The Incomplete Gamma Function

Manuel Eberl

July 8, 2026

Abstract

This entry provides two special functions, the lower and upper incomplete gamma functions. Similarly to the "complete" gamma function $\Gamma(s)$, which is defined as $\Gamma(s) = \int_0^\infty t^{s-1}e^{-t}\,\mathrm{d}t$ for $\mathrm{Re}(s) > 0$ and by analytic continuation elsewhere, these are defined as $\gamma(s,z) = \int_0^z t^{s-1}e^{-t}\,\mathrm{d}t$ and $\Gamma(s,z) = \int_z^\infty t^{s-1}e^{-t}\,\mathrm{d}t$, respectively, for $\mathrm{Re}(s) > 0$ and analytically continued to the entire complex plane.

$\gamma(s,z)$ is constructed using the regularised hypergeometric series and $\Gamma(s,z)$ via its contour integral representation. Various results are provided, including:

- holomorphicity, continuity, limits, and derivatives
- series and integral representations
- shift identities such as $\Gamma(s+1,z) = s\Gamma(s,z) + z^s e^{-z}$
- the identity $\gamma(s,z) + \Gamma(s,z) = \Gamma(s)$
- the fact that $\Gamma(s,z) \to \Gamma(s)$ as $z \to 0$ within a certain region
- closed forms for $\Gamma(n,z)$ and $\gamma(n,z)$, where n is a positive integer
- the connection to the error function via $\Gamma(\frac{1}{2},z) = \sqrt{\pi}\cdot\mathrm{erf}(\sqrt{z})$

Contents

1 A "safe" power operator for real and complex numbers

theory *Safe_Power*
  imports *"HOL-Complex_Analysis.Complex_Analysis"*
begin

Isabelle/HOL currently has three different power operators:

- *(^)*, denoted x^n, where *n* is a natural number and *x* is an element of a multiplicatively written monoid (*monoid_mult*).
- *(powi)*, denoted *x powi n*, where *n* is an integer and *x* is an element of a multiplicatively written monoid with some kind of inverse operation (typically a *division_ring*).
- *(powr)*, denoted *x powr y*, where *x* and *y* are complete normed real algebra with a 1 and some kind of natural logarithm. In practice, the only reasonable examples of such a structure are probably *real* and *complex*. It is defined as *x powr y = exp (ln x * y)*, except when *x = 0*, in which case it returns *0* by convention.

All three of these operators are required, since none of them is a generalisation of another. For example, the *(^)* operator is the most restrictive in what the right argument can be, but it is the most general in terms of what the left argument can be.

For the most part, the three operators agree in all the cases where they are simultaneously defined, but there are some caveats.

First of all, note that different conventions apply for 0^0: We have x^n *= 1* and *x powr n = 1* for any *x*, including for *x = 0*, whereas *0 powr y = 0* for any *y*, including *y = 0*.

Second, for negative real *x*, the *(powr)* operator inherits somewhat strange behaviour from the real-valued *ln* operator, which in Isabelle/HOL is defined symmetrically, i.e. *ln (- x) = ln x*, so we also have *(- x) powr y = x powr y* for real *x* and *y*. This means that while $(-1)^3 = -1$ as expected, we have *(- 1) powr 3 = 1*. It is therefore better to consider *x powr y* to be undefined for negative real *x*.

In the following, we will define a "safe" version of the *(powr)* operator that coincides with *(^)* and *(powi)* whenever applicable.

First, we define the inverse of the canonical ring homomorphism $\mathbb{Z} \to R$ using the choice operator, i.e. for any $x \in \mathbb{Z}$, the operator *the_int* gives us an integer n such that $x = n = \underbrace{1 + \ldots + 1}_{n \text{ times}}$. For $x \notin \mathbb{Z}$, the operator's behaviour is unspecified.

definition *the_int :: "'a :: ring_char_0 ⇒ int"* **where**
"the_int x = (THE n. x = of_int n)"

lemma *the_int_of_int [simp]: "the_int (of_int n :: 'a :: ring_char_0) = n"*
⟨*proof*⟩

lemma *the_int_eqI: "of_int n = x ⟹ the_int x = n"*
⟨*proof*⟩

lemma *the_int_0 [simp]: "the_int 0 = 0"*
⟨*proof*⟩

lemma *the_int_1 [simp]: "the_int 1 = 1"*
⟨*proof*⟩

lemma *the_int_of_nat [simp]: "the_int (of_nat n) = int n"*
⟨*proof*⟩

lemma *the_int_numeral [simp]: "the_int (numeral n) = numeral n"*
⟨*proof*⟩

lemma *the_int_uminus [simp]: "x ∈ ℤ ⟹ the_int (-x) = -the_int x"*
⟨*proof*⟩

lemma *of_int_the_int: "x ∈ ℤ ⟹ of_int (the_int x) = x"*
⟨*proof*⟩

lemma *the_int_add: "x ∈ ℤ ⟹ y ∈ ℤ ⟹ the_int (x + y) = the_int x + the_int y"*
⟨*proof*⟩

lemma *the_int_diff: "x ∈ ℤ ⟹ y ∈ ℤ ⟹ the_int (x - y) = the_int x - the_int y"*
⟨*proof*⟩

lemma *the_int_mult: "x ∈ ℤ ⟹ y ∈ ℤ ⟹ the_int (x * y) = the_int x * the_int y"*
⟨*proof*⟩

lemma *the_int_power: "x ∈ ℤ ⟹ the_int (x ^ n) = the_int x ^ n"*
⟨*proof*⟩

Now we simply define the *powr'* operator as a version of the *(powr)* operator that falls back to the *(powi)* operator whenever possible.

definition *powr' :: "'a :: {division_ring, ln} ⇒ 'a ⇒ 'a"* (**infixr** *<powr'>* *80*) **where**
"x powr' y = (if y ∈ ℤ then x powi (the_int y) else x powr y)"

lemma *powr'_powr: "y ∉ ℤ ⟹ x powr' y = x powr y"*
⟨*proof*⟩

lemma *powr'_0_left [simp]: "y ≠ 0 ⟹ 0 powr' y = 0"*
⟨*proof*⟩

lemma *powr'_0_left_if: "0 powr' y = (if y = 0 then 1 else 0)"*
⟨*proof*⟩

lemma *powr'_of_int [simp]: "x powr' of_int n = x powi n"*
⟨*proof*⟩

lemma *powr'_of_nat [simp]: "x powr' of_nat n = x ^ n"*
⟨*proof*⟩

lemma *powr'_0 [simp]: "x powr' 0 = 1"*
⟨*proof*⟩

lemma *powr'_1 [simp]: "x powr' 1 = x"*
⟨*proof*⟩

lemma *powr'_numeral [simp]: "x powr' numeral n = x ^ numeral n"*
⟨*proof*⟩

If x is positive or if x is non-negative and y is positive, the safe power operator and the normal power operator agree..

lemma *powr'_real: "x ≥ 0 ⟹ x ≠ 0 ∨ y > 0 ⟹ x powr' y = x powr (y :: real)"*
⟨*proof*⟩

lemma *powr'_real_pos: "x > 0 ⟹ x powr' y = x powr (y :: real)"*
⟨*proof*⟩

For complex inputs, the two operators always agree except in the case of 0^0.

lemma *powr'_complex: "x ≠ 0 ∨ y ≠ 0 ⟹ x powr' y = x powr (y :: complex)"*
⟨*proof*⟩

lemma *powr'_complex_of_real:*
"x ≥ 0 ∨ y ∈ ℤ ⟹ complex_of_real x powr' of_real y = (of_real (x powr' y))"
⟨*proof*⟩

lemma *powr'_Reals_eq: "⟦x ∈ ℝ; y ∈ ℝ; Re x ≥ 0⟧ ⟹ x powr' y = of_real (Re x powr' Re y)"*
⟨*proof*⟩

For this reason, the *(powr')* operator is holomorphic in both inputs except for a branch cut along the non-positive reals.

lemma *holomorphic_on_powr' [holomorphic_intros]:*

assumes *"f holomorphic_on A" "g holomorphic_on A" "⋀x. x ∈ A ⟹ f x ∉ ℝ≤0"*
shows *"(λx. f x powr' g x) holomorphic_on A"*
⟨*proof*⟩

lemma *analytic_on_powr' [analytic_intros]:*
assumes *"f analytic_on A" "g analytic_on A" "⋀x. x ∈ A ⟹ f x ∉ ℝ≤0"*
shows *"(λx. f x powr' g x) analytic_on A"*
⟨*proof*⟩

lemma *has_field_derivative_powr_complex [derivative_intros]:*
assumes *"(f has_field_derivative f') (at x within A)"*
assumes *"(g has_field_derivative g') (at x within A)"*
assumes *"f x ∉ (ℝ≤0 :: complex set)"*
shows *"((λx. f x powr g x) has_field_derivative*
*(f x powr g x * (f' / f x * g x + g' * ln (f x)))) (at x within A)"*
⟨*proof*⟩

lemma *has_field_derivative_powr'_complex [derivative_intros]:*
assumes *"(f has_field_derivative f') (at x within A)"*
assumes *"(g has_field_derivative g') (at x within A)"*
assumes *"f x ∉ (ℝ≤0 :: complex set)"*
shows *"((λx. f x powr' g x) has_field_derivative*
*(f x powr' g x * (f' / f x * g x + g' * ln (f x)))) (at x within A)"*
⟨*proof*⟩

lemma *has_field_derivative_powr'_Ints:*
assumes *"(f has_field_derivative f') (at x within A)"*
assumes *"c ∈ (ℤ :: 'a :: {real_normed_field, ln} set)" "c ∈ ℕ ∨ f x ≠ 0"*
shows *"((λx. f x powr' c) has_field_derivative (c * f x powr' (c-1) * f')) (at x within A)"*
⟨*proof*⟩

lemma *continuous_on_powr'_complex [continuous_intros]:*
assumes *"A ⊆ {z. Re (f z) ≥ 0 ∨ Im (f z) ≠ 0}"*
assumes *"⋀z. z ∈ A ⟹ f z = 0 ⟹ Re (g z) > 0"*
assumes *"continuous_on A f" "continuous_on A g"*
shows *"continuous_on A (λz. f z powr' g z)"*
⟨*proof*⟩

lemma *tendsto_powr'_complex [tendsto_intros]:*
fixes *f g :: "_ ⇒ complex"*
assumes *"a ∉ ℝ≤0 ∨ (a = 0 ∧ Re b > 0)"* **and** *"(f ⟶ a) F" "(g ⟶*

```
b) F"
  shows   "((λz. f z powr' g z) ⟶ a powr' b) F"
```
⟨*proof*⟩

lemma `ln_at_0': "filterlim (ln :: real ⇒ real) at_bot (at 0)"`
⟨*proof*⟩

lemma `tendsto_powr_real [tendsto_intros]:`
  fixes `f g :: "_ ⇒ real"`
  assumes `"(f ⟶ a) F" "(g ⟶ b) F"`
  assumes `"a = 0 ⟶ b > 0"`
  shows   `"((λz. f z powr g z) ⟶ a powr b) F"`
⟨*proof*⟩

lemmas `[tendsto_intros del] = tendsto_powr tendsto_powr'`

lemma `tendsto_powr'_real [tendsto_intros]:`
  fixes `f g :: "_ ⇒ real"`
  assumes `"(f ⟶ a) F" "(g ⟶ b) F"`
  assumes `"a > 0 ∨ (a = 0 ∧ b > 0)"`
  shows   `"((λz. f z powr' g z) ⟶ a powr' b) F"`
⟨*proof*⟩

lemma `continuous_powr'_complex [continuous_intros]:`
  assumes `"continuous F f" "continuous F g"`
  assumes `"Re (f (netlimit F)) ≥ 0 ∨ Im (f (netlimit F)) ≠ 0"`
  assumes `"f (netlimit F) = 0 ⟶ Re (g (netlimit F)) > 0"`
  shows   `"continuous F (λz. f z powr' g z :: complex)"`
  ⟨*proof*⟩

end

2 Auxiliary material on filters and dominated convergence

theory `More_Dominated_Convergence`
  imports `"HOL-Complex_Analysis.Complex_Analysis" "HOL-Library.Going_To_Filter"`
begin

2.1 Linear orders with unbounded sequences

We define type classes for linear orders that contain sequences that "tend to infinity". These are also known as "countably cofinite".

class `seq_at_top = linorder +`
  assumes `seq_at_top_aux: "∃f::nat ⇒ 'a. ∀x. eventually (λn. f n ≥ x) sequentially"`

lemma *seq_at_top: "∃f::nat ⇒ 'a :: seq_at_top. filterlim f at_top sequentially"*
⟨*proof*⟩

lemma *seq_at_topI [intro?]:*
"filterlim (f :: nat ⇒ 'a :: linorder) at_top sequentially ⟹ OFCLASS('a, seq_at_top_class)"
⟨*proof*⟩

class *seq_at_bot = linorder +*
assumes *seq_at_bot_aux: "∃f::nat ⇒ 'a. ∀x. eventually (λn. f n ≤ x) sequentially"*

lemma *seq_at_bot: "∃f::nat ⇒ 'a :: seq_at_bot. filterlim f at_bot sequentially"*
⟨*proof*⟩

lemma *seq_at_botI [intro?]:*
"filterlim (f :: nat ⇒ 'a :: linorder) at_bot sequentially ⟹ OFCLASS('a, seq_at_bot_class)"
⟨*proof*⟩

An archimedian field is countably cofinite, i.e. there exist sequences that tend to $\pm\infty$.

context *archimedean_field*
begin

lemma *eventually_of_nat_ge: "eventually (λn. of_nat n ≥ x) sequentially"*
⟨*proof*⟩

subclass *seq_at_top*
⟨*proof*⟩

subclass *seq_at_bot*
⟨*proof*⟩

end

Complete linear orders are obviously countably cofinite, since even the singleton set *{top}* is cofinite.

context *complete_linorder*
begin

subclass *seq_at_top*
⟨*proof*⟩

subclass *seq_at_bot*
⟨*proof*⟩

end

instance *nat :: seq_at_bot*
⟨*proof*⟩

instance *nat :: seq_at_top*
⟨*proof*⟩

instance *int :: seq_at_top*
⟨*proof*⟩

instance *int :: seq_at_bot*
⟨*proof*⟩

2.2 Countably generated filters

For convenience, we show that if we have a countably generated filter, we can assume w.l.o.g. that the sequence of sets generating it is decreasing.

lemma *countably_generated_filterI_decseq:*
assumes *"antimono_on UNIV B" "⋀P. eventually P F ⟷ (∃i::nat. ∀x∈B i. P x)"*
shows *"countably_generated_filter F"*
⟨*proof*⟩

lemma *countably_generated_filter_iff_decseq:*
"countably_generated_filter F ⟷
(∃B. antimono_on UNIV B ∧ (∀P. eventually P F ⟷ (∃i::nat. ∀x∈B i. P x)))"
⟨*proof*⟩

lemma *countably_generated_filter_altdef:*
"countably_generated_filter F ⟷ (∃U. countable U ∧ F = (INF X∈U. principal X))"
⟨*proof*⟩

Countably generated filters are sequential, i.e. if any sequence that tends to the filter is eventually contained in some set, then that set is in the filter.

lemma *countably_generated_filter_sequential:*
assumes *"countably_generated_filter F"*
assumes *"(⋀f. filterlim f F sequentially ⟹ eventually (λn. P (f n)) sequentially)"*
shows *"eventually P F"*
⟨*proof*⟩

named_theorems *countably_generated_filter_intros*

lemma *countably_generated_filter_top [countably_generated_filter_intros]:*

"countably_generated_filter top_class.top"
⟨*proof*⟩

lemma *countably_generated_filter_bot [countably_generated_filter_intros]:*
"countably_generated_filter bot_class.bot"
⟨*proof*⟩

lemma *countably_generated_filter_principal [countably_generated_filter_intros]:*
"countably_generated_filter (principal A)"
⟨*proof*⟩

lemma *countably_based_filtermap [countably_generated_filter_intros]:*
assumes *"countably_generated_filter F"*
shows *"countably_generated_filter (filtermap f F)"*
⟨*proof*⟩

lemma *countably_based_filtercomap [countably_generated_filter_intros]:*

assumes *"countably_generated_filter F"*
shows *"countably_generated_filter (filtercomap f F)"*
⟨*proof*⟩

lemma *countably_generated_filter_inf [countably_generated_filter_intros]:*
assumes *"countably_generated_filter F" "countably_generated_filter G"*
shows *"countably_generated_filter (inf F G)"*
⟨*proof*⟩

lemma *countably_generated_filter_sup [countably_generated_filter_intros]:*
assumes *"countably_generated_filter F" "countably_generated_filter G"*
shows *"countably_generated_filter (sup F G)"*
⟨*proof*⟩

lemma *countably_generated_filter_prod [countably_generated_filter_intros]:*
assumes *"countably_generated_filter F" "countably_generated_filter G"*
shows *"countably_generated_filter (F ×$_F$ G)"*
⟨*proof*⟩

lemma *countably_generated_filter_Inf [countably_generated_filter_intros]:*
assumes *"countable F" "⋀G. G ∈ F ⟹ countably_generated_filter G"*
shows *"countably_generated_filter (Inf F)"*
⟨*proof*⟩

lemma *countably_generated_filter_Sup_finite [countably_generated_filter_intros]:*
assumes *"finite F" "⋀G. G ∈ F ⟹ countably_generated_filter G"*
shows *"countably_generated_filter (Sup F)"*
⟨*proof*⟩

lemma *countably_generated_filter_going_to [countably_generated_filter_intros]:*
assumes *"countably_generated_filter F"*
shows *"countably_generated_filter (f going_to F within A)"*
⟨*proof*⟩

lemma *countably_generated_filter_at_top [countably_generated_filter_intros]:*
"countably_generated_filter (at_top :: 'a :: seq_at_top filter)"
⟨*proof*⟩

lemma *countably_generated_filter_at_top [countably_generated_filter_intros]:*
"countably_generated_filter (at_top :: 'a :: {second_countable_topology,linorder_topology
filter)"
⟨*proof*⟩

lemma *countably_generated_filter_at_bot [countably_generated_filter_intros]:*
"countably_generated_filter (at_bot :: 'a :: seq_at_bot filter)"
⟨*proof*⟩

lemma *countably_generated_filter_nhds [countably_generated_filter_intros]:*
"countably_generated_filter (nhds (x :: 'a :: first_countable_topology))"
⟨*proof*⟩

lemma *countably_generated_filter_at_within [countably_generated_filter_intros]:*
"countably_generated_filter (at (x :: 'a :: first_countable_topology)
within A)"
⟨*proof*⟩

lemma *at_infinity_eq_filtercomap: "at_infinity = filtercomap norm at_top"*
⟨*proof*⟩

lemma *countably_generated_filter_at_infinity [countably_generated_filter_intros]:*
"countably_generated_filter at_infinity"
⟨*proof*⟩

lemmas *[countably_generated_filter_intros] = countably_generated_uniformity*

2.3 Sequential filters

We call a filter *sequential* if it can be "approached" by sequences in the sense that if a a properties holds eventually on all sequences that approach the filter, then it also holds eventually for the filter.

Importantly, any countably generated filter is sequential.

locale *sequential_filter* =
  fixes *F :: "'a filter"*
  assumes *approachable:*
    "(⋀f. filterlim f F sequentially ⟹ eventually (λn. P (f n)) sequentially)
⟹ eventually P F"
begin

lemma *filterlim_sequentially_imp_filterlim:*
  assumes *"⋀X. filterlim X F sequentially ⟹ filterlim (λn. f (X n))*
G sequentially"
  shows *"filterlim f G F"*
  ⟨*proof*⟩

end

lemma *sequential_filtermap:*
  assumes *"sequential_filter F"*
  shows *"sequential_filter (filtermap (g :: 'a ⇒ 'b) F)"*
⟨*proof*⟩

lemma *countably_generated_filter_imp_sequential_filter:*
  assumes *"countably_generated_filter F"*
  shows *"sequential_filter F"*
  ⟨*proof*⟩

interpretation *bot: sequential_filter "bot_class.bot"*
  ⟨*proof*⟩

interpretation *top: sequential_filter "top_class.top"*
  ⟨*proof*⟩

interpretation *principal: sequential_filter "principal A"*
  ⟨*proof*⟩

interpretation *nhds: sequential_filter "nhds (x :: 'a :: first_countable_topology)"*
  ⟨*proof*⟩

interpretation *at_within: sequential_filter*
  "at (x :: 'a :: first_countable_topology) within A"
  ⟨*proof*⟩

interpretation *at_top: sequential_filter "at_top :: 'a :: seq_at_top filter"*
  ⟨*proof*⟩

interpretation *at_bot: sequential_filter "at_bot :: 'a :: seq_at_bot filter"*
  ⟨*proof*⟩

interpretation *at_infinity: sequential_filter at_infinity*
⟨*proof*⟩

2.4 Set-wise monotone convergence

We now introduce the notion of a family of sets $A(x)$ converging to another set B "from the inside" as $x \to F$, in the sense that eventually $A(x) \subseteq B$ as $x \to F$ and, for every y, eventually $y \in A(x) \longleftrightarrow y \in B$ as $x \to F$.

That is, $A(x)$ converges to B pointwise from within B in the discrete topology. Typical examples include that e.g. $A(x) = [l(x), r(x)]$ converges to (a, b) if $l(x) \to a^+$ and $r(x) \to b^-$, or $A(x) = [l(x), r(x)]$ converges to $\mathbb{R}$ if $l(x) \to -\infty$ and $r(x) \to \infty$.

definition *tendsto_set :: "'b measure ⇒ ('a ⇒ 'b set) ⇒ 'b set ⇒ 'a filter ⇒ bool"* **where**
"tendsto_set M A B F ⟷
(∃C. C ∈ null_sets M ∧ (∀x. x ∉ C ⟶ eventually (λy. x ∈ A y ⟷ x ∈ B) F))"

named_theorems *tendsto_set_intros*

lemma *tendsto_set_null_sets_transfer:*
assumes *"tendsto_set M f A F" "sym_diff A B ∈ null_sets M"*
shows *"tendsto_set M f B F"*
⟨*proof*⟩

lemma *tendsto_set_cong:*
assumes *"null_sets M = null_sets N" "eventually (λx. f x = g x) F"*
"sym_diff A B ∈ null_sets M" "F = F'"
shows *"tendsto_set M f A F ⟷ tendsto_set N g B F'"*
⟨*proof*⟩

lemma *tendsto_set_Icc_Icc [tendsto_set_intros]:*
fixes *a b :: "'a :: linorder_topology"*
assumes *lim: "filterlim l (nhds a) F" "filterlim r (nhds b) F"*
assumes *"{a, b} ∈ null_sets M"*
shows *"tendsto_set M (λx. {l x..r x}) {a..b} F"*
⟨*proof*⟩

lemma *tendsto_set_Icc_Ici [tendsto_set_intros]:*
fixes *a :: "'a :: linorder_topology"*
assumes *lim: "filterlim l (nhds a) F" "filterlim r at_top F"*
assumes *"{a} ∈ null_sets M"*
shows *"tendsto_set M (λx. {l x..r x}) {a..} F"*
⟨*proof*⟩

lemma *tendsto_set_Icc_Iic [tendsto_set_intros]:*
fixes *b :: "'a :: linorder_topology"*
assumes *lim: "filterlim l at_bot F" "filterlim r (nhds b) F"*

assumes *"{b} ∈ null_sets M"*
shows *"tendsto_set M (λx. {l x..r x}) {..b} F"*
⟨*proof*⟩

lemma *tendsto_set_Icc_UNIV [tendsto_set_intros]:*
fixes *l r :: "_ ⇒ 'a :: linorder_topology"*
assumes *lim: "filterlim l at_bot F" "filterlim r at_top F"*
shows *"tendsto_set M (λx. {l x..r x}) UNIV F"*
⟨*proof*⟩

lemma *tendsto_set_Ici_Ici [tendsto_set_intros]:*
fixes *a :: "'a :: linorder_topology"*
assumes *lim: "filterlim l (nhds a) F"*
assumes *"{a} ∈ null_sets M"*
shows *"tendsto_set M (λx. {l x..}) {a..} F"*
⟨*proof*⟩

lemma *tendsto_set_Iic_Iic [tendsto_set_intros]:*
fixes *b :: "'a :: linorder_topology"*
assumes *lim: "filterlim r (nhds b) F"*
assumes *"{b} ∈ null_sets M"*
shows *"tendsto_set M (λx. {..r x}) {..b} F"*
⟨*proof*⟩

lemma *tendsto_set_Ici_UNIV [tendsto_set_intros]:*
fixes *l :: "_ ⇒ 'a :: linorder_topology"*
assumes *lim: "filterlim l at_bot F"*
shows *"tendsto_set M (λx. {l x..}) UNIV F"*
⟨*proof*⟩

lemma *tendsto_set_Iic_UNIV [tendsto_set_intros]:*
fixes *r :: "_ ⇒ 'a :: linorder_topology"*
assumes *lim: "filterlim r at_top F"*
shows *"tendsto_set M (λx. {..r x}) UNIV F"*
⟨*proof*⟩

The next version of the lemma is slightly stronger in the sense that we can also show $\lim_{x \to a^-} \int_0^x = \int_0^a$ (i.e. the endpoint is also included).

lemma (in *sequential_filter) filterlim_set_lebesgue_integral_set:*
fixes *f :: "real ⇒ 'b::{banach, second_countable_topology}"*
assumes *integrable: "set_integrable M B f"*
assumes *measurable: "set_borel_measurable M A f" "eventually (λx. set_borel_measurable M (X x) f) F"*
assumes *X: "tendsto_set M X A F" "eventually (λx. X x ∈ sets M) F"*
assumes *subset: "eventually (λx. X x ⊆ B) F"*
shows *"((λx. set_lebesgue_integral M (X x) f) ⟶ set_lebesgue_integral M A f) F"*
⟨*proof*⟩

end

3 Integral forms of the Beta function

theory *More_Beta*
  imports *"HOL-Complex_Analysis.Complex_Analysis" More_Dominated_Convergence*
begin

lemma *Gamma_legendre_duplication_real:*
  fixes *z :: real*
  assumes *"z ∉ ℤ≤0" "z + 1/2 ∉ ℤ≤0"*
  shows *"Gamma z * Gamma (z + 1/2) = 2 powr (1 - 2 * z) * sqrt pi * Gamma (2*z)"*
⟨*proof*⟩

lemma *Gamma_int_plus_half:*
  *"Gamma (real n + 1 / 2) = sqrt pi * fact (2 * n) / (4 ^ n * fact n)"*
⟨*proof*⟩

lemma *has_field_derivative_complex_powr_right:*
  *"w ≠ 0 ⟹ ((λz. w powr z) has_field_derivative Ln w * w powr z) (at z within A)"*
  ⟨*proof*⟩

lemmas *has_field_derivative_complex_powr_right' =*
  has_field_derivative_complex_powr_right[THEN DERIV_chain2]

lemma *uniform_limit_set_lebesgue_integral:*
  fixes *f :: "'a ⇒ 'b :: euclidean_space ⇒ 'c :: {banach, second_countable_topology}"*
  assumes *"set_integrable lborel X' g"*
  assumes *[measurable]: "X' ∈ sets borel"*
  assumes *[measurable]: "⋀y. y ∈ Y ⟹ set_borel_measurable borel X' (f y)"*
  assumes *"⋀y. y ∈ Y ⟹ (AE t∈X' in lborel. norm (f y t) ≤ g t)"*
  assumes *"eventually (λx. X x ∈ sets borel ∧ X x ⊆ X') F"*
  assumes *"filterlim (λx. set_lebesgue_integral lborel (X x) g)*
(nhds (set_lebesgue_integral lborel X' g)) F"
  shows *"uniform_limit Y*
(λx y. set_lebesgue_integral lborel (X x) (f y))
(λy. set_lebesgue_integral lborel X' (f y)) F"
⟨*proof*⟩

lemma *Beta_nonneg_real:*

assumes *"a > 0" "b > 0"*
shows *"Beta a (b::real) ≥ 0"*
⟨*proof*⟩

The Beta function is given by the following integral:

$$B(a,b) = \int_0^1 t^{a-1}(1-t)^{b-1}\,\mathrm{d}t$$

lemma *has_integral_Beta_complex:*
assumes *a: "Re a > 0"* **and** *b: "Re b > 0"*
shows *"((λt. of_real t powr (a - 1) * of_real (1 - t) powr (b - 1))*

has_integral Beta a b) {0<..<1}"
and *"(λt. of_real t powr (a - 1) * of_real (1 - t) powr (b - 1))*
absolutely_integrable_on {0<..<1}"
⟨*proof*⟩

By change of variables, we can also derive the following integral:

$$\frac{1}{2}B(a,b) = \int_0^{\frac{\pi}{2}} \sin^{2a-1} t \cos^{2b-1} t\,\mathrm{d}t$$

lemma *has_integral_Beta_sin_cos_complex:*
assumes *"Re a > 0" "Re b > 0"*
shows *"(λt. of_real (sin t) powr (2*a-1) * of_real (cos t) powr (2*b-1))*
absolutely_integrable_on {0..pi/2}" (**is** *"?thesis1"*)
and *"((λt. of_real (sin t) powr (2*a-1) * of_real (cos t) powr (2*b-1))*
has_integral (Beta a b / 2)) {0..pi/2}" (**is** *"?thesis2"*)
⟨*proof*⟩

lemma *has_integral_Beta_sin_cos_real:*
assumes *"a > 0" "b > 0"*
shows *"((λt. sin t powr (2*a-1) * cos t powr (2*b-1)) has_integral*
(Beta a b / 2)) {0..pi/2}"
⟨*proof*⟩

lemma *sin_power_integral_0_pi_half:*
"((λt. sin t ^ n) has_integral (Beta ((real n + 1) / 2) (1/2)) / 2)
{0..pi/2}"
⟨*proof*⟩

lemma *cos_power_integral_0_pi_half:*
"((λt. cos t ^ n) has_integral (Beta (1/2) ((real n + 1) / 2)) / 2)
{0..pi/2}"
⟨*proof*⟩

lemma *sin_power_even_integral_0_pi_half_real:*
*"((λt. sin t ^ (2*n)) has_integral (pi / 2 * fact (2 * n) / (fact n*
*^ 2 * 4 ^ n))) {0..pi/2}"*

⟨*proof*⟩

lemma *cos_power_even_integral_0_pi_half_real:*
*"((λt. cos t ^ (2*n)) has_integral (pi / 2 * fact (2 * n) / (fact n ^ 2 * 4 ^ n))) {0..pi/2}"*
⟨*proof*⟩

Lastly, we derive the following integral:

$$B(a,b) = \int_0^\infty \frac{t^{a-1}}{(1+t)^{a+b}}\,\mathrm{d}t$$

lemma *has_integral_Beta_0_infinity_complex:*
assumes *"Re a > 0" "Re b > 0"*
shows *"(λt. of_real t powr (a - 1) / of_real (1 + t) powr (a + b)) absolutely_integrable_on {0<..}"* (**is** *"?thesis1"*)
and *"((λt. of_real t powr (a - 1) / of_real (1 + t) powr (a + b)) has_integral (Beta a b)) {0<..}"* (**is** *"?thesis2"*)
⟨*proof*⟩

lemma *has_integral_Beta_0_infinity_real:*
assumes *"a > 0" "b > (0::real)"*
shows *"((λt. t powr (a - 1) / (1 + t) powr (a + b)) has_integral (Beta a b)) {0<..}"*
⟨*proof*⟩

end

4 A proof method for computing derivatives

theory *Derivative_Method*
imports *Complex_Main*
begin

⟨*ML*⟩

end

5 Auxiliary material

theory *Incomplete_Gamma_Lemma_Bucket*
imports
"HOL-Complex_Analysis.Complex_Analysis"
Safe_Power
More_Dominated_Convergence
Derivative_Method

begin

lemma *sgn_sqrt [simp]: "sgn (sqrt x) = sgn x"*
⟨*proof*⟩

lemma *countably_generated_imp_convergent_sequence:*
assumes *"countably_generated_filter F" "F ≠ bot"*
obtains *X* **where** *"filterlim X F sequentially"*
⟨*proof*⟩

lemma **(in** *sequential_filter)* *dominated_convergence':*
fixes *f :: "'a ⇒ 'n::euclidean_space ⇒ 'm::euclidean_space"*
assumes *f: "eventually (λk. (f k) integrable_on S) F"* **and** *h: "h integrable_on S"*
and *le: "eventually (λk. ∀x∈S. norm (f k x) ≤ h x) F"*
and *conv: "⋀x. x ∈ S ⟹ ((λk. f k x) ⟶ g x) F"*
shows *"((λk. integral S (f k)) ⟶ integral S g) F"*
⟨*proof*⟩

lemma *dominated_convergence_countably_generated_filter:*
fixes *f :: "'a ⇒ 'n::euclidean_space ⇒ 'm::euclidean_space"*
assumes *cg: "countably_generated_filter F"*
assumes *f: "eventually (λk. (f k) integrable_on S) F"* **and** *h: "h integrable_on S"*
and *le: "eventually (λk. ∀x∈S. norm (f k x) ≤ h x) F"*
and *conv: "⋀x. x ∈ S ⟹ ((λk. f k x) ⟶ g x) F"*
shows *"F ≠ bot ⟹ g integrable_on S" "((λk. integral S (f k)) ⟶ integral S g) F"*
⟨*proof*⟩

lemma *nonpos_Reals_real_eq: "*$\mathbb{R}_{\leq 0}$ *= {..(0::real)}"*
⟨*proof*⟩

lemma *has_integral_complex_of_real_iff:*
"((λx. of_real (f x) :: complex) has_integral (of_real I)) A ⟷ (f has_integral I) A"
⟨*proof*⟩

lemma *integrable_on_complex_of_real_iff:*
"(λx. of_real (f x) :: complex) integrable_on A ⟷ f integrable_on A"

⟨*proof*⟩

lemma *integral_complex_of_real:*
"integral A (λx. complex_of_real (f x)) = of_real (integral A f)"
⟨*proof*⟩

lemma *countable_nonpos_Ints [intro]: "countable* $\mathbb{Z}_{\leq 0}$*"*
⟨*proof*⟩

lemma
fixes *f :: "_ ⇒ _ ⇒ 'a :: {banach, second_countable_topology}"*
assumes *integrable: "⋀i. set_integrable M A (f i)"*
and *summable: "AE x∈A in M. summable (λi. norm (f i x))"*
and *sums: "summable (λi. (∫ x∈A. norm (f i x) ∂M))"*
shows *set_integrable_suminf: "set_integrable M A (λx. (∑ i. f i x))"*
and *sums_set_integral: "(λi. set_lebesgue_integral M A (f i)) sums (∫ x∈A. (∑ i. f i x) ∂M)"*
and *set_integral_suminf: "(∫ x∈A. (∑ i. f i x) ∂M) = (∑ i. set_lebesgue_integral M A (f i))"*
and *summable_set_integral: "summable (λi. set_lebesgue_integral M A (f i))"*
⟨*proof*⟩

lemma *leibniz_rule_field_derivative_real:*
fixes *f::"'a::{real_normed_field, banach} ⇒ real ⇒ 'a"*
assumes *fx: "⋀x t. x ∈ U ⟹ t ∈ {a..b} ⟹ ((λx. f x t) has_field_derivative fx x t) (at x within U)"*
assumes *integrable_f2: "⋀x. x ∈ U ⟹ (f x) integrable_on {a..b}"*
assumes *cont_fx: "continuous_on (U × {a..b}) (λ(x, t). fx x t)"*
assumes *U: "x0 ∈ U" "convex U"*
shows *"((λx. integral {a..b} (f x)) has_field_derivative integral {a..b} (fx x0)) (at x0 within U)"*
⟨*proof*⟩

lemma *convex_minkowski_sum_ball:*
fixes *A :: "'a :: euclidean_space set"*
assumes *"convex A"*
shows *"convex (⋃ x∈A. ball x r)"*
⟨*proof*⟩

lemma *convex_minkowski_sum_cball:*
fixes *A :: "'a :: euclidean_space set"*
assumes *"convex A"*
shows *"convex (⋃ x∈A. cball x r)"*

⟨*proof*⟩

lemma *continuous_on_linepath [continuous_intros]:*
assumes *"continuous_on A f" "continuous_on A g" "continuous_on A h"*
shows *"continuous_on A (λx. linepath (f x) (g x) (h x))"*
⟨*proof*⟩

lemma *contour_integral_linepath_has_field_derivative:*
assumes *A: "open A" "a ∈ A" "z ∈ A" "closed_segment a z ⊆ A"*
assumes *holo: "f holomorphic_on A"*
shows *"((λz. contour_integral (linepath a z) f) has_field_derivative f z) (at z within B)"*
⟨*proof*⟩

lemma *set_integrable_bigo:*
fixes *f g :: "real ⇒ 'a :: {banach, real_normed_field, second_countable_topology}"*
assumes *"f ∈ O(λx. g x)"* **and** *"set_integrable lborel {a..} g"*
assumes *"⋀b. b ≥ a ⟹ set_integrable lborel {a..<b} f"*
assumes *[measurable]: "set_borel_measurable borel {a..} f"*
shows *"set_integrable lborel {a..} f"*
⟨*proof*⟩

5.1 Miscellaneous preliminary material

lemma *uniform_limit_analytic_at:*
assumes *"uniform_limit A f g F"*
assumes *"eventually (λx. f x holomorphic_on A) F"*
assumes *"z ∈ A" "open A" "F ≠ bot"*
shows *"g analytic_on {z}"*
⟨*proof*⟩

lemma *uniform_limit_holomorphic:*
assumes *"uniform_limit A f g F"*
assumes *"eventually (λx. f x holomorphic_on A) F"*
assumes *"open A" "F ≠ bot"*
shows *"g holomorphic_on A"*
⟨*proof*⟩

lemma *Re_euler_mascheroni [simp]: "Re euler_mascheroni = euler_mascheroni"*
and *Im_euler_mascheroni [simp]: "Im euler_mascheroni = 0"*
⟨*proof*⟩

lemma *has_real_derivative_imp_has_vector_derivative [derivative_intros]:*
assumes *"(f has_real_derivative f') (at x)"*

shows *"(f has_vector_derivative f') (at x)"*
⟨*proof*⟩

lemma *DERIV_real_sqrt_generic':*
assumes *"x ≠ 0"*
shows *"(sqrt has_field_derivative (sgn x * inverse (sqrt x) / 2)) (at x within A)"*
⟨*proof*⟩

lemma *DERIV_real_root_generic':*
assumes *"x ≠ 0"*
shows *"(root n has_real_derivative (inverse (real n * (sgn x * root n x) ^ (n - Suc 0)))) (at x within A)"*
⟨*proof*⟩

declare
DERIV_real_sqrt_generic[THEN DERIV_chain2, derivative_intros del]
DERIV_real_root_generic[THEN DERIV_chain2, derivative_intros del]
DERIV_real_sqrt_generic'[THEN DERIV_chain2, derivative_intros]
DERIV_real_root_generic'[THEN DERIV_chain2, derivative_intros]

lemmas *[derivative_intros] = has_vector_derivative_real_field*

lemma *DERIV_complex_norm [derivative_intros]:*
fixes *f :: "real ⇒ complex"*
assumes *"f x ≠ 0"* **and** *[derivative_intros]: "(f has_vector_derivative D') (at x)"*
shows *"((λx. norm (f x)) has_field_derivative ((D' · f x) / norm (f x))) (at x)"*
⟨*proof*⟩

5.2 Preliminary facts about the Gamma and Digamma function

lemma *cnj_Digamma:*
assumes *[simp]: "s ≠ 0"*
shows *"cnj (Digamma s) = Digamma (cnj s)"*
⟨*proof*⟩

lemma *cnj_Polygamma:*
assumes *[simp]: "s ≠ 0"*
shows *"cnj (Polygamma n s) = Polygamma n (cnj s)"*
⟨*proof*⟩

lemma *norm_rGamma_Im_mono:*

assumes *"Re z1 = Re z2" "|Im z1| ≤ |Im z2|"*
shows *"norm (rGamma z1) ≤ norm (rGamma z2)"*
⟨*proof*⟩

lemma *Re_Digamma_Im_mono:*
assumes *"Re z1 = Re z2" "Re z1 > 0" "|Im z1| ≤ |Im z2|"*
shows *"Re (Digamma z1) ≤ Re (Digamma z2)"*
⟨*proof*⟩

lemma *Digamma_Re_le_Re_Digamma:*
assumes *"Re z > 0"*
shows *"Digamma (Re z) ≤ Re (Digamma z)"*
⟨*proof*⟩

lemma *Re_Digamma_nonneg:*
assumes *"Re z > 0" "Digamma (Re z) ≥ 0"*
shows *"Re (Digamma z) ≥ 0"*
⟨*proof*⟩

lemma *norm_Gamma_Re_mono:*
assumes *"Im z1 = Im z2" "Re z1 ≤ Re z2" "Re z1 > 0" "Digamma (Re z1) ≥ 0"*
shows *"norm (Gamma z1) ≤ norm (Gamma z2)"*
⟨*proof*⟩

lemma *norm_Gamma_Im_mono:*
assumes *"Re z1 = Re z2" "|Im z1| ≤ |Im z2|" "z1 ∈ ℤ≤0 ⟶ z2 ∈ ℤ≤0"*
shows *"norm (Gamma z1) ≥ norm (Gamma z2)"*
⟨*proof*⟩

lemma *norm_Gamma_bound:*
assumes *"Re z ∉ ℤ≤0"*
shows *"norm (Gamma z) ≤ |Gamma (Re z)|"*
⟨*proof*⟩

lemma *norm_Gamma_bound':*
assumes *"Re z > 0"*
shows *"norm (Gamma z) ≤ Gamma (Re z)"*
⟨*proof*⟩

end

6 The Incomplete Gamma Function

theory *Incomplete_Gamma*
imports
"HOL-Complex_Analysis.Complex_Analysis" "HOL-Library.Going_To_Filter"
Generalized_Hypergeometric_Series.Generalized_Hypergeometric_Series
Safe_Power More_Beta More_Dominated_Convergence

```
  Derivative_Method Incomplete_Gamma_Lemma_Bucket
begin
```

6.1 Construction of the auxiliary entire function

For an overview of the functions we will define, see §8.2 of the NIST Digital Library of Mathematical Functions [1].

We first use the regularised hypergeometric series to define a version of the regularised lower gamma function that is entire in both variables and therefore particularly pleasant to handle. The NIST DLMF calls this function $\gamma^*(s, z)$.

```
definition pre_Gamma_rincl :: "'a :: Gamma ⇒ 'a ⇒ 'a" where
  "pre_Gamma_rincl s z = exp (-z) * reg_hypergeo_F [1] [1+s] z"

lemma pre_Gamma_rincl_complex_of_real:
  "pre_Gamma_rincl (complex_of_real s) (of_real z) = of_real (pre_Gamma_rincl
s z)"
  ⟨proof⟩

lemma pre_Gamma_rincl_0_left [simp]: "pre_Gamma_rincl 0 z = 1"
  ⟨proof⟩

lemma pre_Gamma_rincl_0_right [simp]: "pre_Gamma_rincl s 0 = rGamma (s
+ 1)"
  ⟨proof⟩

lemma pre_Gamma_rincl_1_left:
  assumes  [simp]: "z ≠ 0"
  shows    "pre_Gamma_rincl 1 z = (1 - exp (-z)) / z"
⟨proof⟩

lemma analytic_pre_Gamma_rincl [analytic_intros]:
  assumes [analytic_intros]: "s analytic_on X" "z analytic_on X"
  shows   "(λx. pre_Gamma_rincl (s x) (z x)) analytic_on X"
  ⟨proof⟩

lemma continuous_on_pre_Gamma_rincl [continuous_intros]:
  fixes s z :: "_ ⇒ 'a :: {Gamma, heine_borel}"
  assumes [continuous_intros]: "continuous_on X s" "continuous_on X z"
  shows   "continuous_on X (λx. pre_Gamma_rincl (s x) (z x))"
⟨proof⟩

lemma tendsto_pre_Gamma_rincl [tendsto_intros]:
  fixes s z :: "'a :: {Gamma, heine_borel}"
  assumes "(f ⟶ s) F" "(g ⟶ z) F"
  shows   "((λx. pre_Gamma_rincl (f x) (g x)) ⟶ pre_Gamma_rincl s
z) F"
⟨proof⟩
```

lemma *continuous_pre_Gamma_rincl [continuous_intros]:*
fixes *s z :: "_ ⇒ 'a :: {Gamma, heine_borel}"*
assumes *[continuous_intros]: "continuous (at x within A) s" "continuous (at x within A) z"*
shows *"continuous (at x within A) (λx. pre_Gamma_rincl (s x) (z x))"*
⟨*proof*⟩

lemma *sums_pre_Gamma_rincl:*
*"(λn. exp (-z) * rGamma (s + of_nat n + 1) * z ^ n) sums (pre_Gamma_rincl s z)"*
⟨*proof*⟩

lemma *erf_conv_pre_Gamma_rincl_complex:*
*"erf (z :: complex) = z * pre_Gamma_rincl (1 / 2) (z²)"*
⟨*proof*⟩

lemma *erf_conv_pre_Gamma_rincl_real:*
*"erf (z :: real) = z * pre_Gamma_rincl (1 / 2) (z²)"*
⟨*proof*⟩

6.2 The regularised lower Gamma function

We now add the factor z^s, which contributes a branch cut, to obtain the regularised lower gamma function $P(s,z)$ (again using the NIST DLMF notation).

This is essentially $\gamma(s,z)/\Gamma(s)$, but with the benefit that it is defined even when $\gamma(s,z)$ and $\Gamma(s)$ are not (i.e. when s is a non-negative integer).

definition *Gamma_rincl :: "'a :: {Gamma, ln} ⇒ 'a ⇒ 'a"* **where**
*"Gamma_rincl s z = z powr' s * pre_Gamma_rincl s z"*

lemma *Gamma_rincl_0_left [simp]: "Gamma_rincl 0 z = 1"*
⟨*proof*⟩

lemma *Gamma_rincl_0_right [simp]: "s ≠ 0 ⟹ Gamma_rincl s 0 = 0"*
⟨*proof*⟩

lemma *Gamma_rincl_1_left: "Gamma_rincl 1 z = 1 - exp (-z)"*
⟨*proof*⟩

lemma *Gamma_rincl_complex_of_real:*
assumes *"s ∉ ℤ ⟹ 0 ≤ z"*
shows *"Gamma_rincl (complex_of_real s) (of_real z) = of_real (Gamma_rincl s z)"*
⟨*proof*⟩

$P(s,z)$ is holomorphic in s and z apart from a branch cut along the negative

real axis for z. Alternatively, if s is a fixed integer, $P(s,z)$ is entire in z if $s \geq 0$ and meromorphic with a pole at $z = 0$ if $s < 0$.

lemma *analytic_Gamma_rincl [analytic_intros]:*
assumes *[analytic_intros]: "s analytic_on X" "z analytic_on X"*
assumes *"⋀x. x ∈ X ⟹ z x ∉ ℝ≤0"*
shows *"(λx. Gamma_rincl (s x) (z x)) analytic_on X"*
⟨*proof*⟩

lemma *analytic_Gamma_rincl':*
assumes *[analytic_intros]: "f analytic_on X"*
assumes *"⋀x. x ∈ X ⟹ s ∈ (ℤ≤0 - {0}) ⟹ f x ≠ 0"*
assumes *"⋀x. x ∈ X ⟹ s ∉ ℤ ⟹ f x ∉ ℝ≤0"*
shows *"(λx. Gamma_rincl s (f x)) analytic_on X"*
⟨*proof*⟩

$P(s,z)$ is continuous away from the nonpositive reals. It is additionally also continuous at $z = 0$ if $\mathrm{Re}(s) > 0$.

lemma *continuous_on_Gamma_rincl_complex [continuous_intros]:*
fixes *x z :: "'a :: topological_space ⇒ complex"*
assumes *[continuous_intros]: "continuous_on X s" "continuous_on X z"*
assumes *"⋀x. x ∈ X ⟹ Re (z x) ≥ 0 ∨ Im (z x) ≠ 0"*
assumes *"⋀x. x ∈ X ⟹ z x = 0 ⟹ Re (s x) > 0"*
shows *"continuous_on X (λx. Gamma_rincl (s x) (z x))"*
⟨*proof*⟩

lemma *continuous_on_Gamma_rincl_real [continuous_intros]:*
fixes *x z :: "'a :: topological_space ⇒ real"*
assumes *[continuous_intros]: "continuous_on X s" "continuous_on X z"*
assumes *"⋀x. x ∈ X ⟹ z x ≥ 0"*
assumes *"⋀x. x ∈ X ⟹ z x = 0 ⟹ s x > 0"*
shows *"continuous_on X (λx. Gamma_rincl (s x) (z x))"*
⟨*proof*⟩

lemma *tendsto_Gamma_rincl_complex [tendsto_intros]:*
fixes *s z :: complex*
assumes *"(f ⟶ s) F" "(g ⟶ z) F" "z ∉ ℝ≤0 ∨ (z = 0 ∧ Re s > 0)"*
shows *"((λx. Gamma_rincl (f x) (g x)) ⟶ Gamma_rincl s z) F"*
⟨*proof*⟩

lemma *tendsto_Gamma_rincl_real [tendsto_intros]:*
fixes *s z :: real*
assumes *"(f ⟶ s) F" "(g ⟶ z) F" "z > 0 ∨ (z = 0 ∧ s > 0)"*
shows *"((λx. Gamma_rincl (f x) (g x)) ⟶ Gamma_rincl s z) F"*
⟨*proof*⟩

lemma *continuous_Gamma_rincl_complex [continuous_intros]:*
fixes *s z :: "_ ⇒ complex"*

assumes *"continuous (at x within A) s" "continuous (at x within A) z"*
assumes *"z x ∉ ℝ≤0 ∨ (z x = 0 ∧ Re (s x) > 0)"*
shows *"continuous (at x within A) (λx. Gamma_rincl (s x) (z x))"*
⟨*proof*⟩

lemma *continuous_Gamma_rincl_real [continuous_intros]:*
fixes *s z :: "_ ⇒ real"*
assumes *"continuous (at x within A) s" "continuous (at x within A) z"*
assumes *"z x > 0 ∨ (z x = 0 ∧ s x > 0)"*
shows *"continuous (at x within A) (λx. Gamma_rincl (s x) (z x))"*
⟨*proof*⟩

Writing $P(s, z)$ as a series:

lemma *sums_Gamma_rincl:*
*"(λn. z powr' s * z ^ n * exp (-z) * rGamma (s + of_nat n + 1)) sums (Gamma_rincl s z)"*
⟨*proof*⟩

lemma *sums_Gamma_rincl_complex:*
assumes *"z ≠ 0 ∨ s ∉ ℤ≤0"*
shows *"(λn. z powr (s + of_nat n) * exp (-z) * rGamma (s + of_nat n + 1 :: complex))*
sums (Gamma_rincl s z)"
⟨*proof*⟩

lemma *sums_Gamma_rincl_real:*
assumes *"z > 0 ∨ s ∈ ℤ - ℤ≤0"*
shows *"(λn. z powr' (s + of_nat n) * exp (-z) * rGamma (s + of_nat n + 1 :: real))*
sums (Gamma_rincl s z)"
⟨*proof*⟩

The recurrence relation for $P(s, z)$:

lemma *Gamma_rincl_plus1_complex:*
assumes *"s ≠ -1"*
shows *"Gamma_rincl (s+1) z = Gamma_rincl s z - z powr' s * exp (-z) * rGamma (s+1 :: complex)"*
⟨*proof*⟩

lemma *Gamma_rincl_plus1_real:*
assumes *"z ≥ 0 ∨ s ∈ ℤ" "s ≠ -1"*
shows *"Gamma_rincl (s+1) z = Gamma_rincl s z - z powr' s * exp (-z) * rGamma (s+1 :: real)"*
⟨*proof*⟩

We can now also easily show a closed form for the case where s is a positive integer:

lemma *Gamma_rincl_of_nat_left_complex:*
fixes *z :: complex*
shows *"Gamma_rincl (of_nat (Suc n)) z = 1 - exp (-z) * (∑k≤n. z ^ k / fact k)"*
⟨*proof*⟩

lemma *Gamma_rincl_of_nat_left_real:*
fixes *z :: real*
shows *"Gamma_rincl (of_nat (Suc n)) z = 1 - exp (-z) * (∑k≤n. z ^ k / fact k)"*
⟨*proof*⟩

Lastly, when $s = \frac{1}{2}$, the reduced lower incomplete Gamma function is related to the error function (via their hypergeometric representations):

lemma *erf_conv_Gamma_rincl_real: "erf z = sgn z * Gamma_rincl (1/2) (z ^ 2 :: real)"*
⟨*proof*⟩

lemma *Gamma_rincl_one_half_left_real:*
assumes *"z ≥ 0"*
shows *"Gamma_rincl (1/2) (z :: real) = erf (sqrt z)"*
⟨*proof*⟩

lemma *Gamma_rincl_one_half_left_complex:*
assumes *z: "z ∉ ℝ≤0"*
shows *"Gamma_rincl (1/2) (z :: complex) = erf (csqrt z)"*
⟨*proof*⟩

6.3 The lower incomplete Gamma function

definition *Gamma_incl :: "'a :: {Gamma, ln} ⇒ 'a ⇒ 'a"* **where**
*"Gamma_incl s z = Gamma s * Gamma_rincl s z"*

lemma *Gamma_incl_complex_of_real:*
assumes *"s ∉ ℤ ⟹ 0 ≤ z"*
shows *"Gamma_incl (complex_of_real s) (of_real z) = of_real (Gamma_incl s z)"*
⟨*proof*⟩

The lower incomplete Gamma function $\gamma(s, z)$ is analytic away from $z \leq 0$ and $s \in \{0, -1, -2, \ldots\}$.

lemma *analytic_Gamma_incl [analytic_intros]:*
assumes *[analytic_intros]: "s analytic_on X" "z analytic_on X"*
assumes *"⋀x. x ∈ X ⟹ z x ∉ ℝ≤0" "⋀x. x ∈ X ⟹ s x ∉ ℤ≤0"*
shows *"(λx. Gamma_incl (s x) (z x)) analytic_on X"*
⟨*proof*⟩

When s is a positive integer, $\gamma(s, z)$ is entire in z.

lemma *analytic_Gamma_incl_nonneg_int [analytic_intros]:*

assumes *[analytic_intros]: "z analytic_on X"* **and** *"n ≥ 0"*
shows *"(λx. Gamma_incl (of_int n) (z x)) analytic_on X"*
⟨*proof*⟩

lemma *analytic_Gamma_incl':*
assumes *[analytic_intros]: "f analytic_on X"*
assumes *"⋀x. x ∈ X ⟹ s ∈ ℤ≤0 ⟹ f x ≠ 0"*
assumes *"⋀x. x ∈ X ⟹ s ∉ ℤ ⟹ f x ∉ ℝ≤0"*
shows *"(λx. Gamma_incl s (f x)) analytic_on X"*
⟨*proof*⟩

lemma *continuous_on_Gamma_incl_complex [continuous_intros]:*
fixes *x z :: "'a :: topological_space ⇒ complex"*
assumes *[continuous_intros]: "continuous_on X s" "continuous_on X z"*
assumes *"⋀x. x ∈ X ⟹ Re (z x) ≥ 0 ∨ Im (z x) ≠ 0"*
assumes *"⋀x. x ∈ X ⟹ z x = 0 ⟹ Re (s x) > 0"*
assumes *"⋀x. x ∈ X ⟹ s x ∉ ℤ≤0"*
shows *"continuous_on X (λx. Gamma_incl (s x) (z x))"*
⟨*proof*⟩

lemma *continuous_on_Gamma_incl_real [continuous_intros]:*
fixes *x z :: "'a :: topological_space ⇒ real"*
assumes *[continuous_intros]: "continuous_on X s" "continuous_on X z"*
assumes *"⋀x. x ∈ X ⟹ z x ≥ 0"*
assumes *"⋀x. x ∈ X ⟹ z x = 0 ⟹ s x > 0"*
assumes *"⋀x. x ∈ X ⟹ s x ∉ ℤ≤0"*
shows *"continuous_on X (λx. Gamma_incl (s x) (z x))"*
⟨*proof*⟩

lemma *tendsto_Gamma_incl_complex [tendsto_intros]:*
fixes *s z :: complex*
assumes *"(f ⟶ s) F" "(g ⟶ z) F"*
assumes *"Re z ≥ 0 ∨ Im z ≠ 0" "z = 0 ⟹ Re s > 0" "s ∉ ℤ≤0"*
shows *"((λx. Gamma_incl (f x) (g x)) ⟶ Gamma_incl s z) F"*
thm *Gamma_incl_def*
⟨*proof*⟩

lemma *tendsto_Gamma_incl_real [tendsto_intros]:*
fixes *s z :: real*
assumes *"(f ⟶ s) F" "(g ⟶ z) F" "z ≥ 0" "z = 0 ⟹ s > 0" "s ∉ ℤ≤0"*
shows *"((λx. Gamma_incl (f x) (g x)) ⟶ Gamma_incl s z) F"*
⟨*proof*⟩

lemma *continuous_Gamma_incl_complex [continuous_intros]:*
fixes *s z :: "_ ⇒ complex"*
assumes *"continuous (at x within A) s" "continuous (at x within A) z"*
assumes *"Re (z x) ≥ 0 ∨ Im (z x) ≠ 0" "z x = 0 ⟹ Re (s x) > 0" "s*

```
x ∉ ℤ≤0"
  shows   "continuous (at x within A) (λx. Gamma_incl (s x) (z x))"
```
⟨*proof*⟩

lemma *continuous_Gamma_incl_real [continuous_intros]:*
```
  fixes s z :: "_ ⇒ real"
  assumes "continuous (at x within A) s" "continuous (at x within A)
z"
  assumes "z x ≥ 0" "z x = 0 ⟹ s x > 0" "s x ∉ ℤ≤0"
  shows   "continuous (at x within A) (λx. Gamma_incl (s x) (z x))"
```
⟨*proof*⟩

$\gamma(s, z)$ as a series:

lemma *sums_Gamma_incl:*
```
  "(λn. z powr' s * z ^ n * exp (-z) * Gamma s * rGamma (s + of_nat n
+ 1)) sums (Gamma_incl s z)"
```
⟨*proof*⟩

lemma *sums_Gamma_incl_complex:*
```
  "(λn. z powr (s + complex_of_nat n) * exp (-z) * Gamma s * rGamma (s
+ of_nat n + 1))
      sums (Gamma_incl s z)"
```
⟨*proof*⟩

lemma *sums_Gamma_incl_real_nonneg:*
```
  assumes "z ≥ 0"
  shows   "(λn. z powr (s + real n) * exp (-z) * Gamma s * rGamma (s
+ of_nat n + 1))
             sums (Gamma_incl s z)"
```
⟨*proof*⟩

The recurrence for $\gamma(s, z)$:

lemma *Gamma_incl_plus1_complex:*
```
  assumes "s ∉ ℤ≤0"
  shows   "Gamma_incl (s+1) z = s * Gamma_incl s z - z powr' s * exp (-z
:: complex)"
```
⟨*proof*⟩

lemma *Gamma_incl_plus1_real:*
```
  assumes "s ∈ ℤ ∨ z ≥ 0" "s ∉ ℤ≤0"
  shows   "Gamma_incl (s+1) z = s * Gamma_incl s z - z powr' s * exp (-z
:: real)"
```
⟨*proof*⟩

For $\mathrm{Re}(s) > 0$, $\Gamma(s, z)$ has a representation as a contour integral.

theorem *has_contour_integral_Gamma_incl:*
```
  fixes s z :: complex
  assumes s: "Re s > 0"
```

shows *"((λu. u powr (s-1) * exp (-u)) has_contour_integral Gamma_incl s z) (linepath 0 z)"*
⟨*proof*⟩

lemma *has_integral_Gamma_incl_complex_of_real:*
assumes *s: "Re s > (0::real)"*
assumes *"x ≥ 0"*
shows *"((λt. of_real t powr (s - 1) * of_real (exp (-t)))*
has_integral Gamma_incl s (of_real x)) {0..x}"
⟨*proof*⟩

lemma *has_integral_Gamma_incl_complex_of_real':*
assumes *s: "Re s > (0::real)"*
assumes *"x ≤ 0"*
shows *"((λt. of_real t powr (s - 1) * of_real (exp (-t)))*
has_integral (-Gamma_incl s (of_real x))) {x..0}"
⟨*proof*⟩

lemma *has_integral_Gamma_incl_real:*
assumes *s: "s > (0::real)"*
assumes *"x ≥ 0"*
shows *"((λt. t powr (s - 1) * exp (-t)) has_integral Gamma_incl s x) {0..x}"*
⟨*proof*⟩

6.4 The upper incomplete Gamma function

To make the definition work on as big a domain as possible, we do not define the upper incomplete Gamma function $\Gamma(s,z)$ as $\Gamma(s,z) = \Gamma(s) - \gamma(s,z)$, since there are values of s where the left-hand side exists but the two parts on the right-hand side blow up. Rather, we express $\Gamma(s,z)$ as a contour integral starting at z and going to ∞. The precise path does not matter much, so for convenience, we go from z straight to 1 (the first auxiliary function below) and then from 1 straight to ∞.

To make the first definition work for both the *real* and *complex* type, we express the contour integral in a more explicit fashion.

definition *Gamma_incu_aux1 :: "'a :: {banach, real_inner, real_normed_field, ln} ⇒ 'a ⇒ 'a"* **where**
"Gamma_incu_aux1 s z =
(if (z + 1 ∈ ℝ≤0 ∧ (z ≠ -1 ∨ s · 1 ≤ -1)) ∧ s ∉ ℤ then 0 else

*integral {0..1} (λx. (1 + x *R z) powr' s * exp (- (1 + x *R z))))"*

lemma *Gamma_incu_aux1_complex_of_real:*
"Gamma_incu_aux1 (complex_of_real s) (complex_of_real z) = of_real (Gamma_incu_aux1 s z)"
⟨*proof*⟩

lemma *Gamma_incu_aux1_conv_contour_integral:*
assumes *"z ∉ ℝ≤0 ∨ z = 0 ∧ Re s > 0 ∨ s ∈ ℤ"*
shows *"(z-1) * Gamma_incu_aux1 (s-1) (z-1) =*
*contour_integral (linepath 1 z) (λu. u powr' (s - 1) * exp (-u))"*
⟨*proof*⟩

lemma *analytic_Gamma_incu_aux1 [analytic_intros]:*
assumes *"f analytic_on A" "g analytic_on A" "⋀x. x ∈ A ⟹ 1 + g x ∉ ℝ≤0"*
shows *"(λx. Gamma_incu_aux1 (f x) (g x)) analytic_on A"*
⟨*proof*⟩

lemma *analytic_Gamma_incu_aux1_nat [analytic_intros]:*
assumes *"g analytic_on A"*
shows *"(λx. Gamma_incu_aux1 (of_nat n) (g x)) analytic_on A"*
⟨*proof*⟩

lemma *continuous_on_Gamma_incu_aux1_complex':*
fixes *A :: "(complex × complex) set"*
assumes *A: "⋀s z. (s, z) ∈ A ⟹ (Re z ≥ -1 ∨ Im z ≠ 0) ∧ z ≠ -1"*
shows *"continuous_on A (λsz. Gamma_incu_aux1 (fst sz) (snd sz))"*
⟨*proof*⟩

lemma *continuous_on_Gamma_incu_aux1_complex [continuous_intros]:*
assumes *"continuous_on A f" "continuous_on A g"*
assumes *A: "⋀x. x ∈ A ⟹ (Re (g x) ≥ -1 ∨ Im (g x) ≠ 0) ∧ g x ≠ -1"*
shows *"continuous_on A (λx. Gamma_incu_aux1 (f x) (g x :: complex))"*
⟨*proof*⟩

lemma *continuous_Gamma_incu_aux1_at_neg1_aux:*
defines *"A ≡ {(s,z). Re s > -1 ∧ Re z > -1}"*
assumes *"Re s > -1"*
shows *"((λ(s,z). Gamma_incu_aux1 s z) ⟶ Gamma_incu_aux1 s (-1)) (at (s, -1) within A)"*
⟨*proof*⟩

definition *Gamma_incu_aux2 :: "'a :: {banach, real_normed_algebra_1} ⇒ 'a"* **where**
*"Gamma_incu_aux2 s = integral {1..} (λt. exp ((s-1) * of_real (ln t) - of_real t))"*

lemma *Gamma_incu_aux2_complex_of_real:*
"Gamma_incu_aux2 (complex_of_real s) = of_real (Gamma_incu_aux2 s)"

⟨*proof*⟩

lemma *absolutely_integrable_incomplete_Gamma:*
fixes *s :: complex*
assumes *r: "r > 0"*
shows *"(λt. exp (of_real (-t) - s * of_real (ln t))) absolutely_integrable_on {r..}"*
⟨*proof*⟩

lemma *analytic_Gamma_incu_aux2_aux:*
fixes *r :: real* **and** *f :: "complex ⇒ complex"*
assumes *r: "r > 0"*
defines *"f ≡ (λz. integral {r..} (λt. exp (-of_real t - z * of_real (ln t))))"*
shows *"f holomorphic_on UNIV"*
⟨*proof*⟩

lemma *analytic_Gamma_incu_aux2 [analytic_intros]:*
assumes *"f analytic_on A"*
shows *"(λx. Gamma_incu_aux2 (f x)) analytic_on A"*
⟨*proof*⟩

lemma *continuous_on_Gamma_incu_aux2_complex [continuous_intros]:*
assumes *"continuous_on A f"*
shows *"continuous_on A (λx. Gamma_incu_aux2 (f x :: complex))"*
⟨*proof*⟩

lemma *continuous_Gamma_incu_aux2_complex [continuous_intros]:*
assumes *"continuous (at x within A) f"*
shows *"continuous (at x within A) (λx. Gamma_incu_aux2 (f x :: complex))"*
⟨*proof*⟩

Finally, we can define the upper incomplete Gamma function $\Gamma(s, z)$:

definition *Gamma_incu :: "'a :: {banach, real_inner, real_normed_field, ln} ⇒ 'a ⇒ 'a"*
where *"Gamma_incu s z = Gamma_incu_aux2 s - (z - 1) * Gamma_incu_aux1 (s - 1) (z - 1)"*

lemma *Gamma_incu_complex_of_real:*
"Gamma_incu (complex_of_real s) (of_real z) = of_real (Gamma_incu s z)"
⟨*proof*⟩

In general, $\Gamma(s, z)$ is analytic away from $z \leq 0$ (where it has a branch cut).

lemma *analytic_Gamma_incu [analytic_intros]:*
assumes *"f analytic_on A" "g analytic_on A" "⋀x. x ∈ A ⟹ g x ∉ ℝ≤0"*
shows *"(λx. Gamma_incu (f x) (g x)) analytic_on A"*
⟨*proof*⟩

For s a positive integer, $\Gamma(s, z)$ is entire in z:

lemma *analytic_Gamma_incu_pos_int [analytic_intros]:*
assumes *"g analytic_on A" "n > 0"*
shows *"(λx. Gamma_incu (of_int n) (g x)) analytic_on A"*
⟨*proof*⟩

lemma *continuous_on_Gamma_incu_complex [continuous_intros]:*
fixes *x z :: "'a :: topological_space ⇒ complex"*
assumes *[continuous_intros]: "continuous_on X s" "continuous_on X z"*
assumes *"⋀x. x ∈ X ⟹ Re (z x) > 0 ∨ Im (z x) ≠ 0"*
shows *"continuous_on X (λx. Gamma_incu (s x) (z x))"*
⟨*proof*⟩

lemma *tendsto_Gamma_incu_complex [tendsto_intros]:*
fixes *s z :: complex*
assumes *"(f ⟶ s) F" "(g ⟶ z) F" "Re z > 0 ∨ Im z ≠ 0"*
shows *"((λx. Gamma_incu (f x) (g x)) ⟶ Gamma_incu s z) F"*
⟨*proof*⟩

lemma *tendsto_Gamma_incu_real [tendsto_intros]:*
fixes *s z :: real*
assumes *"(f ⟶ s) F" "(g ⟶ z) F" "z > 0"*
shows *"((λx. Gamma_incu (f x) (g x)) ⟶ Gamma_incu s z) F"*
⟨*proof*⟩

lemma *continuous_Gamma_incu_complex [continuous_intros]:*
fixes *s z :: "_ ⇒ complex"*
assumes *"continuous (at x within A) s" "continuous (at x within A) z"*
assumes *"Re (z x) > 0 ∨ Im (z x) ≠ 0"*
shows *"continuous (at x within A) (λx. Gamma_incu (s x) (z x))"*
⟨*proof*⟩

lemma *continuous_Gamma_incu_real [continuous_intros]:*
fixes *s z :: "_ ⇒ real"*
assumes *"continuous (at x within A) s" "continuous (at x within A) z"*
assumes *"z x > 0"*
shows *"continuous (at x within A) (λx. Gamma_incu (s x) (z x))"*
⟨*proof*⟩

The behaviour at the branch point $z = 0$ is also interesting: when approaching from the right (i.e. $\mathrm{Re}(s) > 0$ and $\mathrm{Re}(z) > 0$, the function $\Gamma(s, z)$ converges as $z \to 0$. We will later see that it converges to $\Gamma(s)$.

lemma *continuous_Gamma_incu_0_strong_complex:*
assumes *"Re s > 0"*
defines *"F ≡ (at (s,0) within ({s. Re s > 0} × {z. Re z > 0}))"*
shows *"continuous F (λ(s,z). Gamma_incu s z)"*
⟨*proof*⟩

lemma *continuous_Gamma_incu_0_complex:*
assumes *"Re s > 0"*
shows *"continuous (at 0 within {z. Re z > 0}) (λz. Gamma_incu s z)"*
⟨*proof*⟩

It is also straightforward to show that:

$$\frac{\mathrm{d}}{\mathrm{d}z}\,\Gamma(s,z) = -z^{s-1}\exp(-z)$$

This is, unsurprisingly, the opposite of the derivative of $\gamma(s,z)$.

lemma *has_field_derivative_Gamma_incu_complex:*
assumes *z: "s ∈ (ℤ-ℤ≤0) ∨ (z :: complex) ∉ ℝ≤0"*
shows *"((λz. Gamma_incu s z) has_field_derivative (-(z powr' (s-1)*
** exp (-z)))) (at z within A)"*
⟨*proof*⟩

lemma *has_field_derivative_Gamma_incu_complex' [derivative_intros]:*
assumes *"(f has_field_derivative f') (at z within A)" "s ∈ (ℤ-ℤ≤0)*
∨ (f z :: complex) ∉ ℝ≤0"
shows *"((λz. Gamma_incu s (f z))*
*has_field_derivative (-(f z powr' (s-1) * exp (-f z)) * f'))*
(at z within A)"
⟨*proof*⟩

lemma *has_field_derivative_Gamma_incu_real:*
assumes *"(f has_field_derivative f') (at x within A)" "s ∈ (ℤ-ℤ≤0)*
∨ (f x :: real) > 0"
shows *"((λx. Gamma_incu s (f x)) has_field_derivative*
*(-(f x powr' (s-1) * exp (-f x) * f'))) (at x within A)"*
⟨*proof*⟩

lemma *has_integral_Gamma_incu_complex_of_real':*
assumes *x: "x > (0 :: real)"*
shows *"((λz. exp ((s-1) * of_real (ln z) - complex_of_real z)) has_integral*

(Gamma_incu s (of_real x))) {x..}"
⟨*proof*⟩

lemma *has_integral_Gamma_incu_complex_of_real:*
assumes *x: "x > (0 :: real)"*
shows *"((λt. complex_of_real t powr (s - 1) * of_real (exp (-t)))*

has_integral (Gamma_incu s (of_real x))) {x..}"
⟨*proof*⟩

lemma *has_integral_Gamma_incu_real:*
fixes *x s :: real*

assumes *"x > (0::real)"*
shows *"((λt. t powr (s - 1) * exp (-t)) has_integral Gamma_incu s x) {x..}"*
⟨*proof*⟩

6.5 Identities

All the facts we have collected so far now allow us to prove that $\gamma(s,z) + \Gamma(s,z) = \Gamma(s)$ if $\mathrm{Re}(s) > 0$ (where the integral expressions for γ and Γ are valid). Then we use analytic continuation to lift this result to the rest of the domain.

lemma *Gamma_incl_plus_incu_complex_aux:*
assumes *"s ∉ ℤ≤0" "s - 1 ∈ ℕ ∨ z ∉ ℝ≤0"*
shows *"Gamma_incl s z + Gamma_incu s z = Gamma (s :: complex)"*
⟨*proof*⟩

The recurrence for $\Gamma(s,z)$:

lemma *Gamma_incu_plus1_complex_aux:*
assumes *z: "z ∉ ℝ≤0 ∨ s - 1 ∈ ℕ"*
shows *"Gamma_incu (s+1) z = s * Gamma_incu s z + z powr' s * exp (-z :: complex)"*
⟨*proof*⟩

theorem *Gamma_incu_plus1_complex:*
assumes *z: "z ∉ ℝ≤0 ∨ s - 1 ∈ ℕ ∨ (z = 0 ∧ Re s > 0)"*
shows *"Gamma_incu (s+1) z = s * Gamma_incu s z + z powr' s * exp (-z :: complex)"*
⟨*proof*⟩

hide_fact *Gamma_incu_plus1_complex_aux*

lemma *Gamma_incu_plus1_real:*
assumes *z: "z > 0 ∨ (z = 0 ∧ s > 0)"*
shows *"Gamma_incu (s+1) z = s * Gamma_incu s z + z powr' s * exp (-z :: real)"*
⟨*proof*⟩

theorem *Gamma_incl_plus_incu_complex:*
assumes *"s ∉ ℤ≤0" "z ∉ ℝ≤0 ∨ s - 1 ∈ ℕ ∨ (z = 0 ∧ Re s > 0)"*
shows *"Gamma_incl s z + Gamma_incu s z = Gamma (s :: complex)"*
⟨*proof*⟩

hide_fact *Gamma_incl_plus_incu_complex_aux*

lemma *Gamma_incl_plus_incu_real:*
assumes *"s ∉ ℤ≤0" "z > 0 ∨ (z = 0 ∧ s > 0)"*

shows *"Gamma_incl s z + Gamma_incu s z = Gamma (s :: real)"*
⟨*proof*⟩

6.6 Derivative of $\gamma(s, z)$

Via the relationship with $\Gamma(s)$ and $\Gamma(s, z)$, it is now also straightforward to prove the derivative of $\gamma(s, z)$:

lemma *has_field_derivative_Gamma_incl_complex:*
fixes *s z :: complex*
assumes *"s ∉ ℤ≤0" "s - 1 ∈ ℕ ∨ z ∉ ℝ≤0"*
shows *"((λx. Gamma_incl s x) has_field_derivative (z powr' (s-1) * exp (-z))) (at z within A)"*
⟨*proof*⟩

lemma *has_field_derivative_Gamma_incl_complex' [derivative_intros]:*
fixes *f :: "_ ⇒ complex"*
assumes *"(f has_field_derivative f') (at x within A)" "s ∉ ℤ≤0" "s - 1 ∈ ℕ ∨ f x ∉ ℝ≤0"*
shows *"((λx. Gamma_incl s (f x)) has_field_derivative (f x powr' (s-1) * exp (-f x) * f')) (at x within A)"*
⟨*proof*⟩

lemma *has_field_derivative_Gamma_incl_real [derivative_intros]:*
fixes *f :: "_ ⇒ real"*
assumes *"(f has_field_derivative f') (at x within A)"* **and** *s: "s ∉ ℤ≤0"* **and** *fx: "f x > 0"*
shows *"((λx. Gamma_incl s (f x)) has_field_derivative (f x powr (s-1) * exp (-f x) * f')) (at x within A)"*
⟨*proof*⟩

6.7 Special values

Lastly, we examine the values of $\Gamma(s, z)$ specifically for $z = 0$, s is a positive integer, $s = \frac{1}{2}$, and $z \to \infty$.

lemma *Gamma_incl_0_left: "Gamma_incl 0 z = 0"*
⟨*proof*⟩

lemma *Gamma_incl_0_right [simp]: "s ≠ 0 ⟹ Gamma_incl s 0 = 0"*
⟨*proof*⟩

lemma *Gamma_incu_0_right_complex [simp]: "Re s > 0 ⟹ Gamma_incu s 0 = Gamma (s::complex)"*
⟨*proof*⟩

lemma *Gamma_incu_0_right_real [simp]: "s > 0 ⟹ Gamma_incu s 0 = Gamma (s::real)"*
⟨*proof*⟩

The following theorem now summarises the behaviour of $\Gamma(s,z)$ at $z=0$ and $\mathrm{Re}(s)>0$: when approaching from direction $\mathrm{Re}(z)>0$, $\Gamma(s,z)\to\Gamma(s')$ as $s\to s'$ and $z\to 0$.

theorem *tendsto_Gamma_incu_0_right_complex:*
  assumes *"(f ⟶ s) F"* *"(g ⟶ 0) F"*
  assumes *"Re s > 0"* *"eventually (λx. Re (g x) > 0) F"*
  shows *"((λx. Gamma_incu (f x) (g x)) ⟶ Gamma s) F"*
⟨*proof*⟩

lemma *Gamma_incl_1_left: "Gamma_incl 1 z = 1 - exp (-z)"*
  ⟨*proof*⟩

lemma *Gamma_incu_1_left_complex: "Gamma_incu 1 (z::complex) = exp (-z)"*
  ⟨*proof*⟩

lemma *Gamma_incu_1_left_real: "z ≥ 0 ⟹ Gamma_incu 1 (z::real) = exp (-z)"*
  ⟨*proof*⟩

theorem *Gamma_incl_of_nat_left_complex:*
  fixes *z :: complex*
  shows *"Gamma_incl (of_nat (Suc n)) z = fact n * (1 - exp (-z) * (∑k≤n. z ^ k / fact k))"*
⟨*proof*⟩

lemma *Gamma_incl_of_nat_left_real:*
  fixes *z :: real*
  shows *"Gamma_incl (of_nat (Suc n)) z = fact n * (1 - exp (-z) * (∑k≤n. z ^ k / fact k))"*
⟨*proof*⟩

theorem *Gamma_incu_of_nat_left_complex:*
  fixes *z :: complex*
  shows *"Gamma_incu (of_nat (Suc n)) z = fact n * exp (-z) * (∑k≤n. z ^ k / fact k)"*
⟨*proof*⟩

lemma *Gamma_incu_of_nat_left_real:*
  fixes *z :: real*
  shows *"Gamma_incu (of_nat (Suc n)) z = fact n * exp (-z) * (∑k≤n. z ^ k / fact k)"*
⟨*proof*⟩

Via the hypergeometric representation, it is easy to see that for $\gamma(\frac{1}{2},z)$ and $\Gamma(\frac{1}{2},z)$ have representations in terms of $\mathrm{erf}(\sqrt{z})$ and $texterfc(\sqrt{z})$, respectively:

theorem *Gamma_incl_one_half_left_complex:*
assumes *"z = 0 ∨ z ∉ ℝ≤0"*
shows *"Gamma_incl (1/2) (z :: complex) = sqrt pi * erf (csqrt z)"*
⟨*proof*⟩

lemma *Gamma_incl_one_half_left_real:*
assumes *"z ≥ 0"*
shows *"Gamma_incl (1/2) (z :: real) = sqrt pi * erf (sqrt z)"*
⟨*proof*⟩

lemma *Gamma_incu_one_half_left_complex:*
assumes *"z = 0 ∨ z ∉ ℝ≤0"*
shows *"Gamma_incu (1/2) (z :: complex) = sqrt pi * erfc (csqrt z)"*
⟨*proof*⟩

lemma *Gamma_incu_one_half_left_real:*
assumes *"z ≥ 0"*
shows *"Gamma_incu (1/2) (z :: real) = sqrt pi * erfc (sqrt z)"*
⟨*proof*⟩

$\Gamma(s, x)$ vanishes as $x \to \infty$.

lemma *Gamma_incu_at_top_complex: "((λz. Gamma_incu s (complex_of_real z)) ⟶ 0) at_top"*
⟨*proof*⟩

lemma *Gamma_incu_at_top_real: "((λz. Gamma_incu s (z::real)) ⟶ 0) at_top"*
⟨*proof*⟩

Consequently, $\gamma(s, z) \to \Gamma(s)$ as $z \to \infty$:

lemma *Gamma_incl_at_top_complex:*
assumes *"s ∉ ℤ≤0"*
shows *"((λz. Gamma_incl s (complex_of_real z)) ⟶ Gamma s) at_top"*
⟨*proof*⟩

lemma *Gamma_incl_at_top_real:*
assumes *"s ∉ ℤ≤0"*
shows *"((λz. Gamma_incl s (z::real)) ⟶ Gamma s) at_top"*
⟨*proof*⟩

end

References

[1] NIST Digital Library of Mathematical Functions. https://dlmf.nist.gov/, Release 1.2.4 of 2025-03-15. F. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert,

C. W. Clark, B. R. Miller, B. V. Saunders, H. S. Cohl, and M. A. McClain, eds.