λt. 1 − cdf std-normal-distribution t) ∼[at-top] (λt. std-normal-density t / t)
proof(rule asymp-equiv-symI[OF asymp-equiv-sandwich-real'
       [where l=λt. std-normal-density t / t − std-normal-density t / t ^ 3
          and u=λt. std-normal-density t / t]])
  show (λt. std-normal-density t / t)
        ∼[at-top] (λt. std-normal-density t / t − std-normal-density t / t ^ 3)
    unfolding std-normal-density-def
    by(rule asymp-equivI') real-asymp
next
  show ∀F t in at-top. 1 − cdf std-normal-distribution t
```

```
        ∈ {std-normal-density t / t − std-normal-density t / t ^ 3..
           std-normal-density t / t}
    by(intro eventually-mono[OF eventually-ge-at-top[of 2::real]])
      (auto simp: std-normal-tail-upper std-normal-tail-lower)
qed simp

lemma std-normal-density-at-normalization:
  fixes x t :: real
  assumes t: exp 1 < t
  defines c ≡ sqrt (2 * ln t)
    and w ≡ x − (ln (ln t) + ln (4 * pi)) / 2
  shows std-normal-density (c + w / c)
        = exp (− x) * c * exp (− (w^2 / (4 * ln t))) / t
proof −
  have tpos: 0 < t
    using t exp-gt-zero[of 1] by linarith
  have lnt: 1 < ln t
    using t tpos by(simp add: ln-less-cancel-iff[of exp 1 t, symmetric])
  have cpos: 0 < c
    using lnt by(simp add: c-def)
  have c2: c^2 = 2 * ln t
    using lnt by(simp add: c-def)
  have usq: (c + w / c)^2 / 2 = ln t + w + w^2 / (4 * ln t)
  proof −
    have (c + w / c)^2 = c^2 + 2 * w + w^2 / c^2
      using cpos by(simp add: power2-eq-square field-simps)
    thus ?thesis
      unfolding c2 using lnt by(simp add: field-simps)
  qed
  have expd: exp ((ln (ln t) + ln (4 * pi)) / 2) = sqrt (4 * pi * ln t)
    using lnt
   by(intro real-sqrt-unique[symmetric]) (auto simp: power2-eq-square mult-exp-exp
exp-add)
  have sq: sqrt (4 * pi * ln t) = sqrt (2 * pi) * c
    using lnt by(simp add: c-def real-sqrt-mult[symmetric])
  have expw: exp (− w) = exp (− x) * sqrt (4 * pi * ln t)
    unfolding w-def expd[symmetric] by(simp flip: exp-add)
  have std-normal-density (c + w / c) = 1 / sqrt (2 * pi) * exp (− ((c + w / c)^2
/ 2))
    by(simp add: std-normal-density-def)
  also have ... = 1 / sqrt (2 * pi) * exp (− (ln t + w + w^2 / (4 * ln t)))
    unfolding usq ..
  also have ... = 1 / sqrt (2 * pi) * (exp (− ln t) * exp (− w) * exp (− (w^2 / (4
* ln t))))
    by(simp flip: exp-add)
  also have ... = exp (− x) * c * exp (− (w^2 / (4 * ln t))) / t
    unfolding expw sq
    using tpos by(simp add: exp-minus field-simps)
  finally show ?thesis .
```

qed

lemma *normal-cdf-location-scale*:
assumes *sigma*[*arith*]: $0 < \sigma$
shows *cdf* (*density lborel* (*normal-density* μ σ)) x = *cdf std-normal-distribution* (($x - \mu$) / σ)
proof −
have *ind*: *indicator* {..x} ($\mu + \sigma * y$) = (*indicator* {..($x - \mu$) / σ} y :: *ennreal*) **for** y
by(*simp add*: *indicator-def field-simps*)
have ($\int^+ y \in \{..x\}$. *ennreal* (*normal-density* μ σ y) ∂*lborel*)
= *ennreal* |σ| * ($\int^+ y$. *ennreal* (*normal-density* μ σ ($\mu + \sigma * y$)) * *indicator* {..x} ($\mu + \sigma * y$) ∂*lborel*)
by(*subst nn-integral-real-affine* [**where** t=μ **and** c=σ **and** f=λy. *ennreal* (*normal-density* μ σ y) * *indicator* {..x} y]) *auto*
also have ... = ($\int^+ y$. *ennreal* σ * (*ennreal* (*std-normal-density* y / σ) * *indicator* {..($x - \mu$) / σ} y) ∂*lborel*)
by(*subst nn-integral-cmult*) (*auto simp*: *normal-density-affine-eq ind*)
also have ... = ($\int^+ y \in \{..(x - \mu) / \sigma\}$. *ennreal* (*std-normal-density* y) ∂*lborel*)
by(*intro nn-integral-cong*) (*simp add*: *ennreal-mult'*[*symmetric*] *mult.assoc*[*symmetric*])
finally show *?thesis*
by(*simp add*: *cdf-def measure-def emeasure-density*)
qed

theorem *std-normal-MDA*: *domain-of-attraction* (*cdf std-normal-distribution*) (*EVD-cdf 0 0 1*)
proof −
have *F*: *right-continuous-mono* (*cdf std-normal-distribution*) *0 1*
by(*auto intro*!: *right-continuous-mono-cdf real-distribution-normal*)
have *pos*: *0* < *std-normal-density t* / *t* − *std-normal-density t* / *t* ^ *3* **if** *t*: *2* ≤ *t* **for** *t*
proof −
have *std-normal-density t* / *t* ^ *3* < *std-normal-density t* / *t*
using *t normal-density-pos*[**where** σ=*1* **and** μ=*0* **and** x=*t*] *power-strict-increasing*[*of 1 3 t*]
by(*intro divide-strict-left-mono*) *auto*
thus *?thesis*
by *simp*
qed
have *unbdd*: ¬ *bdd-support* (*cdf std-normal-distribution*)
proof(*rule not-bdd-supportI*)
fix x :: *real*
have *s*: *2* ≤ *max x 2* + *1 x* < *max x 2* + *1*
by *auto*

```
    show ∃ y>x. cdf std-normal-distribution y < 1
      using std-normal-tail-lower[OF s(1)] pos[OF s(1)] s by force
  qed
  have ep: endpoint-filter (cdf std-normal-distribution) = at-top
    using unbdd by(simp add: endpoint-filter-def)
  have lim: ((λt. (1 − cdf std-normal-distribution (t + x ∗ (1 / max t 1)))
                  / (1 − cdf std-normal-distribution t))
             ⟶ − ln (EVD-cdf 0 0 1 x)) at-top for x
  proof −
    have shift: filterlim (λt::real. t + x / t) at-top at-top
      by real-asymp
    have (λt. 1 − cdf std-normal-distribution (t + x / t))
          ∼[at-top] (λt. std-normal-density (t + x / t) / (t + x / t))
      by(intro asymp-equiv-compose'[OF std-normal-tail-asymp-equiv] shift)
    hence ae: (λt. (1 − cdf std-normal-distribution (t + x / t))
                    / (1 − cdf std-normal-distribution t))
               ∼[at-top] (λt. std-normal-density (t + x / t) / (t + x / t)
                              / (std-normal-density t / t))
      by(intro asymp-equiv-divide std-normal-tail-asymp-equiv)
    have lim2: ((λt. std-normal-density (t + x / t) / (t + x / t)
                     / (std-normal-density t / t)) ⟶ exp (− x)) at-top
      unfolding std-normal-density-def exp-minus inverse-eq-divide by real-asymp
    from tendsto-asymp-equiv-cong[OF ae] lim2
    have ((λt. (1 − cdf std-normal-distribution (t + x / t))
               / (1 − cdf std-normal-distribution t)) ⟶ exp (− x)) at-top
      by simp
    moreover have ∀ F t in at-top.
      (1 − cdf std-normal-distribution (t + x / t)) / (1 − cdf std-normal-distribution
t)
        = (1 − cdf std-normal-distribution (t + x ∗ (1 / max t 1)))
          / (1 − cdf std-normal-distribution t)
      by(intro eventually-mono[OF eventually-ge-at-top[of 1::real]]) simp
    ultimately have ((λt. (1 − cdf std-normal-distribution (t + x ∗ (1 / max t
1)))
                          / (1 − cdf std-normal-distribution t)) ⟶ exp (− x))
at-top
      by(rule Lim-transform-eventually)
    thus ?thesis
      by(simp add: EVD-cdf-def)
  qed
  show ?thesis
    unfolding MDA-iff-divide-tail[OF F] ep
    using lim by(auto intro!: exI[where x=λt. 1 / max t 1])
qed

corollary normal-MDA:
  assumes 0 < σ
  shows domain-of-attraction (cdf (density lborel (normal-density μ σ))) (EVD-cdf
0 0 1)
```

unfolding *normal-cdf-location-scale*[*OF assms*]
by(*intro domain-of-attraction-affine assms std-normal-MDA*)

theorem *std-normal-normalization*:
fixes x :: *real* **and** a b :: $nat \Rightarrow real$
defines $a \equiv \lambda n.\ 1\ /\ sqrt\ (2 * ln\ (real\ n))$
and $b \equiv \lambda n.\ sqrt\ (2 * ln\ (real\ n))$
$- (ln\ (ln\ (real\ n)) + ln\ (4 * pi))\ /\ (2 * sqrt\ (2 * ln\ (real\ n)))$
shows $(\lambda n.\ cdf\ std\text{-}normal\text{-}distribution\ (a\ n * x + b\ n)\ \hat{}\ n) \longrightarrow EVD\text{-}cdf\ 0\ 0\ 1\ x$
proof −
interpret N: *real-distribution std-normal-distribution*
by(*rule real-distribution-normal*) *simp*
define c **where** $c\ t = sqrt\ (2 * ln\ t)$ **for** t :: *real*
define w **where** $w\ t = x - (ln\ (ln\ t) + ln\ (4 * pi))\ /\ 2$ **for** t :: *real*
define u **where** $u\ t = c\ t + w\ t\ /\ c\ t$ **for** t :: *real*
have *u-top*: *filterlim u at-top at-top*
unfolding *u-def c-def w-def* **by** *real-asymp*
have $(\lambda t.\ 1 - cdf\ std\text{-}normal\text{-}distribution\ (u\ t))$
$\sim[at\text{-}top]\ (\lambda t.\ std\text{-}normal\text{-}density\ (u\ t)\ /\ u\ t)$
by(*intro asymp-equiv-compose'*[*OF std-normal-tail-asymp-equiv*] *u-top*)
hence *ae*: $(\lambda t.\ t * (1 - cdf\ std\text{-}normal\text{-}distribution\ (u\ t)))$
$\sim[at\text{-}top]\ (\lambda t.\ t * (std\text{-}normal\text{-}density\ (u\ t)\ /\ u\ t))$
by(*intro asymp-equiv-mult asymp-equiv-refl*)
have *ev*: $\forall_F\ t\ in\ at\text{-}top.\ t * (std\text{-}normal\text{-}density\ (u\ t)\ /\ u\ t)$
$= exp\ (-x) * (c\ t\ /\ u\ t) * exp\ (-((w\ t)^2\ /\ (4 * ln\ t)))$
proof(*intro eventually-mono*[*OF eventually-gt-at-top*[*of exp 1*]])
fix t :: *real*
assume *t*: $exp\ 1 < t$
hence *tpos*: $0 < t$
using *exp-gt-zero*[*of 1*] **by** *linarith*
show $t * (std\text{-}normal\text{-}density\ (u\ t)\ /\ u\ t)$
$= exp\ (-x) * (c\ t\ /\ u\ t) * exp\ (-((w\ t)^2\ /\ (4 * ln\ t)))$
unfolding *u-def*
using *std-normal-density-at-normalization*[*OF t*, *of x*] *tpos*
by(*simp add*: *c-def w-def*)
qed
have *lim*: $((\lambda t.\ exp\ (-x) * (c\ t\ /\ u\ t) * exp\ (-((w\ t)^2\ /\ (4 * ln\ t)))) \longrightarrow exp\ (-x))\ at\text{-}top$
unfolding *c-def w-def u-def* **by** *real-asymp*
have $(\lambda t.\ t * (1 - cdf\ std\text{-}normal\text{-}distribution\ (u\ t)))$
$\sim[at\text{-}top]\ (\lambda t.\ exp\ (-x) * (c\ t\ /\ u\ t) * exp\ (-((w\ t)^2\ /\ (4 * ln\ t))))$
by(*rule asymp-equiv-transfer*[*OF ae - ev*]) *simp*
from *tendsto-asymp-equiv-cong*[*OF this*] *lim*
have $((\lambda t.\ t * (1 - cdf\ std\text{-}normal\text{-}distribution\ (u\ t))) \longrightarrow exp\ (-x))\ at\text{-}top$
by *simp*
hence *tail*: $(\lambda n.\ real\ n * (1 - cdf\ std\text{-}normal\text{-}distribution\ (u\ (real\ n)))) \longrightarrow exp\ (-x)$
by(*rule filterlim-compose*[*OF - filterlim-real-sequentially*])

have (λn. *cdf std-normal-distribution* (*u* (*real n*)) ^ *n*) ⟶ *EVD-cdf 0 0 1 x*
⟷ (λn. *real n* ∗ (*1* − *cdf std-normal-distribution* (*u* (*real n*))))
⟶ − *ln* (*EVD-cdf 0 0 1 x*)
by(*intro power-tendsto-iff-tendsto-tail N.cdf-nonneg N.cdf-bounded-prob*)
(*auto simp*: *EVD-cdf-def*)
moreover have *a n* ∗ *x* + *b n* = *u* (*real n*) **for** *n*
by(*simp add*: *a-def b-def u-def c-def w-def diff-divide-distrib*)
ultimately show *?thesis*
using *tail* **by**(*simp add*: *EVD-cdf-def*)
qed

5.3.2 Erlang Distribution

lemma *cdf-erlang-density*:
assumes *0* < *l*
shows *cdf* (*density lborel* (*erlang-density k l*)) = *erlang-CDF k l*
using *assms*
by(*auto simp*: *cdf-def measure-def emeasure-erlang-density*)

corollary *real-distribution-erlang*:
0 < *l* ⟹ *real-distribution* (*density lborel* (*erlang-density k l*))
by(*simp add*: *real-distribution-def real-distribution-axioms-def prob-space-erlang-density*)

lemma *right-continuous-mono-erlang-CDF*:
assumes *l*: *0* < *l*
shows *right-continuous-mono* (*erlang-CDF k l*) *0 1*
using *right-continuous-mono-cdf*[*OF real-distribution-erlang*[*OF l*], *of k*]
by(*simp add*: *cdf-erlang-density*[*OF l*])

lemma *erlang-CDF-lt-1*:
assumes *l*[*arith*]: *0* < *l*
shows *erlang-CDF k l t* < *1*
proof(*cases t* < *0*)
case *t*: *False*
have *0* < (∑ *n*≤*k*. (*l* ∗ *t*) ^ *n* ∗ *exp* (− *l* ∗ *t*) / *fact n*)
using *t* **by**(*intro sum-pos2*[**where** *i*=*0*]) *auto*
thus *?thesis*
using *t* **by**(*simp add*: *erlang-CDF-def*)
qed(*simp add*: *erlang-CDF-def*)

lemma *erlang-density-pos*:
assumes *0* < *l* **and** *0* < *x*
shows *0* < *erlang-density k l x*
using *assms* **by**(*simp add*: *erlang-density-def*)

lemma *DERIV-erlang-sum*:
fixes *x* :: *real*
shows ((λy. − (∑ *n*≤*k*. *y* ^ *n* ∗ *exp* (− *y*) / *fact n*)) *has-real-derivative*
x ^ *k* ∗ *exp* (− *x*) / *fact k*) (*at x*)

```
proof(induction k)
  case 0
  show ?case
    by(auto intro!: derivative-eq-intros)
next
  case (Suc k)
  have ((λy. − (∑ n≤k. y ^ n * exp (− y) / fact n) − y ^ Suc k * exp (− y) /
fact (Suc k))
          has-real-derivative
            x ^ k * exp (− x) / fact k
              − (real (Suc k) − x) * x ^ k * exp (− x) / fact (Suc k)) (at x)
    by(intro DERIV-diff Suc)
      (auto intro!: derivative-eq-intros simp del: fact-Suc power-Suc
        simp add: field-simps power-Suc[symmetric])
  moreover have x ^ k * exp (− x) / fact k
                  − (real (Suc k) − x) * x ^ k * exp (− x) / fact (Suc k)
                = x ^ Suc k * exp (− x) / fact (Suc k)
    by(auto simp: field-simps simp del: fact-Suc) (simp-all add: field-simps)
  ultimately show ?case
    by simp
qed

lemma DERIV-erlang-CDF-1:
  fixes t :: real
  assumes t: 0 < t
  shows (erlang-CDF k 1 has-real-derivative erlang-density k 1 t) (at t)
proof −
  have op: ∀ F y in nhds t. y ∈ {0<..}
    using t by(intro eventually-nhds-in-open) auto
  have ev: ∀ F y in nhds t. erlang-CDF k 1 y = 1 + − (∑ n≤k. y ^ n * exp (−
y) / fact n)
    using op by eventually-elim (simp add: erlang-CDF-def)
  have ((λy. 1 + − (∑ n≤k. y ^ n * exp (− y) / fact n)) has-real-derivative
          0 + t ^ k * exp (− t) / fact k) (at t)
    by(intro DERIV-add DERIV-const DERIV-erlang-sum)
  hence ((λy. 1 + − (∑ n≤k. y ^ n * exp (− y) / fact n)) has-real-derivative
          erlang-density k 1 t) (at t)
    using t by(simp add: erlang-density-def)
  thus ?thesis
    using DERIV-cong-ev[OF refl ev refl] by simp
qed

lemma DERIV-erlang-CDF:
  fixes t :: real
  assumes l: 0 < l and t: 0 < t
  shows (erlang-CDF k l has-real-derivative erlang-density k l t) (at t)
proof −
  have eq: erlang-CDF k l = (λa. erlang-CDF k 1 (l * a))
    by(rule ext) (rule erlang-CDF-transform[OF l])
```

```
  have ((λa. erlang-CDF k 1 (l * a)) has-real-derivative
         erlang-density k 1 (l * t) * l) (at t)
    using l t
   by(intro DERIV-chain2[OF DERIV-erlang-CDF-1]) (auto intro!: derivative-eq-intros)
  moreover have erlang-density k 1 (l * t) * l = erlang-density k l t
    using l t by(simp add: erlang-density-def mult-less-0-iff field-simps)
  ultimately show ?thesis
    unfolding eq by simp
qed

lemma DERIV-erlang-density:
  fixes t :: real
  assumes t: 0 < t
  shows (erlang-density k l has-real-derivative
           erlang-density k l t * (real k / t − l)) (at t)
proof −
  have op: ∀F y in nhds t. y ∈ {0<..}
    using t by(intro eventually-nhds-in-open) auto
  have ev: ∀F y in nhds t. erlang-density k l y = l ^ Suc k * y ^ k * exp (− l * y)
/ fact k
    using op by eventually-elim (simp add: erlang-density-def)
  have pow: real k * t ^ (k − Suc 0) = real k * t ^ k / t
    by(cases k) (use t in simp-all)
  have ((λy. l ^ Suc k * y ^ k * exp (− l * y) / fact k) has-real-derivative
           l ^ Suc k * (real k * t ^ (k − Suc 0) − l * t ^ k) * exp (− l * t) / fact
k) (at t)
    by(auto intro!: derivative-eq-intros simp: field-simps)
  moreover have l ^ Suc k * (real k * t ^ (k − Suc 0) − l * t ^ k) * exp (− l *
t) / fact k
                = erlang-density k l t * (real k / t − l)
    unfolding pow using t by(simp add: erlang-density-def field-simps)
  ultimately show ?thesis
    using DERIV-cong-ev[OF refl ev refl] by simp
qed

lemma continuous-on-erlang-density: continuous-on {0<..} (erlang-density k l)
proof −
  have eq: continuous-on {0<..} (erlang-density k l)
            = continuous-on {0<..} (λx. l ^ Suc k * x ^ k * exp (− l * x) / fact k)
    by(intro continuous-on-cong refl) (simp add: erlang-density-def)
  show ?thesis
    unfolding eq by(intro continuous-intros) auto
qed

lemma erlang-tail-ratio:
  fixes t :: real
  assumes l: 0 < l and t: 0 < t
  shows (1 − erlang-CDF k l t) / ((l * t) ^ k * exp (− l * t) / fact k)
         = (∑ n≤k. fact k / fact n / (l * t) ^ (k − n))
```

```
proof −
  have lt: 0 < l ∗ t
    using l t by simp
  have tail: 1 − erlang-CDF k l t = (∑ n≤k. (l ∗ t) ^ n ∗ exp (− l ∗ t) / fact n)
    using t by(simp add: erlang-CDF-def)
  have (1 − erlang-CDF k l t) / ((l ∗ t) ^ k ∗ exp (− l ∗ t) / fact k)
        = (∑ n≤k. ((l ∗ t) ^ n ∗ exp (− l ∗ t) / fact n)
                  / ((l ∗ t) ^ k ∗ exp (− l ∗ t) / fact k))
    unfolding tail by(rule sum-divide-distrib)
  also have ... = (∑ n≤k. fact k / fact n / (l ∗ t) ^ (k − n))
  proof(intro sum.cong refl)
    fix n :: nat
    assume n ∈ {..k}
    hence n: n ≤ k
      by simp
    have pow: (l ∗ t) ^ n ∗ (l ∗ t) ^ (k − n) = (l ∗ t) ^ k
      using n by(simp flip: power-add)
    have nz: l ≠ 0 t ≠ 0
      using l t by auto
    show ((l ∗ t) ^ n ∗ exp (− l ∗ t) / fact n) / ((l ∗ t) ^ k ∗ exp (− l ∗ t) / fact
k)
          = fact k / fact n / (l ∗ t) ^ (k − n)
      unfolding pow[symmetric]
      using nz by(simp add: field-simps)
  qed
  finally show ?thesis .
qed

lemma erlang-tail-asymp-equiv:
  assumes l[arith]: 0 < l
  shows (λt. 1 − erlang-CDF k l t) ∼[at-top] (λt. (l ∗ t) ^ k ∗ exp (− l ∗ t) / fact
k)
proof(rule asymp-equivI')
  have lt: filterlim (λt::real. l ∗ t) at-top at-top
    using l by real-asymp
  have lim: ((λt. ∑ n≤k. fact k / fact n / (l ∗ t) ^ (k − n)) ⟶ 1) at-top
  proof −
    have ((λt. ∑ n≤k. fact k / fact n / (l ∗ t) ^ (k − n))
          ⟶ (∑ n≤k. if n = k then 1 else 0)) at-top
    proof(intro tendsto-sum)
      fix n :: nat
      assume n ∈ {..k}
      hence n: n ≤ k
        by simp
     show ((λt. fact k / fact n / (l ∗ t) ^ (k − n)) ⟶ (if n = k then 1 else 0))
at-top
      proof(cases n = k)
        case False
        hence 0 < k − n
```

```
      using n by simp
    from real-tendsto-divide-at-top[OF tendsto-const filterlim-pow-at-top[OF this
lt]]
     show ?thesis
      by(simp add: False del: divide-divide-eq-left)
   qed simp
  qed
  thus ?thesis
   by simp
 qed
 have ev: ∀ F t in at-top. (∑ n≤k. fact k / fact n / (l * t) ^ (k − n))
                        = (1 − erlang-CDF k l t) / ((l * t) ^ k * exp (− l * t) /
fact k)
  by(intro eventually-mono[OF eventually-gt-at-top[of 0::real]]
      erlang-tail-ratio[OF l, symmetric]) simp
 from Lim-transform-eventually[OF lim ev]
 show ((λt. (1 − erlang-CDF k l t) / ((l * t) ^ k * exp (− l * t) / fact k))
       ⟶ 1) at-top .
qed

lemma erlang-von-Mises-ratio:
 fixes t :: real
 assumes l: 0 < l and t: 0 < t
 shows ((l * t) ^ k * exp (− l * t) / fact k)
        * (erlang-density k l t * (real k / t − l)) / (erlang-density k l t)^2
       = (real k / t − l) / l
proof −
 have g0: (l * t) ^ k * exp (− l * t) / fact k ≠ 0
  using l t by simp
 have f: erlang-density k l t = l * ((l * t) ^ k * exp (− l * t) / fact k)
  using l t by(simp add: erlang-density-def field-simps)
 show ?thesis
  unfolding f using g0 l by(simp add: power2-eq-square field-simps)
qed
```

The Erlang distribution is our example of von Mises' condition. Note that the characterization by regular variation is available only for $\gamma > 0$, so for $\gamma = 0$ the condition on the density is the natural tool.

```
theorem erlang-MDA:
 assumes l: 0 < l
 shows domain-of-attraction (erlang-CDF k l) (EVD-cdf 0 0 1)
proof −
 define f' where f' = (λy::real. erlang-density k l y * (real k / y − l))
 note F = right-continuous-mono-erlang-CDF[OF l]
 have unbdd: ¬ bdd-support (erlang-CDF k l)
  by(rule not-bdd-supportI') (rule erlang-CDF-lt-1[OF l])
 have ep: endpoint-filter (erlang-CDF k l) = at-top
  using unbdd by(simp add: endpoint-filter)
 have Flim: (erlang-CDF k l ⟶ 1) (endpoint-filter (erlang-CDF k l))
```

```
    using right-continuous-mono.top[OF F] unbdd by(simp add: endpoint-filter)
  have set-eq: {y. 0 < y ∧ erlang-CDF k l y < 1} = {0<..}
    using erlang-CDF-lt-1[OF l] by auto
  have cont: continuous-on {y. 0 < y ∧ erlang-CDF k l y < 1} f'
    unfolding set-eq f'-def
    by(intro continuous-intros continuous-on-erlang-density) auto
  have lim: ((λy. (1 − erlang-CDF k l y) * f' y / (erlang-density k l y)²)
              ⟶ − 0 − 1) (endpoint-filter (erlang-CDF k l))
  proof −
    have ae: (λt. (1 − erlang-CDF k l t) * f' t / (erlang-density k l t)²)
              ∼[at-top] (λt. ((l * t) ^ k * exp (− l * t) / fact k) * f' t
                                / (erlang-density k l t)²)
      by(intro asymp-equiv-divide asymp-equiv-mult erlang-tail-asymp-equiv[OF l]
          asymp-equiv-refl)
    have ev: ∀F t in at-top. ((l * t) ^ k * exp (− l * t) / fact k) * f' t
                                / (erlang-density k l t)²
                              = (real k / t − l) / l
      unfolding f'-def
      by(intro eventually-mono[OF eventually-gt-at-top[of 0::real]]
          erlang-von-Mises-ratio[OF l])
    have ae': (λt. (1 − erlang-CDF k l t) * f' t / (erlang-density k l t)²)
              ∼[at-top] (λt. (real k / t − l) / l)
      by(rule asymp-equiv-transfer[OF ae - ev]) simp
    have ((λt. (real k / t − l) / l) ⟶ − 0 − 1) at-top
      using l by real-asymp
    with tendsto-asymp-equiv-cong[OF ae'] show ?thesis
      unfolding ep by simp
  qed
  show ?thesis
    by(rule von-Mises-condition[OF F - Flim - - - cont lim])
      (use erlang-CDF-lt-1[OF l] DERIV-erlang-CDF[OF l] DERIV-erlang-density
        erlang-density-pos[OF l] in ‹auto simp: f'-def›)
qed

theorem erlang-normalization:
  fixes x :: real
  assumes l[arith]: 0 < l
  shows (λn. erlang-CDF k l
              ((x + ln (real n) + real k * ln (ln (real n)) − ln (fact k)) / l) ^ n)
        ⟶ EVD-cdf 0 0 1 x
proof −
  have lnz: l ≠ 0
    using l by simp
  define A where A t = x + ln t + real k * ln (ln t) − ln (fact k) for t :: real
  have A-top: filterlim A at-top at-top
    unfolding A-def by real-asymp
  have b-top: filterlim (λt. A t / l) at-top at-top
    using filterlim-tendsto-pos-mult-at-top[OF tendsto-const - A-top, of 1 / l]
    by(simp add: field-simps)
```

have *ratio*: $((\lambda t.\ A\ t\ /\ ln\ t) \longrightarrow 1)\ at\text{-}top$
unfolding *A-def* **by** *real-asymp*
have $(\lambda t.\ 1 - erlang\text{-}CDF\ k\ l\ (A\ t\ /\ l))$
$\sim[at\text{-}top]\ (\lambda t.\ (l * (A\ t\ /\ l))\ \hat{}\ k * exp\ (-\ l * (A\ t\ /\ l))\ /\ fact\ k)$
by(*intro asymp-equiv-compose'*[*OF erlang-tail-asymp-equiv*[*OF l*]] *b-top*)
hence *ae*: $(\lambda t.\ t * (1 - erlang\text{-}CDF\ k\ l\ (A\ t\ /\ l)))$
$\sim[at\text{-}top]\ (\lambda t.\ t * ((l * (A\ t\ /\ l))\ \hat{}\ k * exp\ (-\ l * (A\ t\ /\ l))\ /\ fact\ k))$
by(*intro asymp-equiv-mult asymp-equiv-refl*)
have *ev*: $\forall_F\ t\ in\ at\text{-}top.$
$t * ((l * (A\ t\ /\ l))\ \hat{}\ k * exp\ (-\ l * (A\ t\ /\ l))\ /\ fact\ k)$
$= exp\ (-\ x) * (A\ t\ /\ ln\ t)\ \hat{}\ k$
proof(*intro eventually-mono*[*OF eventually-gt-at-top*[*of exp 1*]])
fix $t :: real$
assume *t*: $exp\ 1 < t$
hence *tpos*: $0 < t$
using *exp-gt-zero*[*of 1*] **by** *linarith*
have *lnt*: $1 < ln\ t$
using *t tpos* **by**(*simp add*: *ln-less-cancel-iff*[*of exp 1 t*, *symmetric*])
have *elnln*: $exp\ (ln\ (ln\ t)) = ln\ t$
using *lnt* **by** *simp*
have *expA*: $exp\ (A\ t) = exp\ x * t * ln\ t\ \hat{}\ k\ /\ fact\ k$
unfolding *A-def*
using *tpos* **by**(*simp add*: *exp-add exp-diff exp-of-nat-mult elnln*)
have $t * ((l * (A\ t\ /\ l))\ \hat{}\ k * exp\ (-\ l * (A\ t\ /\ l))\ /\ fact\ k)$
$= t * (A\ t\ \hat{}\ k\ /\ (fact\ k * exp\ (A\ t)))$
using *lnz* **by**(*simp add*: *exp-minus field-simps*)
also have $... = exp\ (-\ x) * (A\ t\ /\ ln\ t)\ \hat{}\ k$
unfolding *expA*
using *tpos lnt* **by**(*simp add*: *exp-minus field-simps*)
finally show $t * ((l * (A\ t\ /\ l))\ \hat{}\ k * exp\ (-\ l * (A\ t\ /\ l))\ /\ fact\ k)$
$= exp\ (-\ x) * (A\ t\ /\ ln\ t)\ \hat{}\ k$ **.**
qed
from *tendsto-mult*[*OF tendsto-const tendsto-power*[*OF ratio*, *of k*], *of exp* $(-\ x)$]
have *lim*: $((\lambda t.\ exp\ (-\ x) * (A\ t\ /\ ln\ t)\ \hat{}\ k) \longrightarrow exp\ (-\ x))\ at\text{-}top$
by *simp*
have $(\lambda t.\ t * (1 - erlang\text{-}CDF\ k\ l\ (A\ t\ /\ l))) \sim[at\text{-}top]\ (\lambda t.\ exp\ (-\ x) * (A\ t\ /\ ln\ t)\ \hat{}\ k)$
by(*rule asymp-equiv-transfer*[*OF ae - ev*]) *simp*
from *tendsto-asymp-equiv-cong*[*OF this*] *lim*
have $((\lambda t.\ t * (1 - erlang\text{-}CDF\ k\ l\ (A\ t\ /\ l))) \longrightarrow exp\ (-\ x))\ at\text{-}top$
by *simp*
hence *tail*: $(\lambda n.\ real\ n * (1 - erlang\text{-}CDF\ k\ l\ (A\ (real\ n)\ /\ l))) \longrightarrow exp\ (-\ x)$
by(*rule filterlim-compose*[*OF - filterlim-real-sequentially*])
have $(\lambda n.\ erlang\text{-}CDF\ k\ l\ (A\ (real\ n)\ /\ l)\ \hat{}\ n) \longrightarrow EVD\text{-}cdf\ 0\ 0\ 1\ x$
$\longleftrightarrow (\lambda n.\ real\ n * (1 - erlang\text{-}CDF\ k\ l\ (A\ (real\ n)\ /\ l)))$
$\longrightarrow -\ ln\ (EVD\text{-}cdf\ 0\ 0\ 1\ x)$
by(*intro power-tendsto-iff-tendsto-tail erlang-CDF-nonneg*[*OF l*]
less-imp-le[*OF erlang-CDF-lt-1*[*OF l*]]) (*auto simp*: *EVD-cdf-def*)

thus *?thesis*
using *tail* **by**(*simp add*: *A-def EVD-cdf-def*)
qed

5.3.3 Uniform Distribution

lemma *uniform-cdf-eq-GPD*:
fixes *a b* :: *real*
assumes [*arith*]:*a* < *b*
shows *cdf* (*uniform-measure lborel* {*a*..*b*}) = *GPD-cdf* (− *1*) *a* (*b* − *a*)
proof
fix *x*
have ∗:*a* ≤ *x* ⟹ *x* ≤ *b* ⟹ |*1* + (*a* − *x*) / (*b* − *a*)| = *1* + (*a* − *x*) / (*b* − *a*)
by (*smt* (*verit*, *best*) *less-divide-eq-1-pos minus-divide-left*)
have *cdf* (*uniform-measure lborel* {*a*..*b*}) *x* =
(*if a* ≤ *x* ∧ *x* ≤ *b then* (*x* − *a*) / (*b* − *a*)
else if b < *x then 1 else 0*)
by(*auto simp*: *cdf-def*)
also have ... = *GPD-cdf* (− *1*) *a* (*b* − *a*) *x*
using ∗ **by**(*auto simp*: *GPD-cdf-def minus-divide-left*)
finally show *cdf* (*uniform-measure lborel* {*a*..*b*}) *x* = *GPD-cdf* (− *1*) *a* (*b* − *a*) *x* **.**
qed

theorem *uniform-MDA*:
fixes *a b* :: *real*
assumes *a* < *b*
shows *domain-of-attraction* (*cdf* (*uniform-measure lborel* {*a*..*b*})) (*EVD-cdf* (− *1*) *0 1*)
by (*simp add*: *GPD-cdf-MDA assms uniform-cdf-eq-GPD*)

5.3.4 Cauchy Distribution

definition *Cauchy-density* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* **where**
Cauchy-density μ σ *x* ≡ σ / (*pi* ∗ (σ² + (*x* − μ)²))

abbreviation *std-Cauchy-density* :: *real* ⇒ *real* **where**
std-Cauchy-density ≡ *Cauchy-density 0 1*

abbreviation *std-Cauchy-distribution* :: *real measure* **where**
std-Cauchy-distribution ≡ *density lborel std-Cauchy-density*

lemma *std-Cauchy-density-def*: *std-Cauchy-density x* = *1* / (*pi* ∗ (*1* + *x*²))
by(*simp add*: *Cauchy-density-def*)

lemma *Cauchy-density-nonneg*[*simp*]: *0* < σ ⟹ *0* ≤ *Cauchy-density* μ σ *x*
by(*simp add*: *Cauchy-density-def*)

lemma *Cauchy-density-measurable*[*measurable*]:
Cauchy-density μ σ ∈ *borel-measurable borel*

unfolding *Cauchy-density-def* **by** *measurable*

lemma *Cauchy-density-uminus*: *Cauchy-density* μ σ $(-x)$ = *Cauchy-density* $(-\mu)$ σ x
by(*simp add*: *Cauchy-density-def power2-eq-square algebra-simps*)

lemma *arctan-div-pi-bounds*:
shows *arctan-div-pi-pos*: $0 < 1 / 2 + \arctan z / pi$
and *arctan-div-pi-lt-1*: $1 / 2 + \arctan z / pi < 1$
using *arctan-lbound*[*of z*] *arctan-ubound*[*of z*] *pi-gt-zero*
by(*auto simp*: *field-simps*)

lemma *DERIV-Cauchy-cdf*:
assumes *sigma*[*arith*]: $0 < \sigma$
shows *DERIV* $(\lambda y.\ \arctan\ ((y - \mu) / \sigma) / pi)\ x :>$ *Cauchy-density* μ σ x
proof −
have *ne*: $\sigma^2 + (x - \mu)^2 \neq 0$
using *add-pos-nonneg*[*of* σ^2 $(x - \mu)^2$] **by** *simp*
have *sq*: $1 + ((x - \mu) / \sigma)^2 = (\sigma^2 + (x - \mu)^2) / \sigma^2$
by(*simp add*: *field-simps*)
have *eq*: $1 / (1 + ((x - \mu) / \sigma)^2) * (1 / \sigma) / pi =$ *Cauchy-density* μ σ x
unfolding *sq Cauchy-density-def*
using *ne* **by**(*simp add*: *field-simps power2-eq-square*)
show *?thesis*
unfolding *eq*[*symmetric*]
by(*auto intro*!: *derivative-eq-intros simp*: *inverse-eq-divide*)
qed

lemma *Cauchy-nn-integral-atLeast*:
assumes *sigma*[*arith*]: $0 < \sigma$
shows $(\int^+ y \in \{x..\}.$ *Cauchy-density* μ σ $y\ \partial lborel)$
$=$ *ennreal* $(1 / 2 - \arctan\ ((x - \mu) / \sigma) / pi)$
proof −
have $((\lambda y.\ \arctan\ ((y - \mu) / \sigma) / pi) \longrightarrow pi / 2 / pi)$ *at-top*
using *filterlim-tendsto-pos-mult-at-top*[*OF tendsto-const*[*of 1 / σ*] -
filterlim-tendsto-add-at-top[*OF tendsto-const*[*of* $-\mu$] *filterlim-ident*]]
by(*intro tendsto-divide tendsto-const filterlim-compose*[*OF tendsto-arctan-at-top*])
auto
hence *lim*: $((\lambda y.\ \arctan\ ((y - \mu) / \sigma) / pi) \longrightarrow 1 / 2)$ *at-top*
by *simp*
show *?thesis*
by(*subst nn-integral-FTC-atLeast*
[**where** *F*=$\lambda y.\ \arctan\ ((y - \mu) / \sigma) / pi$ **and** *T*=*1 / 2*])
(*auto intro*!: *DERIV-Cauchy-cdf lim*)
qed

lemma *Cauchy-nn-integral-atMost*:
assumes *sigma*[*arith*]: $0 < \sigma$
shows $(\int^+ y \in \{..x\}.$ *Cauchy-density* μ σ $y\ \partial lborel)$

```
        = ennreal (1 / 2 + arctan ((x − μ) / σ) / pi)
proof −
  have ind: indicator {..x} (− y) = (indicator {− x..} y :: ennreal) for y :: real
    by(simp add: indicator-def)
  have arg: (− x − − μ) / σ = − ((x − μ) / σ)
    by(simp add: minus-divide-left)
  have (∫⁺y∈{..x}. Cauchy-density μ σ y ∂lborel)
       = (∫⁺y∈{− x..}. Cauchy-density (− μ) σ y ∂lborel)
    by(subst nn-integral-real-affine
        [where t=0 and c=− 1 and f=λy. ennreal (Cauchy-density μ σ y) ∗
indicator {..x} y])
      (simp-all add: ind Cauchy-density-uminus)
  also have ... = ennreal (1 / 2 − arctan ((− x − − μ) / σ) / pi)
    by(rule Cauchy-nn-integral-atLeast[OF sigma])
  also have ... = ennreal (1 / 2 + arctan ((x − μ) / σ) / pi)
    unfolding arg
    by(simp add: arctan-minus)
  finally show ?thesis .
qed

lemma prob-space-Cauchy-density:
  assumes sigma[arith]: 0 < σ
  shows prob-space (density lborel (Cauchy-density μ σ))
proof
  have split: (∫⁺y. ennreal (Cauchy-density μ σ y) ∂lborel)
              = (∫⁺y∈{..0}. Cauchy-density μ σ y ∂lborel)
                + (∫⁺y∈{0<..}. Cauchy-density μ σ y ∂lborel)
    by(subst nn-integral-add[symmetric])
      (auto intro!: nn-integral-cong simp: indicator-def)
  have null: (∫⁺y∈{0<..}. Cauchy-density μ σ y ∂lborel)
             = (∫⁺y∈{0..}. Cauchy-density μ σ y ∂lborel)
    by(intro nn-integral-cong-AE AE-I[where N={0}]) (auto simp: indicator-def)
  have nn1: 0 ≤ 1 / 2 + arctan ((0 − μ) / σ) / pi
    using arctan-div-pi-pos[of (0 − μ) / σ] by linarith
  have nn2: 0 ≤ 1 / 2 − arctan ((0 − μ) / σ) / pi
    using arctan-div-pi-lt-1[of (0 − μ) / σ] by linarith
  show emeasure (density lborel (Cauchy-density μ σ))
       (space (density lborel (Cauchy-density μ σ))) = 1
    using nn1 nn2
    by(simp add: emeasure-density split null Cauchy-nn-integral-atMost
       Cauchy-nn-integral-atLeast flip: ennreal-plus)
qed

corollary real-distribution-Cauchy:
  0 < σ ⟹ real-distribution (density lborel (Cauchy-density μ σ))
 by(simp add: real-distribution-def real-distribution-axioms-def prob-space-Cauchy-density)

lemma Cauchy-cdf-def:
  assumes sigma[arith]: 0 < σ
```

shows *cdf* (*density lborel* (*Cauchy-density* μ σ)) x = *1 / 2 + arctan* ((x − μ) / σ) / *pi*
using *arctan-div-pi-pos*[*of* (x − μ) / σ]
by(*simp add*: *cdf-def measure-def emeasure-density Cauchy-nn-integral-atMost*)

lemma *std-Cauchy-cdf-def*: *cdf std-Cauchy-distribution x = 1 / 2 + arctan x / pi*
by(*simp add*: *Cauchy-cdf-def*)

lemma *Cauchy-cdf-location-scale*:
assumes *sigma*: *0* < σ
shows *cdf* (*density lborel* (*Cauchy-density* μ σ)) x
= *cdf std-Cauchy-distribution* ((x − μ) / σ)
by(*simp add*: *Cauchy-cdf-def*[*OF sigma*] *Cauchy-cdf-def*[*OF zero-less-one*])

lemma *std-Cauchy-tail-asymp-equiv*:
(λ*t*. *1* − *cdf std-Cauchy-distribution t*) ∼[*at-top*] (λ*t*. *1* / (*pi* ∗ *t*))
proof(*rule asymp-equivI'*)
have *eq*: ∀ *F t in at-top*. (*1 / 2* − *arctan t / pi*) ∗ (*pi* ∗ *t*)
= (*1* − *cdf std-Cauchy-distribution t*) / (*1* / (*pi* ∗ *t*))
using *eventually-gt-at-top*[*of 0*::*real*]
by *eventually-elim* (*simp add*: *std-Cauchy-cdf-def*)
have ((λ*t*::*real*. (*1 / 2* − *arctan t / pi*) ∗ (*pi* ∗ *t*)) ⟶ *1*) *at-top*
by *real-asymp*
from *Lim-transform-eventually*[*OF this eq*]
show ((λ*t*. (*1* − *cdf std-Cauchy-distribution t*) / (*1* / (*pi* ∗ *t*))) ⟶ *1*) *at-top*
.
qed

theorem *std-Cauchy-MDA*: *domain-of-attraction* (*cdf std-Cauchy-distribution*) (*EVD-cdf 1 0 1*)
proof −
have *F*: *right-continuous-mono* (*cdf std-Cauchy-distribution*) *0 1*
by(*auto intro*!: *right-continuous-mono-cdf real-distribution-Cauchy*)
have *lt1*: *cdf std-Cauchy-distribution x* < *1* **for** *x*
using *arctan-div-pi-lt-1*[*of x*] **by**(*simp add*: *std-Cauchy-cdf-def*)
have *regularly-varying* (λ*x*. *1* − *cdf std-Cauchy-distribution x*) (− *1*)
proof(*rule regularly-varyingI-total*)
show (λ*x*. *1* − *cdf std-Cauchy-distribution x*) ∈ *borel-measurable borel*
by(*auto intro*!: *borel-measurable-continuous-onI simp*: *std-Cauchy-cdf-def*)
next
fix *x* :: *real*
assume *x*: *0* < *x*
have ((λ*t*. (*1 / 2* − *arctan* (*t* ∗ *x*) / *pi*) / (*1 / 2* − *arctan t / pi*))
⟶ *x powr* (− *1*)) *at-top*
using *x* **by**(*real-asymp simp add*: *powr-minus*)
thus ((λ*t*. (*1* − *cdf std-Cauchy-distribution* (*t* ∗ *x*))
/ (*1* − *cdf std-Cauchy-distribution t*)) ⟶ *x powr* (− *1*)) *at-top*
by(*simp add*: *std-Cauchy-cdf-def*)

qed(*use lt1* **in** *simp*)
thus *?thesis*
using *MDA-regularly-varying-pos*[*OF F zero-less-one*] *lt1* **by** *simp*
qed

corollary *Cauchy-MDA*:
assumes $0 < \sigma$
shows *domain-of-attraction* (*cdf* (*density lborel* (*Cauchy-density* μ σ))) (*EVD-cdf 1 0 1*)
unfolding *Cauchy-cdf-location-scale*[*OF assms*]
by(*intro domain-of-attraction-affine assms std-Cauchy-MDA*)

theorem *std-Cauchy-normalization*:
fixes x :: *real*
shows (λn. *cdf std-Cauchy-distribution* (*real n* / *pi* ∗ *x*) ^ *n*) ⟶ *EVD-cdf 1 1 1 x*
proof −
interpret *C*: *real-distribution std-Cauchy-distribution*
by(*rule real-distribution-Cauchy*) *simp*
show *?thesis*
proof(*cases* $0 < x$)
case *x*: *True*
have *EVD*: *EVD-cdf 1 1 1 x* = *exp* (− (*1* / *x*))
using *x* **by**(*simp add*: *EVD-cdf-def powr-minus-divide*)
have *top*: *filterlim* (λ*t*::*real*. *t* / *pi* ∗ *x*) *at-top at-top*
using *filterlim-at-top-mult-tendsto-pos*[*OF tendsto-const*[*of x* / *pi*] - *filterlim-ident*] *x*
by(*simp add*: *field-simps*)
have (λt. *1* − *cdf std-Cauchy-distribution* (*t* / *pi* ∗ *x*))
∼[*at-top*] (λt. *1* / (*pi* ∗ (*t* / *pi* ∗ *x*)))
by(*intro asymp-equiv-compose'*[*OF std-Cauchy-tail-asymp-equiv*] *top*)
hence *ae*: (λt. *t* ∗ (*1* − *cdf std-Cauchy-distribution* (*t* / *pi* ∗ *x*)))
∼[*at-top*] (λt. *t* ∗ (*1* / (*pi* ∗ (*t* / *pi* ∗ *x*))))
by(*intro asymp-equiv-mult asymp-equiv-refl*)
have *ev*: ∀ *F t in at-top*. *t* ∗ (*1* / (*pi* ∗ (*t* / *pi* ∗ *x*))) = *1* / *x*
using *eventually-gt-at-top*[*of 0*::*real*]
by *eventually-elim* (*use x* **in** ‹*simp add*: *field-simps*›)
have (λt. *t* ∗ (*1* − *cdf std-Cauchy-distribution* (*t* / *pi* ∗ *x*))) ∼[*at-top*] (λt. *1* / *x*)
by(*rule asymp-equiv-transfer*[*OF ae - ev*]) *simp*
from *tendsto-asymp-equiv-cong*[*OF this*]
have ((λt. *t* ∗ (*1* − *cdf std-Cauchy-distribution* (*t* / *pi* ∗ *x*))) ⟶ *1* / *x*) *at-top*
by *simp*
hence *tail*: (λn. *real n* ∗ (*1* − *cdf std-Cauchy-distribution* (*real n* / *pi* ∗ *x*)))
⟶ *1* / *x*
by(*rule filterlim-compose*[*OF - filterlim-real-sequentially*])
have (λn. *cdf std-Cauchy-distribution* (*real n* / *pi* ∗ *x*) ^ *n*) ⟶ *exp* (− (*1* / *x*))

```
        ⟷ (λn. real n * (1 - cdf std-Cauchy-distribution (real n / pi * x)))
            ⟶ - ln (exp (- (1 / x)))
    using x
    by(intro power-tendsto-iff-tendsto-tail C.cdf-nonneg C.cdf-bounded-prob) auto
  thus ?thesis
    unfolding EVD
    using tail by simp
 next
  case x: False
  have EVD: EVD-cdf 1 1 1 x = 0
    using x by(simp add: EVD-cdf-def)
  have arg: real n / pi * x ≤ 0 for n
    using x mult-nonneg-nonpos[of real n x] by(simp add: divide-nonpos-pos)
  have half: cdf std-Cauchy-distribution (real n / pi * x) ≤ 1 / 2 for n
    using arctan-monotone'[OF arg, of n] arctan-zero-zero
    by(simp add: std-Cauchy-cdf-def divide-nonpos-pos)
  show ?thesis
    unfolding EVD
  proof(rule tendsto-sandwich[where f=λ-. 0 and h=λn. (1 / 2 :: real) ^ n])
    show ∀F n in sequentially. cdf std-Cauchy-distribution (real n / pi * x) ^ n
                                ≤ (1 / 2) ^ n
      by(intro always-eventually allI power-mono half C.cdf-nonneg)
    show (λn. (1 / 2 :: real) ^ n) ⟶ 0
      by(rule LIMSEQ-realpow-zero) auto
  qed(use C.cdf-nonneg in auto)
 qed
qed

end
```

References

[1] A. A. Balkema and L. de Haan. Residual life time at great age. *The Annals of Probability*, 2(5):792–804, 1974.

[2] L. de Haan and A. Ferreira. *Extreme Value Theory: An Introduction*. Springer Series in Operations Research and Financial Engineering. Springer, 2006.

[3] P. Embrechts, C. Klüppelberg, and T. Mikosch. *Modelling Extremal Events for Insurance and Finance*, volume 33 of *Stochastic Modelling and Applied Probability*. Springer, 1997.

[4] R. A. Fisher and L. H. C. Tippett. Limiting forms of the frequency distribution of the largest or smallest member of a sample. *Mathematical Proceedings of the Cambridge Philosophical Society*, 24(2):180–190, 1928.

[5] B. Gnedenko. Sur la distribution limite du terme maximum d'une série aléatoire. *Annals of Mathematics*, 44(3):423–453, 1943.

[6] J. Pickands, III. Statistical inference using extreme order statistics. *The Annals of Statistics*, 3(1):119–131, 1975.