Bessel Functions of the First Kind

Manuel Eberl

July 8, 2026

Abstract

This entry provides a formalisation of the Bessel function of the first kind J_a and the modified Bessel function of the first kind I_a as well as several related notions such as Bessel polynomials and the spherical Bessel functions.

Properties that are proven about I_a and J_a include:

- the definition as a hypergeometric series

$$\sum_{n\geq 0}(-1)^n(z/2)^{a+2n}/(n!\Gamma(1+a+n))$$

- monotonicity and convexity of I_a on the reals
- the contiguous relations, e.g. $2aJ_a(z) = z(J_{a-1}(z) + J_{a+1}(z))$
- the derivatives, e.g. $J'_a = \frac{1}{2}(J_{a-1} - J_{a+1})$
- closed form expressions for the half-integer case $a = n + \frac{1}{2}$ in terms of trigonometric and hyperbolic functions
- the representation of J_a as contour integral of the form $\int_{c-i\infty}^{c+i\infty}$

Contents

1 Bessel functions **3**
1.1 Auxiliary material 3
1.2 Hyperbolic sine and cosine as formal power series 4
1.3 The cardinal hyperbolic sine 5
1.4 Bessel polynomials 7
1.5 Spherical Bessel and Hankel functions 14
1.5.1 The spherical Bessel function of the first kind 14
1.5.2 The spherical Bessel function of the second kind . . . 15
1.5.3 The modified spherical Bessel function of the first kind 16
1.5.4 Spherical Hankel functions 18
1.5.5 Closed form expressions 21
1.6 The Bessel–Clifford function 27
1.7 The Bessel function of the first kind 31
1.8 The modified Bessel function of the first kind 42

2 A Hankel-style integral formula for the Bessel I function 57
2.1 Auxiliary integrals 57
2.2 A Hankel-style representation for $1/\Gamma(s)$ 60
2.3 The integral for the Bessel function 84

1 Bessel functions

theory *Bessel*
  imports *"Incomplete_Gamma.Incomplete_Gamma"*
begin

1.1 Auxiliary material

lemma *rGamma_real_nonneg [simp]:*
  assumes *"x ≥ (0::real)"*
  shows *"rGamma x ≥ 0"*
  using *Gamma_real_pos[of x] assms* **by** *(cases "x = 0") (auto simp: rGamma_inverse_Gamma)*

lemma *sinh_of_real: "sinh (of_real x :: 'a :: {real_normed_field, banach})*
= of_real (sinh x)"
  by *(simp add: sinh_def scaleR_conv_of_real flip: exp_of_real)*

lemma *cosh_of_real: "cosh (of_real x :: 'a :: {real_normed_field, banach})*
= of_real (cosh x)"
  by *(simp add: cosh_def scaleR_conv_of_real flip: exp_of_real)*

lemma *plus_of_nat_in_nonpos_IntsD:*
  assumes *"x + of_nat n ∈ ℤ≤0"*
  shows *"x ∈ ℤ≤0"*
proof -
  from *assms* **obtain** *k* **where** *"k ≤ 0" "x + of_nat n = of_int k"*
    by *(elim nonpos_Ints_cases) auto*
  hence *"x = of_int (k - int n)" "k - int n ≤ 0"*
    by *(auto simp: algebra_simps)*
  thus *?thesis*
    using *nonpos_Ints_of_int* **by** *blast*
qed

lemma *plus_of_nat_notin_nonpos_Ints:*
  assumes *"x ∉ ℤ≤0"*
  shows *"x + of_nat n ∉ ℤ≤0"*
  using *assms plus_of_nat_in_nonpos_IntsD* **by** *blast*

lemma *plus1_notin_nonpos_Ints:*
  assumes *"x ∉ ℤ≤0"*
  shows *"x + 1 ∉ ℤ≤0"*
  using *plus_of_nat_notin_nonpos_Ints[OF assms, of 1]* **by** *simp*

lemma *plus_numeral_notin_nonpos_Ints:*

assumes *"x ∉ ℤ≤0"*
shows *"x + numeral n ∉ ℤ≤0"*
using *plus_of_nat_notin_nonpos_Ints[OF assms, of "numeral n"]* **by** *simp*

1.2 Hyperbolic sine and cosine as formal power series

definition *fps_sinh :: "'a :: {inverse, ring_1} ⇒ 'a fps"* **where**
*"fps_sinh c = fps_const (1/2) * (fps_exp c - fps_exp (-c))"*

definition *fps_cosh :: "'a :: {inverse, ring_1} ⇒ 'a fps"* **where**
*"fps_cosh c = fps_const (1/2) * (fps_exp c + fps_exp (-c))"*

lemma *fps_nth_cosh:*
fixes *c :: "'a :: field_char_0"*
shows *"fps_nth (fps_cosh c) n = (if even n then c ^ n / of_nat (fact n) else 0)"*
by *(auto simp: fps_cosh_def)*

lemma *has_fps_expansion_sinh [fps_expansion_intros]:*
fixes *c :: "'a :: {banach, real_normed_field, field_char_0}"*
shows *"(λx. sinh (c * x)) has_fps_expansion fps_sinh c"*
proof -
have *"(λx. (1/2) * (exp (c*x) - exp ((-c) * x))) has_fps_expansion fps_sinh c"*
unfolding *sinh_def fps_sinh_def* **by** *(intro fps_expansion_intros)*
thus *?thesis*
by *(simp add: sinh_def scaleR_conv_of_real)*
qed

lemma *has_fps_expansion_sinh' [fps_expansion_intros]:*
"(λx::'a :: {banach, real_normed_field}. sinh x) has_fps_expansion fps_sinh 1"
using *has_fps_expansion_sinh[of 1]* **by** *simp*

lemma *has_fps_expansion_cosh [fps_expansion_intros]:*
fixes *c :: "'a :: {banach, real_normed_field, field_char_0}"*
shows *"(λx. cosh (c * x)) has_fps_expansion fps_cosh c"*
proof -
have *"(λx. (1/2) * (exp (c*x) + exp ((-c) * x))) has_fps_expansion fps_cosh c"*
unfolding *cosh_def fps_cosh_def* **by** *(intro fps_expansion_intros)*
thus *?thesis*
by *(simp add: cosh_def scaleR_conv_of_real)*
qed

lemma *has_fps_expansion_cosh' [fps_expansion_intros]:*
"(λx::'a :: {banach, real_normed_field}. cosh x) has_fps_expansion fps_cosh 1"
using *has_fps_expansion_cosh[of 1]* **by** *simp*

1.3 The cardinal hyperbolic sine

definition *sinch :: "'a ⇒ 'a :: {banach, real_normed_field, field_char_0}"*
where
"sinch z = hypergeo_F [] [3 / 2] (z^2 / 4)"

lemma *sinch_altdef: "sinch z = (if z = 0 then 1 else sinh z / z)"*
using *sinh_conv_hypergeo_F[of z]* **by** *(auto simp: sinch_def)*

lemma *sinch_0 [simp]: "sinch 0 = 1"*
by *(simp add: sinch_altdef)*

lemma *sinch_of_real: "sinch (of_real x) = of_real (sinch x)"*
by *(simp add: sinch_def of_real_hypergeo_F)*

lemma *sums_sinch': "(λn. x ^ (2*n) / fact (2 * n + 1)) sums sinch x"*
proof -
have *"(3 / 2 :: 'a) ∉ $\mathbb{Z}_{\leq 0}$"*
by *auto*
thus *?thesis*
using *sums_hypergeo_F[of "[3/2]" "[]" "x^2 / 4"]*
by *(simp add: sinch_def fps_hypergeo_nth pochhammer_three_halves*
*power_divide power_mult fact_reduce[of "2 * _ + 1"] add_ac)*
qed

lemma *sums_sinch: "(λn. if even n then x ^ n / fact (n+1) else 0) sums sinch x"*
proof -
have *"(λn. x ^ (2*n) / fact (2 * n + 1)) sums sinch x"*
by *(rule sums_sinch')*
also have *"?this ⟷ (λn. if even n then x ^ n / fact (n+1) else 0) sums sinch x"*
by *(subst (2) sums_mono_reindex[of "λn. 2 * n", symmetric]) (auto intro!: strict_monoI)*
finally show *?thesis* .
qed

lemma *has_field_derivative_sinch:*
assumes *"x ≠ 0"*
shows *"(sinch has_field_derivative (cosh x / x - sinh x / x ^ 2)) (at x within A)"*

```
proof -
  have "((λx. sinh x / x) has_field_derivative (cosh x / x - sinh x /
x ^ 2)) (at x within A)"
    using assms by (auto intro!: derivative_eq_intros simp: power2_eq_square
field_simps)
  also have "?this ⟷ (sinch has_field_derivative (cosh x / x - sinh
x / x ^ 2)) (at x within A)"
  proof (rule has_field_derivative_cong_eventually)
    have "eventually (λx. x ≠ 0) (at x within A)"
      by (rule eventually_neq_at_within)
    thus "eventually (λx. sinh x / x = sinch x) (at x within A)"
      by eventually_elim (auto simp: sinch_altdef)
  qed (use assms in ‹auto simp: sinch_altdef›)
  finally show ?thesis .
qed

lemma analytic_on_sinch [analytic_intros]:
  "f analytic_on A ⟹ (λz. sinch (f z)) analytic_on A"
  unfolding sinch_def
proof (intro analytic_intros)
  show "set [3 / 2 :: complex] ∩ ℤ≤0 = {}"
    by auto
qed auto

lemma holomorphic_on_sinch [holomorphic_intros]:
  "f holomorphic_on A ⟹ (λz. sinch (f z)) holomorphic_on A"
  unfolding sinch_def
proof (intro holomorphic_intros)
  show "set [3 / 2 :: complex] ∩ ℤ≤0 = {}"
    by auto
qed auto

lemma continuous_on_sinch [continuous_intros]:
  "continuous_on A f ⟹ continuous_on A (λz. sinch (f z))"
  unfolding sinch_def
proof (intro continuous_intros)
  show "set [3 / 2 :: 'b] ∩ ℤ≤0 = {}"
    by auto
qed auto

lemma subdegree_fps_sinh [simp]: "(c :: 'a :: field_char_0) ≠ 0 ⟹
subdegree (fps_sinh c) = 1"
  by (rule subdegreeI) (auto simp: fps_sinh_def)

lemma has_fps_expansion_sinch [fps_expansion_intros]:
  assumes [simp]: "c ≠ 0"
  shows   "(λz. sinch (c * z)) has_fps_expansion fps_sinh c / (fps_const
c * fps_X)"
proof -
```

```
  have [simp]: "fps_nth (fps_sinh c) (Suc 0) = c"
    by (simp add: fps_sinh_def)
  have "(λx. if x = 0 then 1 else sinh (c * x) / (c * x)) has_fps_expansion

          fps_sinh c / (fps_const c * fps_X)"
    by (intro has_fps_expansion_divide fps_expansion_intros) auto
  thus ?thesis
    unfolding sinch_altdef by simp
qed

lemma has_fps_expansion_sinch' [fps_expansion_intros]:
  "sinch has_fps_expansion fps_sinh 1 / fps_X"
  using has_fps_expansion_sinch[of 1] by simp

lemma has_laurent_expansion_sinch [laurent_expansion_intros]:
  assumes [simp]: "c ≠ 0"
  shows "(λz. sinch (c * z)) has_laurent_expansion fps_to_fls (fps_sinh
c) / (fls_const c * fls_X)"
proof -
  have "(λz. sinh (c * z) / (c * z)) has_laurent_expansion
          fps_to_fls (fps_sinh c) / (fls_const c * fls_X)"
    by (intro laurent_expansion_intros has_laurent_expansion_fps fps_expansion_intros)
  also have "?this ⟷ (λz. sinch (c * z))
                has_laurent_expansion (fps_to_fls (fps_sinh c) / (fls_const
c * fls_X))"
  proof (rule has_laurent_expansion_cong)
    have "eventually (λx. x ≠ (0::complex)) (at 0)"
      by (rule eventually_neq_at_within)
    thus "eventually (λz. sinh (c * z) / (c * z) = sinch (c * z)) (at
0)"
      by eventually_elim (auto simp: sinch_altdef)
  qed auto
  finally show ?thesis
    by simp
qed

lemma has_laurent_expansion_sinch' [laurent_expansion_intros]:
  "sinch has_laurent_expansion fps_to_fls (fps_sinh 1) / fls_X"
  using has_laurent_expansion_sinch[of 1] by simp
```

1.4 Bessel polynomials

The Bessel polynomials $y_n(X)$ are defined by the following recurrence:

$$\begin{aligned} y_0(X) &= 1 \\ y_1(X) &= 1 + X \\ y_n(X) &= (2n-1)Xy_{n-1}(X) + y_{n-2}(X) \end{aligned}$$

Later, we will additionally show the following alternative recurrence:

$$y_n(X) = (1 + nX)y_{n-1}(X) + X^2 y'_{n-1}(X)$$

```
fun bessel_poly :: "nat ⇒ 'a :: semidom poly" where
  "bessel_poly 0 = 1"
| "bessel_poly (Suc 0) = [:1, 1:]"
| "bessel_poly (Suc (Suc n)) = monom (of_nat (2*n+3)) 1 * bessel_poly
(Suc n) + bessel_poly n"

lemma coeff_bessel_poly_eq_0_aux:
  "m > n ⟹ coeff (bessel_poly n) m = 0"
  by (induction n arbitrary: m rule: bessel_poly.induct)
     (simp_all add: monom_altdef coeff_pCons split: nat.splits)

lemma bessel_poly_conv_bessel_poly_of_nat:
  "bessel_poly n = map_poly of_nat (bessel_poly n)"
proof (induction n rule: bessel_poly.induct)
  case (3 n)
  thus ?case
    by (auto simp: poly_eq_iff monom_altdef coeff_pCons coeff_map_poly
split: nat.splits)
qed (auto simp: map_poly_pCons)
```

Their coefficients ([**?**, A001498] on OEIS) have the following closed form:

$$[X^m]y_m(X) = \frac{(n+m)!}{2^m m!(n-m)!}$$

```
lemma coeff_bessel_poly_conv_fact_aux1:
  "m ≤ n ⟹ real (coeff (bessel_poly n) m) = fact (n+m) / (fact m *
fact (n-m) * 2^m)"
proof (induction n  arbitrary: m rule: bessel_poly.induct)
  case (3 n m)
  consider "m = 0" | "m ∈ {0<..n}" | "m = Suc n" | "m = Suc (Suc n)"
    using "3.prems" by force
  thus ?case
  proof cases
    assume "m = 0"
    thus ?thesis
      by (induction n rule: bessel_poly.induct)
         (simp_all add: monom_altdef coeff_pCons)
  next
    assume m_eq: "m = Suc (Suc n)"
    hence "m > Suc n"
      by auto
    have "real (coeff (bessel_poly (Suc (Suc n))) m) = fact (2 * n + 3)
/ (fact (n + 1)* 2 ^ Suc n)"
      using "3.prems"
```

```
      by (simp add: monom_altdef "3.IH" m_eq coeff_bessel_poly_eq_0_aux
fact_reduce flip: mult_2)
    also have "... = fact (Suc (Suc n) + m) / (fact m * fact (Suc (Suc
n) - m) * 2^m)"
      unfolding m_eq
      apply (simp del: fact_Suc add: add_ac eval_nat_numeral)
      apply (simp add: divide_simps)?
      done
    finally show ?case .
  next
    assume m_eq: "m = Suc n"
    hence "m > n"
      by auto
    have "real (coeff (bessel_poly (Suc (Suc n))) m) =
            (2 * real n + 3) * (2 * real n + 1) * fact (2 * n) / (fact
n * 2 ^ n)"
      using "3.prems"
      by (simp add: monom_altdef "3.IH" m_eq coeff_bessel_poly_eq_0_aux
flip: mult_2)
    also have "... = fact (Suc (Suc n) + m) / (fact m * fact (Suc (Suc
n) - m) * 2^m)"
      apply (simp del: fact_Suc add: add_ac eval_nat_numeral m_eq)
      apply (simp add: divide_simps flip: mult_2)?
      done
    finally show ?case .
  next
    assume m: "m ∈ {0<..n}"
    define k where "k = m - 1"
    have m_eq: "m = Suc k" and k: "k < n"
      using m by (auto simp: k_def)
    define D where "D = fact (n + k) / (fact (Suc k) * fact (Suc n -
k) * 2 ^ Suc k :: real)"
    have "real (coeff (bessel_poly (Suc (Suc n))) m) =
            (2 * real n + 3) * (1 + real n + real k) * fact (n + k) /

               (fact k * fact (Suc n - k) * 2 ^ k) +
            (1 + real n + real k) * fact (n + k) /
               (fact (Suc k) * fact (n - Suc k) * 2 ^ Suc k)"
      unfolding m_eq using k by (simp add: "3.IH" monom_altdef mult_ac
add_ac)
    also have "(2 * real n + 3) * (1 + real n + real k) * fact (n + k)
/
                 (fact k * fact (Suc n - k) * 2 ^ k) =
               2 * (real k + 1) * (2 * real n + 3) * (1 + real n + real
k) * D"
      by (simp add: divide_simps D_def)
    also have "(1 + real n + real k) * fact (n + k) /
                 (fact (Suc k) * fact (n - Suc k) * 2 ^ Suc k) =
               (1 + real n + real k) * (real n - real k + 1) * (real n
```

```
- real k) * (fact (n + k) /
                (fact (Suc k) * fact (Suc (Suc (n - Suc k))) * 2 ^ Suc
k))"
      using k unfolding fact_Suc by (simp add: divide_simps)
    also have "Suc (Suc (n - Suc k)) = Suc n - k"
      using k by (simp add: Suc_diff_le)
    also have "fact (n + k) / (fact (Suc k) * fact ... * 2 ^ Suc k) =
D"
      by (simp add: D_def)
    also have "2 * (real k + 1) * (2 * real n + 3) * (1 + real n + real
k) * D +
               (1 + real n + real k) * (real n - real k + 1) * (real n
- real k) * D =
               (real n + real k + 1) * (real n + real k + 2) * (real n
+ real k + 3) * D"
      by Groebner_Basis.algebra
    also have "... = fact (n + k + 3) / (fact (Suc k) * fact (Suc (Suc
(n - Suc k))) * 2 ^ Suc k)"
      using k by (simp add: D_def fact_reduce divide_simps add_ac Suc_diff_le)
    also have "... = fact (Suc (Suc n) + m) / (fact m * fact (Suc (Suc
n) - m) * 2 ^ m)"
      using k by (simp add: m_eq divide_simps Suc_diff_le eval_nat_numeral
Suc_diff_Suc del: fact_Suc)
    finally show ?case .
  qed
qed (auto simp: coeff_pCons split: nat.splits)

lemma degree_bessel_poly: "degree (bessel_poly n :: 'a :: {semidom, semiring_char_0}
poly) = n"
proof -
  have "coeff (bessel_poly n) n ≠ (0::'a)"
  proof -
    have "coeff (bessel_poly n) n = (of_nat (coeff (bessel_poly n) n)
:: 'a)"
      by (subst bessel_poly_conv_bessel_poly_of_nat) (simp_all add: coeff_map_poly)
    moreover have "real (coeff (bessel_poly n) n) ≠ 0"
      by (subst coeff_bessel_poly_conv_fact_aux1) auto
    hence "coeff (bessel_poly n) n ≠ (0 :: nat)"
      by auto
    ultimately show ?thesis
      by auto
  qed
  moreover have "coeff (bessel_poly n) m = (0 :: 'a)" if "m > n" for m
    using that by (simp add: coeff_bessel_poly_eq_0_aux)
  ultimately show ?thesis
    by (meson coeff_eq_0 less_degree_imp nat_neq_iff)
qed
```

The coefficients of the Bessel polynomials also satisfy the following recur-

rence:

```
fun bessel_poly_coeff :: "nat ⇒ nat ⇒ nat" where
  "bessel_poly_coeff n 0 = 1"
| "bessel_poly_coeff 0 (Suc m) = 0"
| "bessel_poly_coeff (Suc n) (Suc m) = bessel_poly_coeff n (Suc m) + (n
+ m + 1) * bessel_poly_coeff n m"

lemma bessel_poly_coeff_eq_0 [simp]: "m > n ⟹ bessel_poly_coeff n
m = 0"
  by (induction n m rule: bessel_poly_coeff.induct) auto

lemma bessel_poly_coeff_pos: "m ≤ n ⟹ bessel_poly_coeff n m > 0"
  by (induction n m rule: bessel_poly_coeff.induct) auto

lemma bessel_poly_coeff_eq_0_iff: "bessel_poly_coeff n m = 0 ⟷ m >
n"
  using bessel_poly_coeff_eq_0 bessel_poly_coeff_pos by (metis linorder_not_le
order_less_irrefl)

lemma bessel_poly_coeff_0_left: "m > 0 ⟹ bessel_poly_coeff 0 m = 0"
  by (cases m) auto

lemma bessel_poly_coeff_conv_fact_aux2:
  "m ≤ n ⟹ real (bessel_poly_coeff n m) = fact (n+m) / (fact m * fact
(n-m) * 2^m)"
proof (induction n m rule: bessel_poly_coeff.induct)
  case (3 n m)
  show ?case
  proof (cases "n = m")
    case True
    hence "real (bessel_poly_coeff (Suc n) (Suc m)) = fact (2*m+1) / (fact
m * 2 ^ m)"
      by (simp add: add_divide_distrib ring_distribs "3.IH"[unfolded True]
flip: mult_2)
    also have "... = fact (Suc n + Suc m) / (fact (Suc m) * fact (Suc
n - Suc m) * 2 ^ Suc m)"
      by (simp add: divide_simps True flip: mult_2)
    finally show ?thesis .
  next
    case False
    hence "m < n"
      using "3.prems" by simp
    hence "real (bessel_poly_coeff (Suc n) (Suc m)) =
             fact (n + m + 1) / (fact (Suc m) * fact (n - Suc m) * 2 ^
Suc m) +
             fact (n + m) / (fact m * fact (n - m) * 2 ^ m) +
             fact (n + m) * (n + m) / (fact m * fact (n - m) * 2 ^ m)"
      by (simp add: "3.IH")
    also have "fact (n + m + 1) = fact (n + m) * real (n + m + 1)"
```

```
      by (simp add: algebra_simps)
    also have "... / (fact (Suc m) * fact (n - Suc m) * 2 ^ Suc m) =
               fact (n + m) * (n + m + 1) * (n - m) / (fact (Suc m) *
fact (Suc (n - Suc m)) * 2 ^ Suc m)"
      unfolding fact_Suc
      using ‹m < n› apply (simp add: divide_simps del: of_nat_Suc)
      apply (simp add: algebra_simps)
      done
    also have "Suc (n - Suc m) = n - m"
      using ‹m < n› by simp
    also have "fact (n + m) / (fact m * fact (n - m) * 2 ^ m) =
               2 * (m + 1) * fact (n + m) / (fact (Suc m) * fact (n -
m) * 2 ^ Suc m)"
      apply (simp add: divide_simps del: of_nat_Suc)
      apply (simp add: algebra_simps)?
      done
    also have "fact (n + m) * (n + m) / (fact m * fact (n - m) * 2 ^ m)
=
               2 * fact (n + m) * (m + 1) * (n + m) / (fact (Suc m) *
fact (n - m) * 2 ^ Suc m)"
      apply (simp add: divide_simps del: of_nat_Suc)
      apply (simp add: algebra_simps)?
      done
    also have "fact (n + m) * (n + m + 1) * (n - m) / (fact (Suc m) *
fact (n - m) * 2 ^ Suc m) +
               2 * (m + 1) * fact (n + m) / (fact (Suc m) * fact (n -
m) * 2 ^ Suc m) +
               2 * fact (n + m) * (m + 1) * (n + m) / (fact (Suc m) *
fact (n - m) * 2 ^ Suc m) =
               (fact (n + m) * (n + m + 1) * (n - m) + 2 * (m + 1) * fact
(n + m) +
                  2 * fact (n + m) * (m + 1) * (n + m)) /
                 (fact (Suc m) * fact (n - m) * 2 ^ Suc m)"
      by (simp only: add_divide_distrib mult_ac of_nat_mult of_nat_add)
    also have "(fact (n + m) * (n + m + 1) * (n - m) + 2 * (m + 1) * fact
(n + m) +
                  2 * fact (n + m) * (m + 1) * (n + m)) =
               fact (n + m) * real ((n + m + 1) * (n - m) + 2 * (m + 1)
* (n + m + 1))"
      unfolding of_nat_mult of_nat_add ring_distribs of_nat_numeral of_nat_1
of_nat_fact
      by Groebner_Basis.algebra
    also have "real ((n + m + 1) * (n - m) + 2 * (m + 1) * (n + m + 1))
= (m + n + 1) * (m + n + 2)"
      using ‹m < n› unfolding of_nat_mult of_nat_add by (simp add: algebra_simps)
    also have "fact (n + m) * real ((m + n + 1) * (m + n + 2)) / (fact
(Suc m) * fact (n - m) * 2 ^ Suc m) =
                 fact (Suc n + Suc m) / (fact (Suc m) * fact (Suc n -
Suc m) * 2 ^ Suc m)"
```

```
    apply (simp add: divide_simps del: of_nat_Suc)
    apply (simp add: algebra_simps)?
    done
  finally show ?thesis .
 qed
qed auto

lemma bessel_poly_coeff_conv_fact:
 assumes "m ≤ n"
 shows   "bessel_poly_coeff n m = fact (n+m) div (fact m * fact (n-m)
* 2^m)"
proof -
 have "real (fact m * fact (n-m) * 2^m * bessel_poly_coeff n m) = real
(fact (n + m))"
   unfolding of_nat_mult by (subst bessel_poly_coeff_conv_fact_aux2)
(use assms in auto)
 hence *: "fact m * fact (n-m) * 2^m * bessel_poly_coeff n m = fact (n
+ m)"
   by (simp only: of_nat_eq_iff)
 show ?thesis
   by (subst * [symmetric]) simp_all
qed

lemma bessel_poly_coeff_conv_fact':
 "bessel_poly_coeff n m = (if m ≤ n then fact (n+m) div (fact m * fact
(n-m) * 2^m) else 0)"
 using bessel_poly_coeff_conv_fact[of m n] by auto

lemma coeff_bessel_poly: "coeff (bessel_poly n) m = of_nat (bessel_poly_coeff
n m)"
proof -
 have "coeff (bessel_poly n) m = (of_nat (coeff (bessel_poly n) m) ::
'a)"
   by (subst bessel_poly_conv_bessel_poly_of_nat) (simp_all add: coeff_map_poly)
 also have "coeff (bessel_poly n) m = bessel_poly_coeff n m"
 proof (cases "m ≤ n")
   case True
   show ?thesis
     using coeff_bessel_poly_conv_fact_aux1[OF True] bessel_poly_coeff_conv_fact_aux2[OF
True]
     by simp
 qed (simp_all add: coeff_bessel_poly_eq_0_aux)
 finally show ?thesis .
qed
```

The Bessel polynomials also satisfy the following recurrence with respect to their derivative:

$$y_n(X) = (1 + nX)y_{n-1}(X) + X^2 y'_{n-1}(X)$$

```
lemma bessel_poly_Suc_conv_pderiv:
  "bessel_poly (Suc n) = [:1, of_nat n + 1:] * bessel_poly n + [:0,0,1:]
* pderiv (bessel_poly n)"
  by (rule poly_eqI)
     (auto simp: coeff_bessel_poly coeff_pCons algebra_simps coeff_pderiv
split: nat.splits)
```

1.5 Spherical Bessel and Hankel functions

1.5.1 The spherical Bessel function of the first kind

The spherical Bessel functions of the first kind $j_n(z)$ are defined by letting $j_0(z) = \sin z/z$ and $j_{-1} = \cos z/z$ and then extending the definition to all integers n via the following recurrence:

$$(2n+1)j_n(z) = z(j_{n+1}(z) + j_{n-1}(z))$$

```
function SBessel_J :: "int ⇒ 'a :: {banach, real_normed_field} ⇒ 'a"
where
  "SBessel_J 0 = (λz. sin z / z)"
| "SBessel_J (-1) = (λz. cos z / z)"
| "n > 0 ⟹ SBessel_J n = (λz. (2 * of_int n - 1) / z * SBessel_J (n-1)
z - SBessel_J (n-2) z)"
| "n < -1 ⟹ SBessel_J n = (λz. (2 * of_int n + 3) / z * SBessel_J (n+1)
z - SBessel_J (n+2) z)"
  by force+
termination
  by (relation "Wellfounded.measure (λn. nat (abs (2 * n + 1)))") (auto
simp: abs_if)

lemmas [simp del] = SBessel_J.simps(3,4)

lemma SBessel_J_induct:
  "P 0 ⟹ P (-1) ⟹
    (⋀n. 0 < n ⟹ (P (n - 1)) ⟹ (P (n - 2)) ⟹ P n) ⟹
    (⋀n. n < - 1 ⟹ (P (n + 1)) ⟹ (P (n + 2)) ⟹ P n) ⟹
  P (n :: int)"
  by (induction n rule: SBessel_J.induct; metis)

lemma SBessel_J_0_right [simp]: "SBessel_J n 0 = 0"
  by (induction n rule: SBessel_J_induct) (auto simp: SBessel_J.simps)

lemma SBessel_J_of_real: "SBessel_J n (of_real x) = of_real (SBessel_J
n x)"
  by (induction n rule: SBessel_J_induct) (simp_all add: SBessel_J.simps
sin_of_real cos_of_real)

lemma SBessel_J_minus_right: "SBessel_J n (-z) = (-1) powi n * SBessel_J
n z"
```

```
  by (induction n rule: SBessel_J_induct)
     (auto simp: power_int_minus_left SBessel_J.simps)

lemma SBessel_J_contiguous:
  assumes "z ≠ 0"
  shows   "(2 * of_int n + 1) * SBessel_J n z = z * (SBessel_J (n+1) z
+ SBessel_J (n-1) z)"
proof (cases n rule: SBessel_J.cases)
  case 1
  thus ?thesis using assms
    by (simp add: SBessel_J.simps field_simps)
next
  case 2
  thus ?thesis using assms
    by (simp add: SBessel_J.simps field_simps)
next
  case (3 n)
  thus ?thesis using assms
    by (simp add: SBessel_J.simps field_simps)
next
  case (4 n)
  thus ?thesis using assms
    by (simp add: SBessel_J.simps field_simps)
qed
```

1.5.2 The spherical Bessel function of the second kind

For convenience, we also define the closely related spherical Bessel functions of the second kind $y_n(z)$. They satisfy the same recurrence, but with the initial conditions $y_0(z) = -\cos z/z$ and $y_{-1}(z) = \sin z/z$. They can easily be expressed in terms of $j_n(z)$ and vice versa.

```
definition SBessel_Y :: "int ⇒ 'a :: {real_normed_field,banach} ⇒ 'a"
  where "SBessel_Y n z = (-1) powi (n+1) * SBessel_J (-n-1) z"

lemma SBessel_Y_0_right [simp]: "SBessel_Y n 0 = 0"
  by (simp add: SBessel_Y_def)

lemma SBessel_Y_conv_J: "SBessel_Y n z = (-1) powi (n+1) * SBessel_J
(-n-1) z"
  unfolding SBessel_Y_def ..

lemma SBessel_J_conv_Y: "SBessel_J n z = (-1) powi n * SBessel_Y (-n-1)
z"
  unfolding SBessel_Y_conv_J by (simp add: power_int_minus)

lemma SBessel_Y_minus_left: "SBessel_Y (-n) z = (-1) powi (n+1) * SBessel_J
(n - 1) z"
  by (subst SBessel_Y_conv_J) (simp_all add: power_int_minus_left)
```

```
lemma SBessel_J_minus_left: "SBessel_J (-n) z = (-1) powi n * SBessel_Y
(n - 1) z"
  by (subst SBessel_Y_conv_J) (simp_all add: power_int_minus_left)

lemma SBessel_Y_minus_right: "SBessel_Y n (-z) = (-1) powi (n+1) * SBessel_Y
n z"
  by (simp add: SBessel_Y_conv_J SBessel_J_minus_right power_int_minus_left)

lemma SBessel_Y_of_real: "SBessel_Y n (of_real x) = of_real (SBessel_Y
n x)"
  by (simp add: SBessel_Y_conv_J SBessel_J_of_real)

lemma SBessel_Y_contiguous:
  assumes "z ≠ 0"
  shows   "(2 * of_int n + 1) * SBessel_Y n z = z * (SBessel_Y (n - 1)
z + SBessel_Y (n + 1) z)"
proof -
  have "z * SBessel_Y (n - 1) z = (-1) powi n * (z * SBessel_J (-n) z)"
    unfolding SBessel_Y_conv_J by simp
  also have "z * SBessel_J (-n) z = -(2 * of_int n + 1) * SBessel_J (-n-1)
z - z * SBessel_J (-n-2) z"
    using SBessel_J_contiguous[of z "-n-1"] assms by (simp add: power_int_add
algebra_simps)
  also have "(-1) powi n * ... = (2 * of_int n + 1) * SBessel_Y n z  -
z * SBessel_Y (1 + n) z"
    unfolding SBessel_J_conv_Y
    by (simp_all add: power_int_diff power_int_minus algebra_simps flip:
power_int_inverse)
  finally show ?thesis
    by (simp add: algebra_simps)
qed
```

1.5.3 The modified spherical Bessel function of the first kind

The modified spherical Bessel functions of the first kind are the hyperbolic versions of $j_n(z)$. They satisfy a slightly different recurrence and, in the complex case, can easily be written in terms of j_n and vice versa.

```
function SBessel_I :: "int ⇒ 'a :: {banach, real_normed_field} ⇒ 'a"
where
  "SBessel_I 0 = (λz. sinh z / z)"
| "SBessel_I (-1) = (λz. cosh z / z)"
| "n > 0 ⟹ SBessel_I n = (λz. -(2 * of_int n - 1) / z * SBessel_I (n-1)
z + SBessel_I (n-2) z)"
| "n < -1 ⟹ SBessel_I n = (λz. (2 * of_int n + 3) / z * SBessel_I (n+1)
z + SBessel_I (n+2) z)"
  by force+
termination
```

```
  by (relation "Wellfounded.measure (λn. nat (abs (2 * n + 1)))") (auto
simp: abs_if)

lemmas [simp del] = SBessel_I.simps(3,4)

lemma SBessel_I_0_right [simp]: "SBessel_I n 0 = 0"
  by (induction n rule: SBessel_J_induct) (auto simp: SBessel_I.simps)

lemma SBessel_I_of_real: "SBessel_I n (of_real x) = of_real (SBessel_I
n x)"
  by (induction n rule: SBessel_J_induct) (simp_all add: SBessel_I.simps
sinh_of_real cosh_of_real)

lemma SBessel_I_minus_right: "SBessel_I n (-z) = (-1) powi n * SBessel_I
n z"
  by (induction n rule: SBessel_J_induct)
     (auto simp: power_int_minus_left SBessel_I.simps)

lemma SBessel_I_contiguous:
  assumes "z ≠ 0"
  shows   "(2 * of_int n + 1) * SBessel_I n z = z * (SBessel_I (n-1) z
- SBessel_I (n+1) z)"
proof (cases n rule: SBessel_I.cases)
  case 1
  thus ?thesis using assms
    by (simp add: SBessel_I.simps field_simps)
next
  case 2
  thus ?thesis using assms
    by (simp add: SBessel_I.simps field_simps)
next
  case (3 n)
  thus ?thesis using assms
    by (simp add: SBessel_I.simps field_simps)
next
  case (4 n)
  thus ?thesis using assms
    by (simp add: SBessel_I.simps field_simps)
qed

lemma SBessel_I_conv_J: "SBessel_I n z = (-i) powi n * SBessel_J n (i
* z)"
proof (cases "z = 0")
  case False
  thus ?thesis
    by (induction n rule: SBessel_J_induct)
       (simp_all add: SBessel_I.simps SBessel_J.simps sin_conv_sinh cos_conv_cosh
                      power_int_add power_int_diff field_simps)
qed auto
```

lemma *SBessel_J_conv_I: "SBessel_J n z =* i *powi n * SBessel_I n (-*i ** z)"*
by *(subst SBessel_I_conv_J) (auto simp flip: mult.assoc power_int_mult_distrib)*

1.5.4 Spherical Hankel functions

The spherical Hankel functions are complex functions that can be written as linear combinations of the spherical Bessel functions. Therefore, they also satisfy the same contiguous relations.

definition *SHankel_1 :: "int ⇒ complex ⇒ complex"*
where *"SHankel_1 n z = SBessel_J n z +* i ** SBessel_Y n z"*

definition *SHankel_2 :: "int ⇒ complex ⇒ complex"*
where *"SHankel_2 n z = SBessel_J n z -* i ** SBessel_Y n z"*

lemma *SHankel_1_minus_left: "SHankel_1 (-n) z = -*i ** (-1) powi n * SHankel_1 (n-1) z"*
unfolding *SHankel_1_def SHankel_2_def SBessel_J_minus_left SBessel_Y_minus_left*
by *(simp add: power_int_add algebra_simps)*

lemma *SHankel_2_minus_left: "SHankel_2 (-n) z =* i ** (-1) powi n * SHankel_2 (n-1) z"*
unfolding *SHankel_1_def SHankel_2_def SBessel_J_minus_left SBessel_Y_minus_left*
by *(simp add: power_int_add algebra_simps)*

lemma *SHankel_1_minus_right: "SHankel_1 n (-z) = (-1) powi n * SHankel_2 n z"*
by *(simp add: SHankel_1_def SHankel_2_def SBessel_J_minus_right SBessel_Y_minus_right power_int_add algebra_simps)*

lemma *SHankel_2_minus_right: "SHankel_2 n (-z) = (-1) powi n * SHankel_1 n z"*
by *(simp add: SHankel_1_def SHankel_2_def SBessel_J_minus_right SBessel_Y_minus_right power_int_add algebra_simps)*

lemma *SBessel_J_conv_SHankel: "SBessel_J n z = (SHankel_1 n z + SHankel_2 n z) / 2"*
by *(simp add: SHankel_1_def SHankel_2_def)*

lemma *SBessel_Y_conv_SHankel: "SBessel_Y n z = (SHankel_1 n z - SHankel_2 n z) / (2 ** i*)"*
by *(simp add: SHankel_1_def SHankel_2_def)*

lemma *SHankel_1_contiguous:*
assumes *"z ≠ 0"*
shows *"(2 * of_int n + 1) * SHankel_1 n z = z * (SHankel_1 (n - 1) z + SHankel_1 (n + 1) z)"*

```
proof -
  have "(2 * of_int n + 1) * SHankel_1 n z =
        (2 * of_int n + 1) * SBessel_J n z + i * ((2 * of_int n + 1) *
SBessel_Y n z)"
    unfolding SHankel_1_def by algebra
  also have "... = z * (SHankel_1 (n-1) z + SHankel_1 (n+1) z)"
    unfolding SHankel_1_def SBessel_J_contiguous[OF assms] SBessel_Y_contiguous[OF
assms]
    by (simp add: algebra_simps)
  finally show ?thesis .
qed

lemma SHankel_2_contiguous:
  assumes "z ≠ 0"
  shows   "(2 * of_int n + 1) * SHankel_2 n z = z * (SHankel_2 (n - 1)
z + SHankel_2 (n + 1) z)"
proof -
  have "(2 * of_int n + 1) * SHankel_2 n z =
        (2 * of_int n + 1) * SBessel_J n z - i * ((2 * of_int n + 1) *
SBessel_Y n z)"
    unfolding SHankel_2_def by algebra
  also have "... = z * (SHankel_2 (n-1) z + SHankel_2 (n+1) z)"
    unfolding SHankel_2_def SBessel_J_contiguous[OF assms] SBessel_Y_contiguous[OF
assms]
    by (simp add: algebra_simps)
  finally show ?thesis .
qed
```

The spherical Hankel functions have a simple closed form in terms of the exponential and the Bessel polynomials.

```
lemma SHankel_1_nonneg_eq:
  assumes "z ≠ 0"
  shows   "SHankel_1 (int n) z = (-i)^(n+1) * exp (i*z) * poly (bessel_poly
n) (i/z) / z"
proof (induction n rule: induct_nat_012)
  case 0
  thus ?case
    using assms by (simp add: SHankel_1_def SBessel_Y_def sin_exp_eq cos_exp_eq
field_simps)
next
  case 1
  thus ?case
    using assms by (simp add: SHankel_1_def SBessel_Y_def sin_exp_eq cos_exp_eq
field_simps SBessel_J.simps)
next
  case (ge2 n)
  have "(-i) ^ (Suc (Suc n) + 1) * exp (i*z) * poly (bessel_poly (Suc (Suc
n))) (i/z) / z =
          (2 * of_nat n + 3) * ((-i)^(Suc n + 1) * exp (i*z) * poly (bessel_poly
```

```
(Suc n)) (i/z) / z) / z -
           (-i)^(n+1) * exp (i*z) * poly (bessel_poly n) (i/z) / z"
    using assms by (simp add: poly_monom field_simps)
  also have "... = ((2 * of_nat n + 3) * SHankel_1 (int (Suc n)) z) /
z - SHankel_1 (int n) z"
    unfolding ge2.IH [symmetric] ..
  also have "(2 * of_nat n + 3) * SHankel_1 (int (Suc n)) z =
              z * (SHankel_1 (int n) z + SHankel_1 (2 + int n) z)"
    using SHankel_1_contiguous[of z "int (Suc n)"] assms by (simp add:
algebra_simps)
  also have "... / z - SHankel_1 (int n) z = SHankel_1 (Suc (Suc n)) z"
    using assms by simp
  finally show ?case ..
qed

lemma SHankel_2_nonneg_eq:
  assumes "z ≠ 0"
  shows   "SHankel_2 (int n) z = i^(n+1) * exp (-i*z) * poly (bessel_poly
n) (-i/z) / z"
proof -
  have "SHankel_2 (int n) z = (-1) powi n * SHankel_1 (int n) (-z)"
    by (simp add: SHankel_1_minus_right)
  also have "... = i^(n+1) * exp (-i*z) * poly (bessel_poly n) (-i/z) /
z"
    using assms by (subst SHankel_1_nonneg_eq) (auto simp: uminus_power_if)
  finally show ?thesis .
qed

lemma SHankel_1_neg_eq:
  assumes "z ≠ 0"
  shows   "SHankel_1 (-int (n+1)) z = i ^ n * exp (i * z) * poly (bessel_poly
n) (i / z) / z"
proof -
  have "SHankel_1 (-int (n+1)) z = -i * (-1) ^ (n+1) * SHankel_1 (int
n) z"
    by (subst SHankel_1_minus_left) (auto simp: power_int_add)
  also have "... = i ^ n * exp (i * z) * poly (bessel_poly n) (i / z) /
z"
    by (subst SHankel_1_nonneg_eq[OF assms]) (auto simp: power_minus'
mult_ac)
  finally show ?thesis .
qed

lemma SHankel_2_neg_eq:
  assumes "z ≠ 0"
  shows   "SHankel_2 (-int (n+1)) z = (-i) ^ n * exp (-i * z) * poly (bessel_poly
n) (-i / z) / z"
proof -
  have "SHankel_2 (-int (n+1)) z = i * (-1) ^ (n+1) * SHankel_2 (int n)
```

```
z"
    by (subst SHankel_2_minus_left) (auto simp: power_int_add)
  also have "... = (-i) ^ n * exp (-i * z) * poly (bessel_poly n) (-i /
z) / z"
    by (subst SHankel_2_nonneg_eq[OF assms]) (auto simp: power_minus'
mult_ac)
  finally show ?thesis .
qed
```

1.5.5 Closed form expressions

We can now give closed-form expressions for all the spherical Bessel functions in terms of sin, cos, sinh, and cosh.

```
lemma SBessel_J_conv_sin_cos_nonneg_complex:
  fixes z :: complex and n :: nat
  shows  "SBessel_J (int n) z =
            (∑k≤n. (-1)^((n+k+1) div 2 + k) * of_nat (bessel_poly_coeff
n k) / z^(k+1) *
                      (if even (n + k) then sin z else cos z))"
proof (cases "z = 0")
  case [simp]: False
  define c :: "nat ⇒ complex" where "c = coeff (bessel_poly n)"
  have "SBessel_J (int n) z =
          ((-i) ^ (n+1) * exp (i*z) * poly (bessel_poly n) (i/z) +
           i ^ (n+1) * exp (-i*z) * poly (bessel_poly n) (-i/z)) / (2 *
z)"
    unfolding SBessel_J_conv_SHankel SHankel_1_nonneg_eq[OF False] SHankel_2_nonneg_eq[OF
False]
    by (simp add: field_simps)
  also have "... = (∑k≤n. (-i)^(n+1) * exp (i*z) * c k * (i/z)^k +
                          i^(n+1) * exp (-i*z) * c k * (-i/z) ^ k) / (2*z)"
    unfolding poly_altdef degree_bessel_poly sum.distrib [symmetric] sum_distrib_left
    by (simp add: c_def mult_ac)
  also have "... = (∑k≤n. (-1)^((n+k+1) div 2 + k) * c k / z^(k+1) *

                         (if even (n + k) then sin z else cos z))"
    unfolding sum_divide_distrib
  proof (intro sum.cong, goal_cases)
    case (2 k)
    have "(-i)^(n+1) * exp (i*z) * c k * (i/z)^k / (2*z) + i^(n+1) * exp
(-i*z) * c k * (-i/z) ^ k / (2*z) =
          c k / z ^ Suc k * (i^(n+k+1) * (-1)^k * (((-1)^(n+k+1) * exp
(i*z) + exp (-i*z)) / 2))"
      by (auto simp: power_minus' field_simps power_add)
    also have "((-1)^(n+k+1) * exp (i*z) + exp (-i*z)) / 2 = (if even (n+k)
then -i * sin z else cos z)"
      by (auto simp: cos_exp_eq sin_exp_eq field_simps)
    also have "i^(n+k+1) * (-1)^k * ... = (-1) ^ ((n+k+1) div 2 + k) *
(if even (n+k) then sin z else cos z)"
```

```
    by (cases "even n"; cases "even k")
       (auto elim!: evenE oddE simp: uminus_power_if split: if_splits)
  finally show ?case
    by (simp add: field_simps)
qed auto
finally show ?thesis
  by (simp add: coeff_bessel_poly c_def)
qed auto

lemma SBessel_Y_conv_sin_cos_nonneg_complex:
  fixes z :: complex and n :: nat
  shows  "SBessel_Y (int n) z =
           (∑k≤n. (-1)^((n+k) div 2 + k + 1) * of_nat (bessel_poly_coeff
n k) / z^(k+1) *
                       (if even (n + k) then cos z else sin z))"
proof (cases "z = 0")
  case [simp]: False
  define c :: "nat ⇒ complex" where "c = coeff (bessel_poly n)"
  have "SBessel_Y (int n) z =
         ((-i) ^ (n+1) * exp (i*z) * poly (bessel_poly n) (i/z) -
          i ^ (n+1) * exp (-i*z) * poly (bessel_poly n) (-i/z)) / (2 *
i * z)"
    unfolding SBessel_Y_conv_SHankel SHankel_1_nonneg_eq[OF False] SHankel_2_nonneg_eq[OF
False]
    by (simp add: field_simps)
  also have "... = (∑k≤n. (-i)^(n+1) * exp (i*z) * c k * (i/z)^k -
                          i^(n+1) * exp (-i*z) * c k * (-i/z) ^ k) / (2*i*z)"
    unfolding poly_altdef degree_bessel_poly sum_subtractf sum_distrib_left
    by (simp add: c_def mult_ac)
  also have "... = (∑k≤n. (-1)^((n+k) div 2 + k + 1) * c k / z^(k+1)
*
                         (if even (n + k) then cos z else sin z))"
    unfolding sum_divide_distrib
  proof (intro sum.cong, goal_cases)
    case (2 k)
    have "(-i)^(n+1) * exp (i*z) * c k * (i/z)^k / (2*i*z) - i^(n+1) * exp
(-i*z) * c k * (-i/z) ^ k / (2*i*z) =
          c k / z ^ Suc k * (i^(n+k) * (-1)^k * (((-1)^(n+k+1) * exp (i*z)
- exp (-i*z)) / 2))"
      by (auto simp: power_minus' field_simps power_add)
    also have "((-1)^(n+k+1) * exp (i*z) - exp (-i*z)) / 2 = (if even (n+k)
then -cos z else i * sin z)"
      by (auto simp: cos_exp_eq sin_exp_eq field_simps)
    also have "i^(n+k) * (-1)^k * ... = (-1) ^ ((n+k) div 2 + k + 1) *
(if even (n+k) then cos z else sin z)"
      by (cases "even n"; cases "even k")
         (auto elim!: evenE oddE simp: uminus_power_if split: if_splits)
    finally show ?case
      by (simp add: field_simps)
```

```
  qed auto
  finally show ?thesis
    by (simp add: coeff_bessel_poly c_def)
qed auto

lemma SBessel_J_conv_sin_cos_neg_complex:
  fixes z :: complex and n :: nat
  shows  "SBessel_J (-int (n+1)) z =
            (∑k≤n. (-1)^(n + k + (n+k) div 2) * of_nat (bessel_poly_coeff
n k) / z^(k+1) *
                      (if even (n + k) then cos z else sin z))"
proof -
  have "SBessel_J (-int (n+1)) z = (-1)^(n+1) * SBessel_Y (int n) z"
    unfolding SBessel_J_conv_Y
    by (simp add: power_int_add minus_diff_commute power_int_minus uminus_power_if)
  also have "... = (∑j≤n. (-1)^(n+1) * ((-1) ^ ((n + j) div 2 + j +
1) *
                          of_nat (bessel_poly_coeff n j) / z^(j+1) *
                          (if even (n+j) then cos z else sin z))) "
    unfolding SBessel_Y_conv_sin_cos_nonneg_complex sum_distrib_left ..
  also have "... = (∑j≤n. (-1) ^ (n + j + (n + j) div 2) *
                          of_nat (bessel_poly_coeff n j) / z^(j+1) *
                          (if even (n+j) then cos z else sin z))"
    by (simp add: power_add mult_ac)
  finally show ?thesis .
qed

lemma SBessel_Y_conv_sin_cos_neg_complex:
  fixes z :: complex and n :: nat
  shows  "SBessel_Y (-int (n+1)) z =
            (∑k≤n. (-1)^(n + k + (n+k+1) div 2) * of_nat (bessel_poly_coeff
n k) / z^(k+1) *
                      (if even (n + k) then sin z else cos z))"
proof -
  have "SBessel_Y (-int (n+1)) z = (-1)^n * SBessel_J (int n) z"
    unfolding SBessel_Y_conv_J
    by (simp add: power_int_add minus_diff_commute power_int_minus uminus_power_if)
  also have "... = (∑j≤n. (-1)^n * ((-1) ^ ((n + j + 1) div 2 + j) *
                          of_nat (bessel_poly_coeff n j) / z^(j+1) *
                          (if even (n+j) then sin z else cos z))) "
    unfolding SBessel_J_conv_sin_cos_nonneg_complex sum_distrib_left ..
  also have "... = (∑j≤n. (-1) ^ (n + j + (n + j + 1) div 2) *
                          of_nat (bessel_poly_coeff n j) / z^(j+1) *
                          (if even (n+j) then sin z else cos z))"
    by (simp add: power_add mult_ac)
  finally show ?thesis .
qed

lemma SBessel_I_conv_sinh_cosh_nonneg_complex:
```

```
  fixes z :: complex and n :: nat
  shows  "SBessel_I (int n) z =
            (∑k≤n. (-1)^k * of_nat (bessel_poly_coeff n k) / z^(k+1)
*
                       (if even (n + k) then sinh z else cosh z))"
proof (cases "z = 0")
  case [simp]: False
  have "SBessel_I (int n) z =
          (-i) ^ n * (∑k≤n. (-1) ^ ((n + k + 1) div 2 + k) * of_nat (bessel_poly_coeff
n k) /
                                (i * z) ^ (k + 1) * (if even (n + k) then
sin (i * z) else cos (i * z)))"
    by (subst SBessel_I_conv_J, subst SBessel_J_conv_sin_cos_nonneg_complex)
auto
  also have "... = (∑k≤n. (-1)^k * of_nat (bessel_poly_coeff n k) /
z^(k+1) *
                       (if even (n + k) then sinh z else cosh z))"
    unfolding sum_distrib_left
    by (intro sum.cong)
       (auto simp: cosh_conv_cos sinh_conv_sin field_simps power_add elim!:
oddE evenE)
  finally show ?thesis .
qed auto

lemma SBessel_I_conv_sinh_cosh_neg_complex:
  fixes z :: complex and n :: nat
  shows  "SBessel_I (-int (n+1)) z =
            (∑k≤n. (-1)^k * of_nat (bessel_poly_coeff n k) / z^(k+1)
*
                       (if even (n + k) then cosh z else sinh z))"
proof (cases "z = 0")
  case [simp]: False
  have "SBessel_I (-int (n+1)) z =
          (-i) powi (-1 - int n) * (∑k≤n. (-1) ^ (n + k + (n + k) div
2) *
          of_nat (bessel_poly_coeff n k) *
          (if even n = even k then cos (i * z) else sin (i * z)) / (i *
z * (i * z) ^ k))"
    by (subst SBessel_I_conv_J, subst SBessel_J_conv_sin_cos_neg_complex)
auto
  also have "... = (∑k≤n. (-1)^k * of_nat (bessel_poly_coeff n k) /
z^(k+1) *
                       (if even (n + k) then cosh z else sinh z))"
    unfolding sum_distrib_left
    by (intro sum.cong)
       (auto simp: cosh_conv_cos sinh_conv_sin field_simps power_int_add
power_int_diff
                   power_int_minus power_add elim!: oddE evenE)
  finally show ?thesis .
```

qed *auto*

Variants of the above for the real-valued functions:

```
lemma SBessel_J_conv_sin_cos_nonneg_real:
  fixes z :: real and n :: nat
  shows  "SBessel_J (int n) z =
            (∑k≤n. (-1)^((n+k+1) div 2 + k) * of_nat (bessel_poly_coeff
n k) / z^(k+1) *
                      (if even (n + k) then sin z else cos z))" (is "_
= ?rhs")
proof -
  have "complex_of_real (SBessel_J (int n) z) = of_real ?rhs"
    unfolding SBessel_J_of_real [symmetric]
    by (subst SBessel_J_conv_sin_cos_nonneg_complex)
       (simp_all add: sin_of_real cos_of_real if_distrib[of of_real] cong:
if_cong)
  thus ?thesis
    by (simp only: of_real_eq_iff)
qed

lemma SBessel_Y_conv_sin_cos_nonneg_real:
  fixes z :: real and n :: nat
  shows  "SBessel_Y (int n) z =
            (∑k≤n. (-1)^((n+k) div 2 + k + 1) * of_nat (bessel_poly_coeff
n k) / z^(k+1) *
                      (if even (n + k) then cos z else sin z))" (is "_
= ?rhs")
proof -
  have "complex_of_real (SBessel_Y (int n) z) = of_real ?rhs"
    unfolding SBessel_Y_of_real [symmetric]
    by (subst SBessel_Y_conv_sin_cos_nonneg_complex)
       (simp_all add: sin_of_real cos_of_real if_distrib[of of_real] cong:
if_cong)
  thus ?thesis
    by (simp only: of_real_eq_iff)
qed

lemma SBessel_J_conv_sin_cos_neg_real:
  fixes z :: real and n :: nat
  shows  "SBessel_J (-int (n+1)) z =
              (∑k≤n. (-1)^(n + k + (n+k) div 2) * of_nat (bessel_poly_coeff
n k) / z^(k+1) *
                      (if even (n + k) then cos z else sin z))" (is "_
= ?rhs")
proof -
  have "complex_of_real (SBessel_J (-int (n+1)) z) = of_real ?rhs"
    unfolding SBessel_J_of_real [symmetric]
    by (subst SBessel_J_conv_sin_cos_neg_complex)
       (simp_all add: sin_of_real cos_of_real if_distrib[of of_real] cong:
```

```
if_cong)
  thus ?thesis
    by (simp only: of_real_eq_iff)
qed

lemma SBessel_Y_conv_sin_cos_neg_real:
  fixes z :: real and n :: nat
  shows  "SBessel_Y (-int (n+1)) z =
            (∑k≤n. (-1)^(n + k + (n+k+1) div 2) * of_nat (bessel_poly_coeff
n k) / z^(k+1) *
                      (if even (n + k) then sin z else cos z))" (is "_
= ?rhs")
proof -
  have "complex_of_real (SBessel_Y (-int (n+1)) z) = of_real ?rhs"
    unfolding SBessel_Y_of_real [symmetric]
    by (subst SBessel_Y_conv_sin_cos_neg_complex)
       (simp_all add: sin_of_real cos_of_real if_distrib[of of_real] cong:
if_cong)
  thus ?thesis
    by (simp only: of_real_eq_iff)
qed

lemma SBessel_I_conv_sinh_cosh_nonneg_real:
  fixes z :: real and n :: nat
  shows  "SBessel_I (int n) z =
            (∑k≤n. (-1)^k * of_nat (bessel_poly_coeff n k) / z^(k+1)
*
                      (if even (n + k) then sinh z else cosh z))" (is "_
= ?rhs")
proof -
  have "complex_of_real (SBessel_I (int n) z) = of_real ?rhs"
    unfolding SBessel_I_of_real [symmetric]
    by (subst SBessel_I_conv_sinh_cosh_nonneg_complex)
       (simp_all add: sinh_of_real cosh_of_real if_distrib[of of_real]
cong: if_cong)
  thus ?thesis
    by (simp only: of_real_eq_iff)
qed

lemma SBessel_I_conv_sinh_cosh_neg_real:
  fixes z :: real and n :: nat
  shows  "SBessel_I (-int (n+1)) z =
            (∑k≤n. (-1)^k * of_nat (bessel_poly_coeff n k) / z^(k+1)
*
                      (if even (n + k) then cosh z else sinh z))" (is "_
= ?rhs")
proof -
  have "complex_of_real (SBessel_I (-int (n+1)) z) = of_real ?rhs"
    unfolding SBessel_I_of_real [symmetric]
```

```
    by (subst SBessel_I_conv_sinh_cosh_neg_complex)
       (simp_all add: sinh_of_real cosh_of_real if_distrib[of of_real]
cong: if_cong)
  thus ?thesis
    by (simp only: of_real_eq_iff)
qed


experiment
begin
```

As an example: the close form of $j_3(z)$:

```
lemma "SBessel_J 3 (z :: complex) =
         15 * sin z / z ^ 4 + cos z / z - 6 * sin z / z ^ 2 - 15 * cos
z / z ^ 3"
  using SBessel_J_conv_sin_cos_nonneg_complex[of 3 z]
  by (simp add: atMost_nat_numeral bessel_poly_coeff_conv_fact' fact_numeral

              power_numeral_reduce mult_ac)

end
```

1.6 The Bessel–Clifford function

The Bessel–Clifford function C_a is a useful building block to construct the Bessel functions. It is a holomorphic function in two variables and therefore very well-behaved. It can be defined most easily via the regularised hypergeometric function.

Two important properties are that it is is that it satisfies $\frac{d}{dz}C_a(z) = C_{a+1}(z)$ and the contiguous relation $C_a(z) = (a+1)C_{a+1}(z) + zC_{a+2}(z)$, which already hints at its connection to the Bessel functions.

A direct consequence of this (which we do not show here) is that it is a solution of the ODE $y = (a+1)y' + xy''$.

We get these properties mostly for free using the development on hypergeometric functions.

```
definition Bessel_Clifford :: "'a :: Gamma ⇒ 'a ⇒ 'a" where
  "Bessel_Clifford a = reg_hypergeo_F [] [a+1]"

lemma Bessel_Clifford_altdef: "Bessel_Clifford a = eval_fps (fps_bessel
a)"
  by (simp add: fps_bessel_def Bessel_Clifford_def reg_hypergeo_F_def)

lemma Bessel_Clifford_conv_hypergeo_F:
  "a + 1 ∉ ℤ_≤0 ⟹ Bessel_Clifford a z = rGamma (a+1) * hypergeo_F []
[a+1] z"
  unfolding Bessel_Clifford_def by (subst reg_hypergeo_F_conv_hypergeo_F)
auto
```

```
lemma Bessel_Clifford_complex_of_real:
  "Bessel_Clifford (of_real a) (of_real z) = complex_of_real (Bessel_Clifford
a z)"
  unfolding Bessel_Clifford_def by (subst complex_of_real_reg_hypergeo_F)
auto

lemma sums_Bessel_Clifford:
  "(λn. rGamma (a + of_nat (Suc n)) * z ^ n / fact n) sums Bessel_Clifford
a z"
  using sums_reg_hypergeo_F[of "[]" "[a+1]" z] by (simp add: Bessel_Clifford_def
add_ac)

lemma Bessel_Clifford_contiguous:
  "Bessel_Clifford a z = (a+1) * Bessel_Clifford (a+1) z + z * Bessel_Clifford
(a+2) z"
proof -
  have conv1: "fps_conv_radius (fps_const (a + 1) * fps_bessel (a + 1))
≥ ∞"
    by (rule fps_conv_radius_mult_ge) auto
  have conv2: "fps_conv_radius (fps_X * fps_bessel (a + 2)) ≥ ∞"
    by (rule fps_conv_radius_mult_ge) auto

  have "Bessel_Clifford a z = eval_fps (fps_bessel a) z"
    by (simp add: Bessel_Clifford_altdef)
  also have "fps_bessel a = fps_const (a+1) * fps_bessel (a+1) + fps_X
* fps_bessel (a+2)"
    using fps_bessel_contiguous[of "a+1"] by (simp add: add_ac)
  also have "eval_fps ... z = eval_fps (fps_const (a+1) * fps_bessel (a+1))
z +
                              eval_fps (fps_X * fps_bessel (a+2)) z"
    by (subst eval_fps_add) (use conv1 conv2 in auto)
  also have "... = (a+1) * Bessel_Clifford (a+1) z + z * Bessel_Clifford
(a+2) z"
    unfolding Bessel_Clifford_altdef by (subst (1 2) eval_fps_mult) auto
  finally show ?thesis .
qed

lemma Bessel_Clifford_0_right [simp]: "Bessel_Clifford a 0 = rGamma (a
+ 1)"
  by (simp add: Bessel_Clifford_def reg_hypergeo_F_0)

lemma Bessel_Clifford_minus_of_nat:
  "Bessel_Clifford (-of_nat n) z = z ^ n * Bessel_Clifford (of_nat n)
z"
proof (cases "z = 0")
  case True
  show ?thesis
  proof (cases "n > 0")
```

```
    case True
    have "rGamma (of_int (1 - int n)) = 0"
      by (subst rGamma_of_int) (use ‹n > 0› in auto)
    thus ?thesis using ‹z = 0› ‹n > 0›
      by (auto simp: power_0_left)
  qed auto
next
  case False
  show ?thesis
  proof (induction "n + 1" arbitrary: n rule: less_induct)
    case (less n)
    consider "n = 0" | "n = 1" | "n ≥ 2"
      by linarith
    thus ?case
    proof cases
      assume [simp]: "n = 1"
      show ?case
        by (subst Bessel_Clifford_contiguous) simp_all
    next
      assume n: "n ≥ 2"
      have "Bessel_Clifford (-of_nat n) z =
              (-of_nat n + 1) * Bessel_Clifford (-of_nat (n-1)) z + z
* Bessel_Clifford (-of_nat (n-2)) z"
        by (subst Bessel_Clifford_contiguous) (use n in ‹simp_all add:
add_ac›)
      also have "... = (- of_nat n + 1) * z^(n-1) * Bessel_Clifford (of_nat
(n - 1)) z +
                       z * z^(n-2) * Bessel_Clifford (of_nat (n - 2)) z"
        by (subst (1 2) less) (use n in ‹auto simp: algebra_simps›)
      also have "... = z ^ n * Bessel_Clifford (of_nat n) z"
        by (subst (2) Bessel_Clifford_contiguous)
           (use n ‹z ≠ 0› in ‹auto simp: field_simps power_diff power2_eq_square›)
      finally show ?thesis .
    qed auto
  qed
qed

lemma Bessel_Clifford_minus_of_int:
  assumes "n ≥ 0 ∨ z ≠ 0"
  shows   "Bessel_Clifford (-of_int n) z = z powi n * Bessel_Clifford
(of_int n) z"
proof (cases "n ≥ 0")
  case True
  thus ?thesis
    using Bessel_Clifford_minus_of_nat[of "nat n" z] by (simp add: power_int_def)
next
  case False
  hence "z ≠ 0"
    using assms by auto
```

```
  show ?thesis
    using Bessel_Clifford_minus_of_nat[of "nat (-n)" z] ‹z ≠ 0› False
    by (simp add: power_int_def field_simps)
qed

lemma analytic_Bessel_Clifford [analytic_intros]:
  assumes "a analytic_on A" "b analytic_on A"
  shows   "(λx. Bessel_Clifford (a x) (b x)) analytic_on A"
  unfolding Bessel_Clifford_def
  by (rule analytic_reg_hypergeo_F[where as = "[]" and bs = "[λx. a
x + 1]"])
     (use assms in ‹auto intro!: analytic_intros›)

lemma has_field_derivative_Bessel_Clifford [derivative_intros]:
  assumes "(f has_field_derivative f') (at x within A)"
  shows   "((λx. Bessel_Clifford a (f x)) has_field_derivative
                  (Bessel_Clifford (a+1) (f x) * f')) (at x within A)"
  unfolding Bessel_Clifford_def
  by (rule DERIV_cong[OF has_field_derivative_reg_hypergeo_F'[OF assms]])
auto

lemma Bessel_Clifford_real_mono:
  assumes xy: "0 ≤ (x::real)" "x ≤ y" "a ≥ -1"
  shows  "Bessel_Clifford a x ≤ Bessel_Clifford a y"
proof (rule sums_le)
  show "(λn. rGamma (a + real (Suc n)) * x ^ n / fact n) sums Bessel_Clifford
a x"
    by (rule sums_Bessel_Clifford)
  show "(λn. rGamma (a + real (Suc n)) * y ^ n / fact n) sums Bessel_Clifford
a y"
    by (rule sums_Bessel_Clifford)
  show "rGamma (a + real (Suc n)) * x ^ n / fact n ≤
          rGamma (a + real (Suc n)) * y ^ n / fact n" for n using assms
    by (intro mult_left_mono divide_right_mono power_mono)
       (auto intro!: rGamma_real_nonneg)
qed

lemma Bessel_Clifford_real_pos [simp]:
  assumes "x ≥ (0::real)" "a > -1"
  shows   "Bessel_Clifford a x > 0"
proof -
  have "rGamma (a + 1) > 0"
    using assms by (auto simp: rGamma_inverse_Gamma intro!: Gamma_real_pos)
  also have "rGamma (a+1) = Bessel_Clifford a 0"
    by simp
  also have "Bessel_Clifford a 0 ≤ Bessel_Clifford a x"
    by (rule Bessel_Clifford_real_mono) (use assms in auto)
  finally show ?thesis
    by simp
```

qed

```
lemma Bessel_Clifford_real_nonneg [simp]: "x ≥ (0::real) ⟹ a ≥ -1
⟹ Bessel_Clifford a x ≥ 0"
  by (rule sums_le[OF _ sums_zero sums_Bessel_Clifford])
     (auto intro!: divide_nonneg_pos mult_nonneg_nonneg rGamma_real_nonneg)

lemma Bessel_Clifford_convex_real:
  assumes "a ≥ -3"
  shows "convex_on {0..} (Bessel_Clifford (a::real))"
proof (rule f''_ge0_imp_convex)
  show "(Bessel_Clifford a has_real_derivative Bessel_Clifford (a+1)
x) (at x)" for x
    by (auto intro!: derivative_eq_intros)
  show "(Bessel_Clifford (a+1) has_real_derivative Bessel_Clifford (a+2)
x) (at x)" for x
    by (auto intro!: derivative_eq_intros simp: add_ac)
  show "Bessel_Clifford (a+2) x ≥ 0" if x: "x ∈ {0..}" for x :: real
    by (rule Bessel_Clifford_real_nonneg) (use assms x in auto)
qed auto
```

1.7 The Bessel function of the first kind

The Bessel function of the first kind $J_a(\mathrm{z})$ can now easily be derived from the Bessel–Clifford function with the change of variables $z \mapsto -z^2/4$ and multiplication with $(z/2)^a$ (from which it gets its branch cut). All the basic properties follow directly from the ones we have for the Bessel–Clifford function.

```
definition Bessel_J :: "'a :: {Gamma, ln} ⇒ 'a ⇒ 'a" where
  "Bessel_J a z = (z / 2) powr' a * Bessel_Clifford a (-(z^2/4))"

lemma Bessel_J_0_0 [simp]: "Bessel_J 0 0 = 1"
  by (simp add: Bessel_J_def)

lemma Bessel_J_0_right [simp]: "a ≠ 0 ⟹ Bessel_J a 0 = 0"
  by (simp add: Bessel_J_def)

lemma Bessel_J_0_right': "Bessel_J a 0 = (if a = 0 then 1 else 0)"
  by (cases "a = 0") auto

lemma Bessel_J_complex_of_real:
  assumes "a ∈ ℤ ∨ z ≥ 0"
  shows   "Bessel_J (complex_of_real a) (of_real z) = of_real (Bessel_J
a z)"
  using assms unfolding Bessel_J_def
  by (auto simp: powr'_def Bessel_Clifford_complex_of_real [symmetric]
           elim!: Ints_cases simp: powr_Reals_eq)
```

```
lemma Bessel_J_contiguous_complex:
  fixes a z :: complex
  shows "2 * a * Bessel_J a z = z * (Bessel_J (a-1) z + Bessel_J (a+1)
z)"
proof (cases "z = 0")
  case False
  define a' where "a' = a - 1"
  have "z * (Bessel_J a' z + Bessel_J (a'+2) z) = 2 * (a' + 1) * Bessel_J
(a'+1) z"
    unfolding Bessel_J_def using False
    by (subst Bessel_Clifford_contiguous)
       (simp add: field_simps power2_eq_square power3_eq_cube powr_add
powr'_complex)
  thus ?thesis
    by (simp add: a'_def add_ac)
qed (auto simp: Bessel_J_0_right' add_eq_0_iff2)

lemma Bessel_J_contiguous_real:
  fixes a z :: real
  assumes "z ≥ 0 ∨ a ∈ ℤ"
  shows   "2 * a * Bessel_J a z = z * (Bessel_J (a-1) z + Bessel_J (a+1)
z)"
proof (cases "z = 0")
  case False
  define a' where "a' = a - 1"
  have [simp]: "a' ∈ ℤ ⟷ a ∈ ℤ"
    by (auto simp: a'_def)
  have "z * (Bessel_J a' z + Bessel_J (a'+2) z) = 2 * (a' + 1) * Bessel_J
(a'+1) z"
    unfolding Bessel_J_def using False assms
    by (subst Bessel_Clifford_contiguous)
       (use assms in ‹auto simp: field_simps power2_eq_square power3_eq_cube
powr_add powr'_def
         the_int_add power_int_add power_int_0_left_if add_eq_0_iff2
elim!: Ints_cases›)
  thus ?thesis
    by (simp add: a'_def add_ac)
qed (auto simp: Bessel_J_0_right' add_eq_0_iff2)

lemma Bessel_J_minus_of_nat_complex:
  "Bessel_J (-of_nat n :: complex) z = (-1) ^ n * Bessel_J (of_nat n)
z"
proof (cases "z = 0")
  case False
  thus ?thesis
    unfolding Bessel_J_def Bessel_Clifford_minus_of_nat power2_eq_square
    using power_mult_distrib[of 2 "2::complex" n]
    by (auto simp: powr'_complex powr_minus Bessel_Clifford_minus_of_nat
field_simps power_minus')
```

```
qed (auto simp: Bessel_J_0_right')

lemma Bessel_J_minus_of_int_complex:
  "Bessel_J (-of_int n :: complex) z = (-1) powi n * Bessel_J (of_int
n) z"
proof (cases "z = 0")
  case False
  show ?thesis
  proof (cases "n ≥ 0")
    case True
    thus ?thesis
      using Bessel_J_minus_of_nat_complex[of "nat n" z] by (simp add:
power_int_def)
  next
    case False
    thus ?thesis
      using Bessel_J_minus_of_nat_complex[of "nat (-n)" z] by (simp add:
power_int_def)
  qed
qed (auto simp: Bessel_J_0_right')

lemma Bessel_J_minus_of_int_real:
  "Bessel_J (-of_int n :: real) z = (-1) powi n * Bessel_J (of_int n)
z"
proof -
  have "complex_of_real (Bessel_J (-of_int n) z) = of_real ((-1) powi
n * Bessel_J (of_int n) z)"
    unfolding of_real_mult using Bessel_J_minus_of_int_complex[of n "of_real
z"]
    by (simp flip: Bessel_J_complex_of_real)
  thus ?thesis
    by (simp only: of_real_eq_iff)
qed

lemma Bessel_J_minus_of_nat_real:
  "Bessel_J (-of_nat n :: real) z = (-1) powi n * Bessel_J (of_nat n)
z"
  using Bessel_J_minus_of_int_real[of "int n" z] by simp

lemma sums_Bessel_J:
  fixes a z :: "'a :: {Gamma, ln}"
  shows  "(λn. (-1)^n * (z/2) powr' a * (z/2) ^ (2*n) / (fact n * Gamma
(1 + a + of_nat n))) sums
            Bessel_J a z"
proof -
  have "(λn. (z/2) powr' a * (rGamma (a + of_nat (Suc n)) * (-(z²/4))
^ n / fact n)) sums
          (Bessel_J a z)"
    unfolding Bessel_J_def by (intro sums_mult sums_Bessel_Clifford)
```

```
  also have "(λn. (z/2) powr' a * (rGamma (a + of_nat (Suc n)) * (-(z^2/4))
^ n / fact n)) =
             (λn. (-1)^n * (z/2) powr' a * (z/2) ^ (2*n) / (fact n * Gamma
(1 + a + of_nat n)))"
    unfolding power_mult power2_eq_square
    by (auto simp: fun_eq_iff powr_add power_minus' field_simps rGamma_inverse_Gamma)
  finally show ?thesis .
qed

lemma sums_Bessel_J_complex:
  fixes a z :: complex
  assumes "z ≠ 0 ∨ a ∉ ℤ≤0"
  shows  "(λn. (-1)^n * (z/2) powr' (a + 2 * of_nat n) / (fact n * Gamma
(1 + a + of_nat n))) sums
            Bessel_J a z"
proof -
  have "(λn. (-1)^n * (z/2) powr' a * (z/2) ^ (2*n) / (fact n * Gamma
(1 + a + of_nat n))) sums
           Bessel_J a z"
    by (rule sums_Bessel_J)
  also have "(λn. (-1)^n * (z/2) powr' a * (z/2) ^ (2*n) / (fact n * Gamma
(1 + a + of_nat n))) =
             (λn. (-1)^n * (z/2) powr' (a + 2 * of_nat n) / (fact n *
Gamma (1 + a + of_nat n)))"
    (is "?lhs = ?rhs")
  proof
    fix n :: nat
    from assms have "z ≠ 0 ∨ (z = 0 ∧ a ∉ ℤ≤0)"
      by auto
    hence "(z/2) powr' (a + 2 * of_nat n) = (z/2) powr' a * (z/2) ^ (2*n)"
    proof
      assume "z ≠ 0"
      thus ?thesis using powr_nat'[of "z/2" "2 * n"]
        by (auto simp: powr'_complex powr_add)
    next
      assume *: "z = 0 ∧ a ∉ ℤ≤0"
      hence "a + 2 * complex_of_nat n ≠ 0"
        by (metis mult_2 of_nat_add plus_of_nat_eq_0_imp)
      thus ?thesis using *
        by (auto simp: powr'_0_left_if)
    qed
    thus "?lhs n = ?rhs n"
      by simp
  qed
  finally show ?thesis .
qed

lemma sums_Bessel_J_real:
  fixes a z :: real
```

```
  assumes "a ∈ ℤ ∨ z ≥ 0" "z ≠ 0 ∨ a ∉ ℤ≤0"
  shows  "(λn. (-1)^n * (z/2) powr' (a + 2 * of_nat n) / (fact n * Gamma
(1 + a + of_nat n))) sums
           Bessel_J a z"
proof -
  have "(λn. (-1)^n * (z/2) powr' a * (z/2) ^ (2*n) / (fact n * Gamma
(1 + a + of_nat n))) sums
          Bessel_J a z"
    by (rule sums_Bessel_J)
  also have "(λn. (-1)^n * (z/2) powr' a * (z/2) ^ (2*n) / (fact n * Gamma
(1 + a + of_nat n))) =
             (λn. (-1)^n * (z/2) powr' (a + 2 * of_nat n) / (fact n *
Gamma (1 + a + of_nat n)))"
    (is "?lhs = ?rhs")
  proof
    fix n :: nat
    from assms have "z ≠ 0 ∨ (z = 0 ∧ a ∉ ℤ≤0)"
      by auto
    have "(z/2) powr' (a + 2 * of_nat n) = (z/2) powr' a * (z/2) ^ (2*n)"
    proof (cases "z = 0")
      case [simp]: True
      with assms(2) have "a + 2 * real n ≠ 0"
        by (metis mult.commute mult_2_right of_nat_add plus_of_nat_eq_0_imp)
      thus ?thesis
        using assms by (auto simp: powr'_0_left_if)
    next
      case nz: False
      show ?thesis
      proof (cases "a ∈ ℤ")
        case False
        thus ?thesis using powr_realpow[of "z/2" "2*n"] assms(1)
          using nz by (auto simp: powr'_def powr_add)
      next
        case True
        then obtain k where a_eq: "a = of_int k"
          by (auto elim!: Ints_cases)
        have "(z / 2) powi (int (2*n)) = (z / 2) ^ (2 * n)"
          by (subst power_int_of_nat) auto
        thus ?thesis using nz unfolding of_nat_mult
          by (auto simp: a_eq powr'_def the_int_add power_int_add the_int_mult)
      qed
    qed
    thus "?lhs n = ?rhs n"
      by simp
  qed
  finally show ?thesis .
qed

lemma has_field_derivative_Bessel_J_complex:
```

```
  assumes "a ∈ ℤ ∨ (z::complex) ∉ ℝ≤0"
  shows   "(Bessel_J a has_field_derivative
             ((Bessel_J (a-1) z - Bessel_J (a+1) z) / 2)) (at z within
A)"
proof (cases "a ∈ ℤ")
  case False
  with assms have "z ∉ ℝ≤0"
    by auto
  moreover from this have "z ≠ 0"
    by auto
  ultimately have "(Bessel_J a has_field_derivative
                     (a * Bessel_J a z / z - Bessel_J (a+1) z)) (at z
within A)"
    unfolding Bessel_J_def
    by (auto intro!: derivative_eq_intros
             simp: complex_nonpos_Reals_iff powr'_complex powr_add field_simps)
  also have "a * Bessel_J a z / z - Bessel_J (a+1) z = (Bessel_J (a-1)
z - Bessel_J (a+1) z) / 2"
    using Bessel_J_contiguous_complex[of a z] ‹z ≠ 0› by (auto simp:
field_simps)
  finally show ?thesis .
next
  case True
  then obtain n where a: "a = of_int n"
    by (auto elim: Ints_cases)
  have *: "(Bessel_J a has_field_derivative
           (Bessel_J (a - 1) z - Bessel_J (a + 1) z) / 2) (at z within
A)"
    if a: "a = of_int n" "n ≥ 0" for a n
  proof -
    have a': "a - 1 = of_int (n - 1)" "a + 1 = of_int (n + 1)"
      by (auto simp: a)
    have "((λz. (z / 2) powi n * Bessel_Clifford a (-(z^2/4))) has_field_derivative
           (a / 2 * (z / 2) powi (n-1) * Bessel_Clifford a (- (z² / 4))
-
           (z / 2) powi (n+1) * Bessel_Clifford (a+1) (- (z² / 4))))
(at z within A)"
      using a by (auto intro!: derivative_eq_intros simp: power_int_add)
    also have "(a / 2 * (z / 2) powi (n-1) * Bessel_Clifford a (- (z²
/ 4)) -
                 (z / 2) powi (n+1) * Bessel_Clifford (a+1) (- (z² / 4)))
=
              (Bessel_J (a-1) z - Bessel_J (a+1) z) / 2"
      unfolding Bessel_J_def a' powr'_of_int using a
      by (subst Bessel_Clifford_contiguous[of "of_int (n-1)" "-(z ^ 2
/ 4)"], cases "z = 0")
         (auto simp: powr'_complex power_int_0_left_if add_eq_0_iff2 power_int_divide_distr

                     power_int_add power_int_diff power2_eq_square field_simps)
```

```
    also have "((λz. (z / 2) powi n * Bessel_Clifford a (- (z^2 / 4)))
has_field_derivative
                (Bessel_J (a - 1) z - Bessel_J (a + 1) z) / 2) (at z
within A) ⟷
              ((λz. Bessel_J a z) has_field_derivative
                (Bessel_J (a - 1) z - Bessel_J (a + 1) z) / 2) (at z
within A)"
    proof (rule has_field_derivative_cong_eventually)
      have "eventually (λx. x ≠ 0) (at z within A)"
        by (rule eventually_neq_at_within)
      thus "∀_F x in at z within A. (x / 2) powi n * Bessel_Clifford a
(- (x^2 / 4)) = Bessel_J a x"
        unfolding Bessel_J_def by eventually_elim (auto simp: powr'_complex
a)
    next
      show "(z / 2) powi n * Bessel_Clifford a (- (z^2 / 4)) = Bessel_J
a z"
        by (auto simp: Bessel_J_def powr'_complex power_int_0_left_if
a)
    qed
    finally show ?thesis .
  qed

  show ?thesis
  proof (cases "n ≥ 0")
    case True
    thus ?thesis using *[of a n] a by simp
  next
    case False
    have "((λz. (-1) powi n * Bessel_J (-of_int n) z) has_field_derivative
            (-1) powi n * ((Bessel_J ((-of_int n) - 1) z - Bessel_J ((-of_int
n) + 1) z) / 2)) (at z within A)"
      by (rule DERIV_cmult, rule *[of "-of_int n" "-n"]) (use False in
auto)
    also have "(λz. (-1) powi n * Bessel_J (-of_int n) z) = Bessel_J a"
      unfolding a by (subst Bessel_J_minus_of_int_complex) auto
    also have "(-1) powi n * ((Bessel_J ((-of_int n) - 1) z - Bessel_J
((-of_int n) + 1) z) / 2) =
               (-1) powi n * ((Bessel_J (-(of_int (n+1))) z - Bessel_J
(-(of_int (n-1))) z) / 2)"
      by simp
    also have "... = (Bessel_J (a-1) z - Bessel_J (a+1) z) / 2"
      unfolding a
      by (subst (1 2) Bessel_J_minus_of_int_complex) (auto simp: power_int_add
power_int_minus_left)
    finally show ?thesis .
  qed
qed
```

```
lemma has_field_derivative_Bessel_J_complex' [derivative_intros]:
  assumes "(f has_field_derivative f') (at x within A)" "a ∈ ℤ ∨ (f
x :: complex) ∉ ℝ≤0"
  shows   "((λx. Bessel_J a (f x)) has_field_derivative
             (f' * (Bessel_J (a-1) (f x) - Bessel_J (a+1) (f x)) / 2))
(at x within A)"
  using DERIV_chain[OF has_field_derivative_Bessel_J_complex[of a] assms(1)]
assms(2)
  by (simp add: o_def mult_ac)

lemma has_field_derivative_Bessel_J_real:
  assumes "a ∈ ℤ ∨ (z :: real) > 0"
  shows   "(Bessel_J a has_field_derivative
             ((Bessel_J (a-1) z - Bessel_J (a+1) z) / 2)) (at z within
A)"
proof -
  have *: "complex_of_real a ∈ ℤ ∨ complex_of_real z ∉ ℝ≤0"
    using assms by auto
  have "((λx. Re (Bessel_J (of_real a) (of_real x))) has_field_derivative
          (Re (Bessel_J (of_real (a - 1)) (of_real z)) -
           Re (Bessel_J (of_real (a + 1)) (of_real z))) / 2) (at z within
A)"
    by (rule derivative_eq_intros has_vector_derivative_real_field refl
*)+ simp_all
  also have "(Re (Bessel_J (of_real (a - 1)) (of_real z)) -
              Re (Bessel_J (of_real (a + 1)) (of_real z))) / 2 =
             (Bessel_J (a-1) z - Bessel_J (a+1) z) / 2"
    by (subst (1 2) Bessel_J_complex_of_real) (use assms in auto)
  also have "((λx. Re (Bessel_J (of_real a) (of_real x))) has_real_derivative
               (Bessel_J (a - 1) z - Bessel_J (a + 1) z) / 2) (at z within
A) ⟷ ?thesis"
  proof (rule has_field_derivative_cong_eventually)
    have "eventually (λx. x ∈ (if a ∈ ℤ then UNIV else {0<..})) (nhds
z)"
      by (rule eventually_nhds_in_open) (use assms in auto)
    hence ev: "eventually (λx. Re (Bessel_J (of_real a) (of_real x)) =
Bessel_J a x) (nhds z)"
      by eventually_elim (auto simp: Bessel_J_complex_of_real split: if_splits)
    show "eventually (λx. Re (Bessel_J (of_real a) (of_real x)) = Bessel_J
a x) (at z within A)"
      using ev by (simp add: eventually_at_filter eventually_mono)
    show "Re (Bessel_J (complex_of_real a) (complex_of_real z)) = Bessel_J
a z"
      using ev by (meson eventually_nhds)
  qed
  finally show ?thesis .
qed

lemma has_field_derivative_Bessel_J_real' [derivative_intros]:
```

```
  assumes "(f has_field_derivative f') (at x within A)" "a ∈ ℤ ∨ f x
> (0 :: real)"
  shows   "((λx. Bessel_J a (f x)) has_field_derivative
             ((Bessel_J (a-1) (f x) - Bessel_J (a+1) (f x)) / 2) * f')
(at x within A)"
  using DERIV_chain[OF has_field_derivative_Bessel_J_real assms(1), of
a] assms
  by (simp add: o_def mult_ac)

lemma analytic_Bessel_J [analytic_intros]:
  assumes "a analytic_on A" "b analytic_on A" "⋀x. x ∈ A ⟹ b x ∉
ℝ≤0"
  shows   "(λx. Bessel_J (a x) (b x)) analytic_on A"
  unfolding Bessel_J_def
  by (auto intro!: analytic_intros assms(1,2))
     (use assms(3) in ‹auto simp: complex_nonpos_Reals_iff›)

lemma analytic_Bessel_J_Ints:
  assumes "b analytic_on A" "a ∈ ℤ"
  shows   "(λx. Bessel_J a (b x)) analytic_on A"
proof -
  from ‹a ∈ ℤ› obtain n where a: "a = of_int n"
    by (elim Ints_cases)
  have *: "(λx. Bessel_J (of_int n) (b x)) analytic_on A" if n: "n ≥
0" for n
    unfolding Bessel_J_def powr'_of_int by (intro analytic_intros assms(1))
(use n in auto)
  show ?thesis
  proof (cases "n ≥ 0")
    case True
    thus ?thesis using *[of n] by (simp add: a)
  next
    case False
    note [analytic_intros del] = analytic_Bessel_J
    have "(λx. (-1) powi n * Bessel_J (of_int (-n)) (b x)) analytic_on
A"
      by (intro analytic_intros *) (use False in auto)
    also have "(λx. (-1) powi n * Bessel_J (of_int (-n)) (b x)) =
               (λx. Bessel_J (of_int n) (b x))"
      unfolding of_int_minus by (subst Bessel_J_minus_of_int_complex)
(simp flip: mult.assoc)
    finally show ?thesis by (simp add: a)
  qed
qed
```

The half-integer case $J_{n+\frac{1}{2}}$ can be expressed in terms of spherical Bessel functions of the first kind j_n and therefore has a closed form.

```
theorem Bessel_J_conv_SBessel_J_complex:
  assumes z: "z ≠ (0 :: complex)"
```

```
  defines "C ≡ sqrt (2 / pi) * csqrt z"
  shows   "Bessel_J (of_int n + 1 / 2) z = C * SBessel_J n z"
proof (induction n rule: SBessel_J_induct)
  case 1
  have "Bessel_J (of_int 0 + 1 / 2) z =
          (z / 2) powr' (1 / 2) * rGamma (3/2) * hypergeo_F [] [3 / 2]
(- (z^2 / 4))"
    unfolding Bessel_J_def by (subst Bessel_Clifford_conv_hypergeo_F)
auto
  also have "rGamma (3 / 2 :: complex) = 2 * rGamma (1/2)"
    using rGamma_plus1[of "1/2 :: complex"] by simp
  also have "... = of_real (2 / sqrt pi)"
    unfolding rGamma_inverse_Gamma Gamma_one_half_complex by (simp add:
field_simps)
  also have "(z / 2) powr' (1 / 2) = ((1/2) * z) powr (1 / 2)"
    by (auto simp: powr'_def)
  also have "... = csqrt z / sqrt 2"
    by (subst powr_times_real_left)
       (auto simp: powr_half_sqrt powr_Reals_eq real_sqrt_divide simp
flip: csqrt_conv_powr)
  also have "hypergeo_F [] [3/2] (-(z^2/4)) = sin z / z"
    using sin_conv_hypergeo_F[of z] z by simp
  also have "sin z / z = SBessel_J 0 z"
    by simp
  also have "csqrt z / complex_of_real (sqrt 2) * complex_of_real (2 /
sqrt pi) =
             C"
    using z by (simp add: C_def real_sqrt_divide field_simps flip: power2_eq_square
of_real_power)
  finally show ?case
    by simp
next
  case 2
  have "Bessel_J (of_int (-1) + 1 / 2) z =
          (z / 2) powr' (-1 / 2) * rGamma (1/2) * hypergeo_F [] [1 / 2]
(- (z^2 / 4))"
    unfolding Bessel_J_def by (subst Bessel_Clifford_conv_hypergeo_F)
auto
  also have "rGamma (1 / 2 :: complex) = of_real (1 / sqrt pi)"
    unfolding rGamma_inverse_Gamma Gamma_one_half_complex by (simp add:
field_simps)
  also have "(z / 2) powr' (-1 / 2) = ((1/2) * z) powr (-1 / 2)"
    by (auto simp: powr'_def)
  also have "... = sqrt 2 * (1 / csqrt z)"
    by (subst powr_times_real_left)
       (auto simp: powr_minus powr_half_sqrt powr_Reals_eq real_sqrt_divide
field_simps
             simp flip: csqrt_conv_powr)
  also have "1 / csqrt z = csqrt z / z"
```

```
    using z by (simp add: field_simps flip: power2_eq_square)
  also have "hypergeo_F [] [1/2] (-(z^2/4)) = cos z"
    using cos_conv_hypergeo_F[of z] z by simp
  also have "of_real (sqrt 2) * (csqrt z / z) * of_real (1 / sqrt pi)
* cos z =
             of_real (sqrt (2 / pi)) * csqrt z * (cos z / z)"
    using z by (simp add: field_simps real_sqrt_divide)
  also have "cos z / z = SBessel_J (-1) z"
    by simp
  finally show ?case
    by (simp add: C_def)
next
  case (3 n)
  have "Bessel_J (of_int n + 1 / 2) z =
          2 * (of_int n - 1 / 2) / z * Bessel_J (of_int (n - 1) + 1 /
2) z -
          Bessel_J (of_int (n - 2) + 1 / 2) z"
    using Bessel_J_contiguous_complex[of "of_int n - 1 / 2" z] using z
    by (simp add: field_simps)
  also have "2 * (of_int n - 1 / 2) = 2 * complex_of_int n - 1"
    by (simp add: field_simps)
  also have "Bessel_J (of_int (n - 1) + 1 / 2) z = C * SBessel_J (n -
1) z"
    by (subst "3.IH") auto
  also have "Bessel_J (of_int (n - 2) + 1 / 2) z = C * SBessel_J (n -
2) z"
    by (subst "3.IH") auto
  also have "(2 * of_int n - 1) / z * (C * SBessel_J (n - 1) z) - C *
SBessel_J (n - 2) z =
               C * SBessel_J n z"
    using "3.hyps" z by (subst (3) SBessel_J.simps) (auto simp: field_simps)
  finally show ?case .
next
  case (4 n)
  have "Bessel_J (of_int n + 1 / 2) z =
          2 * (of_int n + 3 / 2) / z * Bessel_J (of_int (n + 1) + 1 /
2) z -
          Bessel_J (of_int (n + 2) + 1 / 2) z"
    using Bessel_J_contiguous_complex[of "of_int n + 3 / 2" z] using z
    by (simp add: field_simps)
  also have "2 * (of_int n + 3 / 2) = 2 * complex_of_int n + 3"
    by (simp add: field_simps)
  also have "Bessel_J (of_int (n + 1) + 1 / 2) z = C * SBessel_J (n +
1) z"
    by (subst "4.IH") auto
  also have "Bessel_J (of_int (n + 2) + 1 / 2) z = C * SBessel_J (n +
2) z"
    by (subst "4.IH") auto
  also have "(2 * of_int n + 3) / z * (C * SBessel_J (n + 1) z) - C *
```

```
SBessel_J (n + 2) z =
            C * ((2 * of_int n + 3) / z * SBessel_J (n + 1) z - SBessel_J
(n + 2) z)"
    using z by (simp add: field_simps)
  also have "(2 * of_int n + 3) / z * SBessel_J (n + 1) z - SBessel_J
(n + 2) z = SBessel_J n z"
    by (subst (2) SBessel_J.simps) (use "4.hyps" in ‹simp_all add: add_ac›)
  finally show ?case .
qed

lemma Bessel_J_conv_SBessel_J_real:
  assumes z: "z > (0 :: real)"
  shows   "Bessel_J (of_int n + 1 / 2) z = sqrt (2 * z / pi) * SBessel_J
n z"
proof -
  have "complex_of_real (Bessel_J (of_int n + 1 / 2) z) = Bessel_J (of_int
n + 1 / 2) (of_real z)"
    by (subst Bessel_J_complex_of_real [symmetric]) (use z in auto)
  also have "Bessel_J (of_int n + 1 / 2) (complex_of_real z) =
              of_real (sqrt (2 * z / pi) * SBessel_J n z)"
    by (subst Bessel_J_conv_SBessel_J_complex)
       (use z in ‹auto simp: SBessel_J_of_real real_sqrt_divide real_sqrt_mult›)
  finally show ?thesis
    by (simp only: of_real_eq_iff)
qed
```

1.8 The modified Bessel function of the first kind

Again, the modified Bessel function of the first kind I_a is essentially the hyperbolic version of J_a. In the complex case, it can also easily be written in terms of I_a and vice versa.

```
definition Bessel_I :: "'a :: {Gamma, ln} ⇒ 'a ⇒ 'a" where
  "Bessel_I a z = (z / 2) powr' a * Bessel_Clifford a (z^2/4)"

lemma Bessel_I_0_0 [simp]: "Bessel_I 0 0 = 1"
  by (simp add: Bessel_I_def)

lemma Bessel_I_0_right [simp]: "a ≠ 0 ⟹ Bessel_I a 0 = 0"
  by (simp add: Bessel_I_def)

lemma Bessel_I_0_right': "Bessel_I a 0 = (if a = 0 then 1 else 0)"
  by (cases "a = 0") auto

lemma Bessel_I_complex_of_real:
  assumes "a ∈ ℤ ∨ z ≥ 0"
  shows   "Bessel_I (complex_of_real a) (of_real z) = of_real (Bessel_I
a z)"
  using assms unfolding Bessel_I_def
  by (auto simp: powr'_def Bessel_Clifford_complex_of_real [symmetric]
```

```
          elim!: Ints_cases simp: powr_Reals_eq)

lemma Bessel_I_contiguous_complex:
  fixes a z :: complex
  shows "z * Bessel_I a z = 2 * (a + 1) * Bessel_I (a+1) z + z * Bessel_I
(a+2) z"
proof (cases "z = 0")
  case False
  show "z * Bessel_I a z = 2 * (a + 1) * Bessel_I (a+1) z + z * Bessel_I
(a+2) z"
    unfolding Bessel_I_def using False
    by (subst Bessel_Clifford_contiguous)
       (simp add: field_simps power2_eq_square power3_eq_cube powr_add
powr'_complex)
qed (auto simp: Bessel_I_0_right' add_eq_0_iff2)

lemma Bessel_I_contiguous_real:
  fixes a z :: real
  assumes "z ≥ 0 ∨ a ∈ ℤ"
  shows   "z * Bessel_I a z = 2 * (a + 1) * Bessel_I (a+1) z + z * Bessel_I
(a+2) z"
  unfolding Bessel_I_def
  by (subst Bessel_Clifford_contiguous; cases "z = 0")
     (use assms in ‹auto simp: field_simps power2_eq_square power3_eq_cube
powr_add powr'_def
       the_int_add power_int_add power_int_0_left_if add_eq_0_iff2 elim!:
Ints_cases›)

lemma Bessel_I_minus_of_nat_complex:
  "Bessel_I (-of_nat n :: complex) z = Bessel_I (of_nat n) z"
proof (cases "z = 0")
  case False
  thus ?thesis
    unfolding Bessel_I_def Bessel_Clifford_minus_of_nat power2_eq_square
    using power_mult_distrib[of 2 "2::complex" n] False
    by (auto simp: powr'_complex powr_minus Bessel_Clifford_minus_of_nat
field_simps power_minus')
qed (auto simp: Bessel_I_0_right')

lemma Bessel_I_minus_of_int_complex:
  "Bessel_I (-of_int n :: complex) z = Bessel_I (of_int n) z"
proof (cases "z = 0")
  case False
  show ?thesis
  proof (cases "n ≥ 0")
    case True
    thus ?thesis
      using Bessel_I_minus_of_nat_complex[of "nat n" z] by (simp add:
power_int_def)
```

```
  next
    case False
    thus ?thesis
      using Bessel_I_minus_of_nat_complex[of "nat (-n)" z] by (simp add:
power_int_def)
  qed
qed (auto simp: Bessel_I_0_right')

lemma Bessel_I_minus_of_int_real:
  "Bessel_I (-of_int n :: real) z = Bessel_I (of_int n) z"
proof -
  have "complex_of_real (Bessel_I (-of_int n) z) = of_real (Bessel_I (of_int
n) z)"
    unfolding of_real_mult using Bessel_I_minus_of_int_complex[of n "of_real
z"]
    by (simp flip: Bessel_I_complex_of_real)
  thus ?thesis
    by (simp only: of_real_eq_iff)
qed

lemma Bessel_I_minus_of_nat_real:
  "Bessel_I (-of_nat n :: real) z = Bessel_I (of_nat n) z"
  using Bessel_I_minus_of_int_real[of "int n" z] by simp

lemma Bessel_I_conv_J:
  assumes "a ∈ ℤ ∨ Re z ≥ 0 ∨ Im z < 0"
  shows "Bessel_I a z = i powr (-a) * Bessel_J a (i * z)"
proof (cases "z = 0")
  case [simp]: True
  show ?thesis
    by (cases "a = 0") (auto simp: Bessel_I_def Bessel_J_def)
next
  case z: False
  have "i powr -a * (i * (z / 2)) powr' a = (z / 2) powr' a"
    using assms
  proof
    assume "a ∈ ℤ"
    then obtain n where a: "a = of_int n"
      by (auto elim!: Ints_cases)
    show ?thesis
      by (simp add: a powr_minus complex_powr_of_int power_int_divide_distrib

                    power_int_mult_distrib field_simps)
  next
    assume z': "Re z ≥ 0 ∨ Im z < 0"
    have "(i * (z / 2)) powr' a = (i * (z / 2)) powr a"
      using z by (subst powr'_complex) auto
    also have "... = exp (Ln (i * (z / 2)) * a)"
      using z by (simp add: powr_def mult_ac)
```

```
    also have "Ln (i * (z / 2)) = Ln i + Ln (z / 2)"
      by (subst Ln_times_ii) (use z' z in <auto simp: not_le not_less
mult_ac>)
    also have "exp (... * a) = exp (Ln i * a) * exp (Ln (z / 2) * a)"
      unfolding exp_add ring_distribs by simp
    also have "exp (Ln (z / 2) * a) = (z / 2) powr a"
      using z by (auto simp: powr_def mult_ac)
    also have "... = (z / 2) powr' a"
      using z by (subst powr'_complex) auto
    also have "exp (Ln i * a) = i powr a"
      by (auto simp: powr_def mult_ac)
    finally show ?thesis
      by (simp add: powr_minus field_simps)
  qed
  thus ?thesis
    using z by (simp add: Bessel_I_def Bessel_J_def power_mult_distrib
powr'_complex powr_minus)
qed

lemma Bessel_J_conv_I:
  assumes "a ∈ ℤ ∨ Re x > 0 ∨ Im x ≥ 0"
  shows "Bessel_J a x = i powr a * Bessel_I a (-i * x)"
  by (subst Bessel_I_conv_J) (use assms in <auto simp: powr_minus>)

lemma sums_Bessel_I:
  fixes a z :: "'a :: {Gamma, ln}"
  shows  "(λn. (z/2) powr' a * (z/2) ^ (2*n) / (fact n * Gamma (1 + a
+ of_nat n))) sums
            Bessel_I a z"
proof -
  have "(λn. (z/2) powr' a * (rGamma (a + of_nat (Suc n)) * (z^2/4) ^ n
/ fact n)) sums
          (Bessel_I a z)"
    unfolding Bessel_I_def by (intro sums_mult sums_Bessel_Clifford)
  also have "(λn. (z/2) powr' a * (rGamma (a + of_nat (Suc n)) * (z^2/4)
^ n / fact n)) =
             (λn. (z/2) powr' a * (z/2) ^ (2*n) / (fact n * Gamma (1 +
a + of_nat n)))"
    unfolding power_mult power2_eq_square
    by (auto simp: fun_eq_iff powr_add power_minus' field_simps rGamma_inverse_Gamma)
  finally show ?thesis .
qed

lemma sums_Bessel_I_complex:
  fixes a z :: complex
  assumes "z ≠ 0 ∨ a ∉ ℤ≤0"
  shows  "(λn. (z/2) powr' (a + 2 * of_nat n) / (fact n * Gamma (1 +
a + of_nat n))) sums
            Bessel_I a z"
```

```
proof -
  have "(λn. (z/2) powr' a * (z/2) ^ (2*n) / (fact n * Gamma (1 + a +
of_nat n))) sums
           Bessel_I a z"
    by (rule sums_Bessel_I)
  also have "(λn. (z/2) powr' a * (z/2) ^ (2*n) / (fact n * Gamma (1 +
a + of_nat n))) =
             (λn. (z/2) powr' (a + 2 * of_nat n) / (fact n * Gamma (1
+ a + of_nat n)))"
    (is "?lhs = ?rhs")
  proof
    fix n :: nat
    from assms have "z ≠ 0 ∨ (z = 0 ∧ a ∉ ℤ≤0)"
      by auto
    hence "(z/2) powr' (a + 2 * of_nat n) = (z/2) powr' a * (z/2) ^ (2*n)"
    proof
      assume "z ≠ 0"
      thus ?thesis using powr_nat'[of "z/2" "2 * n"]
        by (auto simp: powr'_complex powr_add)
    next
      assume *: "z = 0 ∧ a ∉ ℤ≤0"
      hence "a + 2 * complex_of_nat n ≠ 0"
        by (metis mult_2 of_nat_add plus_of_nat_eq_0_imp)
      thus ?thesis using *
        by (auto simp: powr'_0_left_if)
    qed
    thus "?lhs n = ?rhs n"
      by simp
  qed
  finally show ?thesis .
qed

lemma sums_Bessel_I_real:
  fixes a z :: real
  assumes "a ∈ ℤ ∨ z ≥ 0" "z ≠ 0 ∨ a ∉ ℤ≤0"
  shows  "(λn. (z/2) powr' (a + 2 * of_nat n) / (fact n * Gamma (1 +
a + of_nat n))) sums
            Bessel_I a z"
proof -
  have "(λn. (z/2) powr' a * (z/2) ^ (2*n) / (fact n * Gamma (1 + a +
of_nat n))) sums
           Bessel_I a z"
    by (rule sums_Bessel_I)
  also have "(λn. (z/2) powr' a * (z/2) ^ (2*n) / (fact n * Gamma (1 +
a + of_nat n))) =
             (λn. (z/2) powr' (a + 2 * of_nat n) / (fact n * Gamma (1
+ a + of_nat n)))"
    (is "?lhs = ?rhs")
  proof
```

```
    fix n :: nat
    from assms have "z ≠ 0 ∨ (z = 0 ∧ a ∉ ℤ_{≤0})"
      by auto
    have "(z/2) powr' (a + 2 * of_nat n) = (z/2) powr' a * (z/2) ^ (2*n)"
    proof (cases "z = 0")
      case [simp]: True
      with assms(2) have "a + 2 * real n ≠ 0"
        by (metis mult.commute mult_2_right of_nat_add plus_of_nat_eq_0_imp)
      thus ?thesis
        using assms by (auto simp: powr'_0_left_if)
    next
      case nz: False
      show ?thesis
      proof (cases "a ∈ ℤ")
        case False
        thus ?thesis using powr_realpow[of "z/2" "2*n"] assms(1)
          using nz by (auto simp: powr'_def powr_add)
      next
        case True
        then obtain k where a_eq: "a = of_int k"
          by (auto elim!: Ints_cases)
        have "(z / 2) powi (int (2*n)) = (z / 2) ^ (2 * n)"
          by (subst power_int_of_nat) auto
        thus ?thesis using nz unfolding of_nat_mult
          by (auto simp: a_eq powr'_def the_int_add power_int_add the_int_mult)
      qed
    qed
    thus "?lhs n = ?rhs n"
      by simp
  qed
  finally show ?thesis .
qed

lemma has_field_derivative_Bessel_I_complex:
  assumes "a ∈ ℤ ∨ (z::complex) ∉ ℝ_{≤0}"
  shows   "(Bessel_I a has_field_derivative
             ((Bessel_I (a-1) z + Bessel_I (a+1) z) / 2)) (at z within
A)"
proof (cases "a ∈ ℤ")
  case False
  with assms have "z ∉ ℝ_{≤0}"
    by auto
  moreover from this have "z ≠ 0"
    by auto
  ultimately have "(Bessel_I a has_field_derivative
                    (a * Bessel_I a z / z + Bessel_I (a+1) z)) (at z
within A)"
    unfolding Bessel_I_def
    by (auto intro!: derivative_eq_intros
```

```
            simp: complex_nonpos_Reals_iff powr'_complex powr_add field_simps)
  also have "a * Bessel_I a z / z + Bessel_I (a+1) z = (Bessel_I (a-1)
z + Bessel_I (a+1) z) / 2"
    using Bessel_I_contiguous_complex[of z "a-1"] ‹z ≠ 0› by (auto simp:
field_simps)
  finally show ?thesis .
next
  case True
  then obtain n where a: "a = of_int n"
    by (auto elim: Ints_cases)
  have *: "(Bessel_I a has_field_derivative
          (Bessel_I (a - 1) z + Bessel_I (a + 1) z) / 2) (at z within
A)"
    if a: "a = of_int n" "n ≥ 0" for a n
  proof -
    have a': "a - 1 = of_int (n - 1)" "a + 1 = of_int (n + 1)"
      by (auto simp: a)
    have "((λz. (z / 2) powi n * Bessel_Clifford a (z^2/4)) has_field_derivative
            (a / 2 * (z / 2) powi (n-1) * Bessel_Clifford a (z^2 / 4) +
            (z / 2) powi (n+1) * Bessel_Clifford (a+1) (z^2 / 4))) (at
z within A)"
      using a by (auto intro!: derivative_eq_intros simp: power_int_add)
    also have "(a / 2 * (z / 2) powi (n-1) * Bessel_Clifford a (z^2 / 4)
+
                (z / 2) powi (n+1) * Bessel_Clifford (a+1) (z^2 / 4))
=
              (Bessel_I (a-1) z + Bessel_I (a+1) z) / 2"
      unfolding Bessel_I_def a' powr'_of_int using a
      by (subst Bessel_Clifford_contiguous[of "of_int (n-1)" "z ^ 2 /
4"], cases "z = 0")
         (auto simp: powr'_complex power_int_0_left_if add_eq_0_iff2 power_int_divide_distr

                     power_int_add power_int_diff power2_eq_square field_simps)
    also have "((λz. (z / 2) powi n * Bessel_Clifford a (z^2 / 4)) has_field_derivative
                (Bessel_I (a - 1) z + Bessel_I (a + 1) z) / 2) (at z
within A) ⟷
              ((λz. Bessel_I a z) has_field_derivative
                (Bessel_I (a - 1) z + Bessel_I (a + 1) z) / 2) (at z
within A)"
    proof (rule has_field_derivative_cong_eventually)
      have "eventually (λx. x ≠ 0) (at z within A)"
        by (rule eventually_neq_at_within)
      thus "∀F x in at z within A. (x / 2) powi n * Bessel_Clifford a
(x^2 / 4) = Bessel_I a x"
        unfolding Bessel_I_def by eventually_elim (auto simp: powr'_complex
a)
    next
      show "(z / 2) powi n * Bessel_Clifford a (z^2 / 4) = Bessel_I a
z"
```

```
      by (auto simp: Bessel_I_def powr'_complex power_int_0_left_if
a)
    qed
    finally show ?thesis .
  qed

  show ?thesis
  proof (cases "n ≥ 0")
    case True
    thus ?thesis using *[of a n] a by simp
  next
    case False
    have "((λz. Bessel_I (-of_int n) z) has_field_derivative
           ((Bessel_I ((-of_int n) - 1) z + Bessel_I ((-of_int n) + 1)
z) / 2)) (at z within A)"
      by (rule *[of "-of_int n" "-n"]) (use False in auto)
    also have "(λz. Bessel_I (-of_int n) z) = Bessel_I a"
      unfolding a by (subst Bessel_I_minus_of_int_complex) auto
    also have "((Bessel_I ((-of_int n) - 1) z + Bessel_I ((-of_int n)
+ 1) z) / 2) =
               ((Bessel_I (-(of_int (n+1))) z + Bessel_I (-(of_int (n-1)))
z) / 2)"
      by simp
    also have "... = (Bessel_I (a-1) z + Bessel_I (a+1) z) / 2"
      unfolding a
      by (subst (1 2) Bessel_I_minus_of_int_complex) (auto simp: power_int_add
power_int_minus_left)
    finally show ?thesis .
  qed
qed

lemma has_field_derivative_Bessel_I_complex' [derivative_intros]:
  assumes "(f has_field_derivative f') (at x within A)" "a ∈ ℤ ∨ (f
x :: complex) ∉ ℝ≤0"
  shows   "((λx. Bessel_I a (f x)) has_field_derivative
             (f' * (Bessel_I (a-1) (f x) + Bessel_I (a+1) (f x)) / 2))
(at x within A)"
  using DERIV_chain[OF has_field_derivative_Bessel_I_complex[of a] assms(1)]
assms(2)
  by (simp add: o_def mult_ac)

lemma has_field_derivative_Bessel_I_real:
  assumes "a ∈ ℤ ∨ (z :: real) > 0"
  shows   "(Bessel_I a has_field_derivative
             ((Bessel_I (a-1) z + Bessel_I (a+1) z) / 2)) (at z within
A)"
proof -
  have *: "complex_of_real a ∈ ℤ ∨ complex_of_real z ∉ ℝ≤0"
    using assms by auto
```

```
  have "((λx. Re (Bessel_I (of_real a) (of_real x))) has_field_derivative
          (Re (Bessel_I (of_real (a - 1)) (of_real z)) +
           Re (Bessel_I (of_real (a + 1)) (of_real z))) / 2) (at z within
A)"
    by (rule derivative_eq_intros has_vector_derivative_real_field refl
*)+ simp_all
  also have "(Re (Bessel_I (of_real (a - 1)) (of_real z)) +
              Re (Bessel_I (of_real (a + 1)) (of_real z))) / 2 =
             (Bessel_I (a-1) z + Bessel_I (a+1) z) / 2"
    by (subst (1 2) Bessel_I_complex_of_real) (use assms in auto)
  also have "((λx. Re (Bessel_I (of_real a) (of_real x))) has_real_derivative
               (Bessel_I (a - 1) z + Bessel_I (a + 1) z) / 2) (at z within
A) ⟷ ?thesis"
  proof (rule has_field_derivative_cong_eventually)
    have "eventually (λx. x ∈ (if a ∈ ℤ then UNIV else {0<..})) (nhds
z)"
      by (rule eventually_nhds_in_open) (use assms in auto)
    hence ev: "eventually (λx. Re (Bessel_I (of_real a) (of_real x)) =
Bessel_I a x) (nhds z)"
      by eventually_elim (auto simp: Bessel_I_complex_of_real split: if_splits)
    show "eventually (λx. Re (Bessel_I (of_real a) (of_real x)) = Bessel_I
a x) (at z within A)"
      using ev by (simp add: eventually_at_filter eventually_mono)
    show "Re (Bessel_I (complex_of_real a) (complex_of_real z)) = Bessel_I
a z"
      using ev by (meson eventually_nhds)
  qed
  finally show ?thesis .
qed

lemma has_field_derivative_Bessel_I_real' [derivative_intros]:
  assumes "(f has_field_derivative f') (at x within A)" "a ∈ ℤ ∨ f x
> (0 :: real)"
  shows   "((λx. Bessel_I a (f x)) has_field_derivative
             ((Bessel_I (a-1) (f x) + Bessel_I (a+1) (f x)) / 2) * f')
(at x within A)"
  using DERIV_chain[OF has_field_derivative_Bessel_I_real assms(1), of
a] assms
  by (simp add: o_def mult_ac)

lemma analytic_Bessel_I [analytic_intros]:
  assumes "a analytic_on A" "b analytic_on A" "⋀x. x ∈ A ⟹ b x ∉
ℝ≤0"
  shows   "(λx. Bessel_I (a x) (b x)) analytic_on A"
  unfolding Bessel_I_def
  by (auto intro!: analytic_intros assms(1,2))
     (use assms(3) in ‹auto simp: complex_nonpos_Reals_iff›)

lemma analytic_Bessel_I_Ints:
```

```
  assumes "b analytic_on A" "a ∈ ℤ"
  shows   "(λx. Bessel_I a (b x)) analytic_on A"
proof -
  from ‹a ∈ ℤ› obtain n where a: "a = of_int n"
    by (elim Ints_cases)
  have *: "(λx. Bessel_I (of_int n) (b x)) analytic_on A" if n: "n ≥
0" for n
    unfolding Bessel_I_def powr'_of_int by (intro analytic_intros assms(1))
(use n in auto)
  show ?thesis
  proof (cases "n ≥ 0")
    case True
    thus ?thesis using *[of n] by (simp add: a)
  next
    case False
    note [analytic_intros del] = analytic_Bessel_I
    have "(λx. Bessel_I (of_int (-n)) (b x)) analytic_on A"
      by (intro analytic_intros *) (use False in auto)
    also have "(λx. Bessel_I (of_int (-n)) (b x)) = (λx. Bessel_I (of_int
n) (b x))"
      unfolding of_int_minus by (subst Bessel_I_minus_of_int_complex)
(simp flip: mult.assoc)
    finally show ?thesis by (simp add: a)
  qed
qed

theorem Bessel_I_conv_SBessel_I_complex:
  assumes z: "z ≠ (0 :: complex)"
  defines "C ≡ sqrt (2 / pi) * csqrt z"
  shows   "Bessel_I (of_int n + 1 / 2) z = C * SBessel_I n z"
proof (induction n rule: SBessel_J_induct)
  case 1
  have "Bessel_I (of_int 0 + 1 / 2) z =
          (z / 2) powr' (1 / 2) * rGamma (3/2) * hypergeo_F [] [3 / 2]
(z^2 / 4)"
    unfolding Bessel_I_def by (subst Bessel_Clifford_conv_hypergeo_F)
auto
  also have "rGamma (3 / 2 :: complex) = 2 * rGamma (1/2)"
    using rGamma_plus1[of "1/2 :: complex"] by simp
  also have "... = of_real (2 / sqrt pi)"
    unfolding rGamma_inverse_Gamma Gamma_one_half_complex by (simp add:
field_simps)
  also have "(z / 2) powr' (1 / 2) = ((1/2) * z) powr (1 / 2)"
    by (auto simp: powr'_def)
  also have "... = csqrt z / sqrt 2"
    by (subst powr_times_real_left)
       (auto simp: powr_half_sqrt powr_Reals_eq real_sqrt_divide simp
flip: csqrt_conv_powr)
  also have "hypergeo_F [] [3/2] (z^2/4) = sinh z / z"
```

```
    using sinh_conv_hypergeo_F[of z] z by simp
  also have "sinh z / z = SBessel_I 0 z"
    by simp
  also have "csqrt z / complex_of_real (sqrt 2) * complex_of_real (2 /
sqrt pi) =
             C"
    using z by (simp add: C_def real_sqrt_divide field_simps flip: power2_eq_square
of_real_power)
  finally show ?case
    by simp
next
  case 2
  have "Bessel_I (of_int (-1) + 1 / 2) z =
          (z / 2) powr' (-1 / 2) * rGamma (1/2) * hypergeo_F [] [1 / 2]
(z^2 / 4)"
    unfolding Bessel_I_def by (subst Bessel_Clifford_conv_hypergeo_F)
auto
  also have "rGamma (1 / 2 :: complex) = of_real (1 / sqrt pi)"
    unfolding rGamma_inverse_Gamma Gamma_one_half_complex by (simp add:
field_simps)
  also have "(z / 2) powr' (-1 / 2) = ((1/2) * z) powr (-1 / 2)"
    by (auto simp: powr'_def)
  also have "... = sqrt 2 * (1 / csqrt z)"
    by (subst powr_times_real_left)
       (auto simp: powr_minus powr_half_sqrt powr_Reals_eq real_sqrt_divide
field_simps
             simp flip: csqrt_conv_powr)
  also have "1 / csqrt z = csqrt z / z"
    using z by (simp add: field_simps flip: power2_eq_square)
  also have "hypergeo_F [] [1/2] (z^2/4) = cosh z"
    using cosh_conv_hypergeo_F[of z] z by simp
  also have "of_real (sqrt 2) * (csqrt z / z) * of_real (1 / sqrt pi)
* cosh z =
             of_real (sqrt (2 / pi)) * csqrt z * (cosh z / z)"
    using z by (simp add: field_simps real_sqrt_divide)
  also have "cosh z / z = SBessel_I (-1) z"
    by simp
  finally show ?case
    by (simp add: C_def)
next
  case (3 n)
  have "Bessel_I (of_int n + 1 / 2) z =
          -2 * (of_int n - 1 / 2) / z * Bessel_I (of_int (n - 1) + 1 /
2) z +
          Bessel_I (of_int (n - 2) + 1 / 2) z"
    using Bessel_I_contiguous_complex[of z "of_int n - 3 / 2"] using z
    by (simp add: field_simps)
  also have "-2 * (of_int n - 1 / 2) = -2 * complex_of_int n + 1"
    by (simp add: field_simps)
```

```
  also have "Bessel_I (of_int (n - 1) + 1 / 2) z = C * SBessel_I (n -
1) z"
    by (subst "3.IH") auto
  also have "Bessel_I (of_int (n - 2) + 1 / 2) z = C * SBessel_I (n -
2) z"
    by (subst "3.IH") auto
  also have "(-2 * of_int n + 1) / z * (C * SBessel_I (n - 1) z) + C *
SBessel_I (n - 2) z =
              C * SBessel_I n z"
    using "3.hyps" z by (subst (3) SBessel_I.simps) (auto simp: field_simps)
  finally show ?case .
next
  case (4 n)
  have "Bessel_I (of_int n + 1 / 2) z =
          2 * (of_int n + 3 / 2) / z * Bessel_I (of_int (n + 1) + 1 /
2) z +
          Bessel_I (of_int (n + 2) + 1 / 2) z"
    using Bessel_I_contiguous_complex[of z "of_int n + 1 / 2"] using z
    by (simp add: field_simps)
  also have "2 * (of_int n + 3 / 2) = 2 * complex_of_int n + 3"
    by (simp add: field_simps)
  also have "Bessel_I (of_int (n + 1) + 1 / 2) z = C * SBessel_I (n +
1) z"
    by (subst "4.IH") auto
  also have "Bessel_I (of_int (n + 2) + 1 / 2) z = C * SBessel_I (n +
2) z"
    by (subst "4.IH") auto
  also have "(2 * of_int n + 3) / z * (C * SBessel_I (n + 1) z) + C *
SBessel_I (n + 2) z =
             C * ((2 * of_int n + 3) / z * SBessel_I (n + 1) z + SBessel_I
(n + 2) z)"
    using z by (simp add: field_simps)
  also have "(2 * of_int n + 3) / z * SBessel_I (n + 1) z + SBessel_I
(n + 2) z = SBessel_I n z"
    by (subst (2) SBessel_I.simps) (use "4.hyps" in ‹simp_all add: add_ac›)
  finally show ?case .
qed

lemma Bessel_I_conv_SBessel_I_real:
  assumes z: "z > (0 :: real)"
  shows   "Bessel_I (of_int n + 1 / 2) z = sqrt (2 * z / pi) * SBessel_I
n z"
proof -
  have "complex_of_real (Bessel_I (of_int n + 1 / 2) z) = Bessel_I (of_int
n + 1 / 2) (of_real z)"
    by (subst Bessel_I_complex_of_real [symmetric]) (use z in auto)
  also have "Bessel_I (of_int n + 1 / 2) (complex_of_real z) =
              of_real (sqrt (2 * z / pi) * SBessel_I n z)"
    by (subst Bessel_I_conv_SBessel_I_complex)
```

```
      (use z in ‹auto simp: SBessel_I_of_real real_sqrt_divide real_sqrt_mult›)
  finally show ?thesis
    by (simp only: of_real_eq_iff)
qed

lemma Bessel_I_pos_real:
  assumes "(x :: real) > 0" "a > -1"
  shows   "Bessel_I a x > 0"
proof -
  define f where "f = (λn. (x / 2) powr' (a + 2 * real n) / (fact n *
Gamma (1 + a + real n)))"
  have *: "f sums Bessel_I a x"
    unfolding f_def by (rule sums_Bessel_I_real) (use assms in auto)
  have "0 < f 0"
    unfolding f_def using assms
    by (auto intro!: divide_pos_pos simp: powr'_real)
  also have "... = (∑n∈{0}. f n)"
    by simp
  also have "... ≤ (∑n. f n)"
  proof (rule sum_le_suminf)
    show "f n ≥ 0" for n
      unfolding f_def using assms
      by (intro divide_nonneg_pos mult_pos_pos) (auto simp: powr'_real)
  qed (use * in ‹auto simp: sums_iff›)
  also have "... = Bessel_I a x"
    using * by (simp add: sums_iff)
  finally show ?thesis .
qed

lemma Bessel_I_strict_mono_real:
  assumes "0 ≤ x" "x < (y :: real)" "a > 0"
  shows   "Bessel_I a x < Bessel_I a y"
proof (cases "x = 0")
  case True
  thus ?thesis using assms
    by (auto simp: Bessel_I_0_right intro!: Bessel_I_pos_real)
next
  case False
  hence "x > 0"
    using assms by auto
  show ?thesis
  proof (rule DERIV_pos_imp_increasing[where f = "Bessel_I a"])
    fix t assume t: "x ≤ t" "t ≤ y"
    define D where "D = (Bessel_I (a - 1) t + Bessel_I (a + 1) t) / 2"
    have "(Bessel_I a has_real_derivative D) (at t)"
      using t ‹x > 0› by (auto intro!: derivative_eq_intros simp: D_def)
    moreover have "D > 0" unfolding D_def
      by (intro divide_pos_pos Bessel_I_pos_real add_pos_pos)
         (use assms t ‹x > 0› in auto)
```

```
    ultimately show "∃D. (Bessel_I a has_real_derivative D) (at t) ∧
D > 0"
      by blast
  qed fact
qed

lemma Bessel_I_mono_real:
  assumes "0 ≤ x" "x ≤ (y :: real)" "a > 0"
  shows   "Bessel_I a x ≤ Bessel_I a y"
  using Bessel_I_strict_mono_real[of x y a] assms by (cases "x = y") auto


lemma convex_on_realI_strong:
  assumes "connected A"
    and "⋀x. x ∈ A ⟹ (f has_real_derivative f' x) (at x within A)"
    and "⋀x y. x ∈ A ⟹ y ∈ A ⟹ x ≤ y ⟹ f' x ≤ f' y"
  shows "convex_on A f"
proof (rule convex_on_linorderI)
  show "convex A"
    using ‹connected A› convex_real_interval interval_cases
    by (smt (verit, ccfv_SIG) connectedD_interval convex_UNIV convex_empty)
      — the equivalence of "connected" and "convex" for real intervals is proved
later
next
  fix t x y :: real
  assume t: "t > 0" "t < 1"
  assume xy: "x ∈ A" "y ∈ A" "x < y"
  define z where "z = (1 - t) * x + t * y"
  with ‹connected A› and xy have ivl: "{x..y} ⊆ A"
    using connected_contains_Icc by blast

  from xy t have xz: "z > x"
    by (simp add: z_def algebra_simps)
  have "y - z = (1 - t) * (y - x)"
    by (simp add: z_def algebra_simps)
  also from xy t have "... > 0"
    by (intro mult_pos_pos) simp_all
  finally have yz: "z < y"
    by simp

  have cont: "continuous_on A f"
    using assms(2) by (intro DERIV_continuous_on)
  have deriv: "(f has_real_derivative f' ξ) (at ξ)" if ξ: "ξ ∈ {x<..<y}"
for ξ
  proof -
    have "(f has_real_derivative f' ξ) (at ξ within A)"
      by (rule assms) (use ξ ivl xz yz in auto)
    moreover have "{x<..<y} ⊆ A"
      using ivl by auto
```

```
    ultimately have "(f has_real_derivative f' ξ) (at ξ within {x<..<y})"
      using DERIV_subset by blast
    also have "at ξ within {x<..<y} = at ξ"
      using ξ by (intro at_within_open) auto
    finally show ?thesis .
  qed
  have diff: "f differentiable at ξ" if "ξ ∈ {x<..<y}" for ξ
    using deriv[OF that] real_differentiable_def by blast

  have "∃D ξ. ξ > x ∧ ξ < z ∧ (f has_real_derivative D) (at ξ) ∧ f z
- f x = (z - x) * D"
    by (rule MVT) (use xz yz ivl in <auto intro!: continuous_on_subset[OF
cont] diff>)
  then obtain ξ D1 where ξ: "ξ ∈ {x<..<z}" "(f has_field_derivative D1)
(at ξ)" "D1 = (f z - f x) / (z - x)"
    by auto
  have "(f has_field_derivative f' ξ) (at ξ)"
    by (rule deriv) (use ξ xy xz yz in auto)
  from DERIV_unique[OF ξ(2) this] have [simp]: "D1 = f' ξ" .

  have "∃D η. η > z ∧ η < y ∧ (f has_real_derivative D) (at η) ∧ f
y - f z = (y - z) * D"
    by (rule MVT) (use xz yz ivl in <auto intro!: continuous_on_subset[OF
cont] diff>)
  then obtain D2 η where η: "η ∈ {z<..<y}" "(f has_field_derivative
D2) (at η)" "D2 = (f y - f z) / (y - z)"
    by auto
  have "(f has_field_derivative f' η) (at η)"
    by (rule deriv) (use η xy xz yz in auto)
  from DERIV_unique[OF η(2) this] have [simp]: "D2 = f' η" .

  from η have "(f y - f z) / (y - z) = f' η"
    by simp
  also from ξ η ivl have "ξ ∈ A" "η ∈ A"
    by auto
  with ξ η have "f' η ≥ f' ξ"
    by (intro assms(3)) auto
  also from ξ(3) have "f' ξ = (f z - f x) / (z - x)"
    by simp
  finally have "(f y - f z) * (z - x) ≥ (f z - f x) * (y - z)"
    using xz yz by (simp add: field_simps)
  also have "z - x = t * (y - x)"
    by (simp add: z_def algebra_simps)
  also have "y - z = (1 - t) * (y - x)"
    by (simp add: z_def algebra_simps)
  finally have "(f y - f z) * t ≥ (f z - f x) * (1 - t)"
    using xy by simp
  then show "(1 - t) * f x + t * f y ≥ f ((1 - t) *R x + t *R y)"
    by (simp add: z_def algebra_simps)
```

qed

```
lemma Bessel_I_convex_real:
  assumes "a > 1"
  shows   "convex_on {0<..} (Bessel_I a)"
proof (rule convex_on_realI)
  define f where "f = (λx. (Bessel_I (a - 1) x + Bessel_I (a + 1) x)
/ 2)"
  show "(Bessel_I a has_real_derivative f x) (at x)" if "x ∈ {0<..}"
for x
    using that by (auto simp: f_def intro!: derivative_eq_intros)
  show "f x ≤ f y" if "x ≤ y" "x ∈ {0<..}" for x y
    using that unfolding f_def
    by (intro divide_right_mono add_mono Bessel_I_mono_real) (use assms
in auto)
qed auto
```

The Bessel function of the second kind Y_a and its modified version K_a are easy to derive from J_a and I_a whenever $a \notin \mathbb{Z}$, but the case where $a \in \mathbb{Z}$ is more tedious to do, so we leave that as future work.

end

2 A Hankel-style integral formula for the Bessel I function

```
theory Bessel_Hankel_Integral
imports
  "Bessel.Bessel"
  "Path_Automation.Path_Automation"
  "Prime_Number_Theorem.Prime_Number_Theorem_Library"
  "Incomplete_Gamma.More_Beta"
begin
```

2.1 Auxiliary integrals

Later on, we will need the integral

$$\int_{-\infty}^{\infty} (c^2 + t^2)^{-a} \,\mathrm{d}t = \frac{1}{2} c^{1-2a} B(\frac{1}{2}, a - \frac{1}{2})$$

for $c > 0$, $a > \frac{1}{2}$. We first show the $\int_0^\infty$ version. To this end, we perform the substitution $x = (t/c)^2$ to get

$$\frac{1}{2} c^{1-2e} \int_0^\infty x^{-\frac{1}{2}} (1 + x)^{-e} \,\mathrm{d}x \;,$$

which is one of the various equivalent integral definitions of the Beta function.

```
lemma Bessel_I_aux_integral1:
  fixes c e :: real
  assumes c: "c > 0" and e: "e > 1/2"
  shows "(λt. (c^2 + t^2) powr -e) absolutely_integrable_on {0<..}"
        "integral {0<..} (λt. (c^2 + t^2) powr -e) = c powr (1 - 2 * e)
/ 2 * Beta (1/2) (e - 1/2)"
proof -
  define C where "C = c powr (1-2*e) / 2"
  define f where "f = (λx. C * (x powr (-1/2) / (1 + x) powr e))"
  define g where "g = (λx. (x / c) ^ 2)"
  define g' where "g' = (λx. 2 * x / c ^ 2)"
  define h where "h = (λx. (c^2 + x^2) powr (-e))"
  define I where "I = c powr (1-2*e) / 2 * Beta (1/2) (e - 1/2)"

  have bij: "bij_betw g {0<..} {0<..}"
    by (rule bij_betwI[of _ _ _ "λy. sqrt y * c"]) (use c in <auto simp:
g_def>)

  have eq: "|g' x| *_R f (g x) = h x" if x: "x > 0" for x
  proof -
    have "|g' x| *_R f (g x) = c powr (-2 * e) * (c ^ 2) powr e / (c^2 +
x^2) powr e" using x c
      by (auto simp: g'_def f_def g_def field_simps powr_divide powr_minus

                     powr_half_sqrt power2_eq_square C_def powr_diff)
    also have "(c ^ 2) powr e = c powr (2*e)"
      by (subst powr_powr [symmetric]) (use c in simp_all)
    also have "c powr (-2 * e) * ... = 1"
      by (subst powr_add [symmetric]) (use c in auto)
    finally show ?thesis
      by (simp add: powr_minus field_simps h_def)
  qed

  have "(f has_integral I) {0<..}"
    using has_integral_mult_right[OF has_integral_Beta_0_infinity_real[of
"1/2" "e - 1/2"], of C] e
    by (simp add: f_def I_def C_def)
  hence "f absolutely_integrable_on {0<..} ∧ integral {0<..} f = I"
    using c by (simp add: f_def has_integral_iff absolutely_integrable_on_iff_nonneg
C_def)
  also have "{0<..} = g ` {0<..}"
    using bij by (simp add: bij_betw_def)
  also have "f absolutely_integrable_on g ` {0<..} ∧ integral (g ` {0<..})
f = I ⟷
             (λx. |g' x| * f (g x)) absolutely_integrable_on {0<..} ∧
             integral {0<..} (λx. |g' x| * f (g x)) = I"
    by (subst (2) eq_commute, intro has_absolute_integral_change_of_variables_1')
```

```
      (use c bij in ‹auto simp: g_def g'_def power2_eq_square bij_betw_def
                          intro!: derivative_eq_intros›)
  also have "(λx. |g' x| * f (g x)) absolutely_integrable_on {0<..} ⟷
             h absolutely_integrable_on {0<..}"
    by (rule set_integrable_cong) (use eq in auto)
  also have "integral {0<..} (λx. |g' x| * f (g x)) = integral {0<..} h"
    by (rule integral_cong) (use eq in auto)
  finally show "h absolutely_integrable_on {0<..}" "integral {0<..} h =
I"
    by (simp_all add: h_def)
qed
```

The $\int_{-\infty}^{\infty}$ version then easily follows by symmetry.

```
lemma Bessel_I_aux_integral2:
  fixes c e :: real
  assumes c: "c > 0" and e: "e > 1/2"
  shows "integrable lebesgue (λt. (c^2 + t^2) powr -e)"
  and   "lebesgue_integral lebesgue (λt. (c^2 + t^2) powr -e) = c powr
(1-2*e) * Beta (1/2) (e - 1/2)"
proof -
  define I where "I = c powr (1-2*e) / 2 * Beta (1/2) (e - 1/2)"
  have integrable: "(λt. (c^2 + t^2) powr -e) absolutely_integrable_on
{0<..}"
   and integral:   "integral {0<..} (λt. (c^2 + t^2) powr -e) = I"
   using Bessel_I_aux_integral1[of c e] c e by (simp_all add: I_def)

  show integrable': "integrable lebesgue (λt. (c^2 + t^2) powr -e)"
  proof -
    have "(λt. (c^2 + t^2) powr -e) absolutely_integrable_on {..<0}"
      using lebesgue_integrable_real_affine_iff[of "-1" "λt. indicator
{0<..} t * (c^2 + t^2) powr -e" 0]
            integrable unfolding set_integrable_def by (simp add: indicator_def)
    hence "(λt. (c^2 + t^2) powr -e) absolutely_integrable_on ({0<..} ∪
{..<0} ∪ {0})"
      by (intro set_integrable_Un integrable) (auto intro: absolutely_integrable_negligible
    also have "({0<..} ∪ {..<0} ∪ {0}) = (UNIV :: real set)"
      by auto
    finally show ?thesis
      by (simp add: set_integrable_def)
  qed

  show "lebesgue_integral lebesgue (λt. (c^2 + t^2) powr -e) = c powr (1-2*e)
* Beta (1/2) (e - 1/2)"
  proof -
    have integrable'': "set_integrable lebesgue UNIV (λt. (c^2 + t^2) powr
-e)"
      using integrable' by (simp add: set_integrable_def)
    have "I = integral {0<..} (λt. (c^2 + t^2) powr -e)"
      using integral by simp
```

```
    also have "... = (LINT t:{0<..}|lebesgue. (c^2 + t^2) powr -e)"
      using integrable by (rule set_lebesgue_integral_eq_integral(2) [symmetric])
    finally have *: "(LINT t:{0<..}|lebesgue. (c^2 + t^2) powr -e) = I" ..
    have **: "(LINT t:{..<0}|lebesgue. (c^2 + t^2) powr -e) = I"
      unfolding set_lebesgue_integral_def
      by (subst lebesgue_integral_real_affine[of "-1" _ 0])
         (use * in ‹simp_all add: set_lebesgue_integral_def indicator_def›)

    have "(LINT t|lebesgue. (c^2 + t^2) powr -e) = (LINT t:-{0}|lebesgue.
(c^2 + t^2) powr -e)"
      unfolding set_lebesgue_integral_def
    proof (rule integral_cong_AE)
      have "AE x in lebesgue. x ≠ 0"
        by (metis AE_completion_iff AE_lborel_singleton)
      thus "AE x in lebesgue. (c^2 + x^2) powr -e = indicator (-{0}) x *_R
(c^2 + x^2) powr -e"
        by eventually_elim (auto simp: indicator_def)
    qed (auto intro!: measurable_completion)
    also have "-{0} = {0<..} ∪ {..<0::real}"
      by auto
    also have "(LINT x:({0<..} ∪ {..<0::real})|lebesgue. (c^2 + x^2) powr
-e) =
                (LINT x:{0<..}|lebesgue. (c^2 + x^2) powr -e) + (LINT x:{..<0}|lebesgue.
(c^2 + x^2) powr -e)"
      by (rule set_integral_Un) (auto intro!: set_integrable_subset[OF
integrable''])
    also have "... = 2 * I"
      using * ** by simp
    finally show ?thesis
      by (simp add: I_def)
  qed
qed
```

2.2 A Hankel-style representation for $1/\Gamma(s)$

The biggest piece of work in this section will be to derive the following Hankel-style contour integral formula for Γ:

$$\frac{1}{\Gamma(s)} = \frac{1}{2\pi i}\int_{c-i\infty}^{c+i\infty} e^z z^{-s}\,\mathrm{d}z$$

For $\mathrm{Re}(s) > 1$, the integral clearly converges absolutely as a Lebesgue integral. For $\mathrm{Re}(s) \in (0, 1]$ it has to be interpreted as a Cauchy principal value, i.e. $\int_{c-i\infty}^{c+i\infty} = \lim_{T\to\infty} \int_{c-iT}^{c+IT}$.
We first show the existence of the integral as a Lebesgue integral if $\mathrm{Re}(s) > 1$. This is fairly easy to see by a comparison test, since it is $O(|t|^{-\mathrm{Re}(s)})$ uniformly for all t.

lemma *rGamma_hankel_integrable:*

```
  assumes c: "c > 0" and s: "Re s > 1"
  shows   "((λz. exp z * z powr (-s)) ∘ (λt. Complex c t)) absolutely_integrable_on
UNIV"
proof -
  have *: "((λz. exp z * z powr (-s)) ∘ (λt. Complex c (a*t))) absolutely_integrable_on
{1..}"
    if [simp]: "|a| = 1" for a :: real
  proof (rule set_integrable_bound)
    have "AE t in lebesgue. t ≠ 0"
      by (simp add: AE_completion_iff AE_lborel_singleton)
    thus "AE t∈{1..} in lebesgue. norm (((λz. exp z * z powr (-s)) ∘
(λt. Complex c (a*t))) t) ≤
                                      norm (exp c * exp (pi * |Im s|) * |t|
powr -Re s)"
    proof eventually_elim
      case (elim t)
      have "norm (exp (Complex c (a*t)) * Complex c (a*t) powr - s) =
              exp c * norm (Complex c (a*t)) powr -Re s * exp (Im s *
Arg (Complex c (a*t)))"
        using c by (simp add: norm_mult norm_exp norm_powr_complex)
      also have "... ≤ exp c * |t| powr -Re s * exp (|Im s| * pi)"
      proof (intro mult_mono)
        show "norm (Complex c (a*t)) powr -Re s ≤ |t| powr -Re s"
          by (rule powr_mono2')
             (use s elim abs_Im_le_cmod[of "Complex c (a*t)"] in <auto
simp: abs_mult>)
      next
        have "Im s * Arg (Complex c (a*t)) ≤ |Im s * Arg (Complex c (a*t))|"
          by linarith
        also have "... ≤ |Im s| * pi"
          unfolding abs_mult using Arg_bounded[of "Complex c (a*t)"] by
(auto intro!: mult_left_mono)
        finally show "exp (Im s * Arg (Complex c (a*t))) ≤ exp (|Im s|
* pi)"
          by simp
      qed auto
      finally show ?case
        by (auto simp: mult_ac)
    qed
  next
    have "(λx. x powr -Re s) absolutely_integrable_on {1<..}"
      using s set_integrable_powr_at_top[of 1 0 "-Re s"] by simp
    hence "(λx. x powr -Re s) integrable_on {1<..}"
      using set_lebesgue_integral_eq_integral(1) by blast
    also have "?this ⟷ (λx. x powr -Re s) integrable_on {1..}"
    proof (rule integrable_spike_set_eq)
      have "negligible {1::real}"
        by simp
      also have "... = sym_diff {1<..} {1..}"
```

```
      by auto
    finally show "negligible (sym_diff {1<..} {1..} :: real set)" .
  qed
  finally have "(λx. exp c * exp (pi * |Im s|) * x powr - Re s) integrable_on
{1..}"
    by (rule integrable_on_mult_right)
  also have "?this ⟷ (λx. exp c * exp (pi * |Im s|) * |x| powr - Re
s) integrable_on {1..}"
    by (rule integrable_cong) auto
  finally show "(λx. exp c * exp (pi * |Im s|) * |x| powr - Re s) absolutely_integrable_on
{1..}"
    by (rule nonnegative_absolutely_integrable_1) auto
qed (auto simp: set_borel_measurable_def Complex_eq intro!: measurable_completion)

have "integrable lebesgue (λt. indicator {1..} t *R (((λz. exp z *
z powr (-s)) ∘ (λt. Complex c (-t))) t))"
  using *[of "-1"] by (simp add: set_integrable_def)
from lebesgue_integrable_real_affine[OF this, of "-1" 0]
  have "((λz. exp z * z powr (-s)) ∘ (λt. Complex c t)) absolutely_integrable_on
{..-1}"
    unfolding set_integrable_def by (simp add: indicator_def)
moreover have "((λz. exp z * z powr (-s)) ∘ (λt. Complex c t)) absolutely_integrable_on
{1..}"
  using *[of 1] by simp
moreover have "((λz. exp z * z powr (-s)) ∘ (λt. Complex c t)) absolutely_integrable_on
{-1..1}"
  by (rule absolutely_integrable_continuous_real)
     (use c in ‹auto intro!: continuous_intros simp: complex_eq_iff›)
ultimately have "((λz. exp z * z powr (-s)) ∘ (λt. Complex c t)) absolutely_integrable_on

      ({-1..1} ∪ {1..} ∪ {..-1})"
  by (intro set_integrable_Un) auto
also have "{-1..1} ∪ {1..} ∪ {..-1::real} = UNIV"
  by auto
finally show ?thesis .
qed
```

Next, we show the actual integral formula in the case that $\mathrm{Re}(s) > 0$. The proof works is modelled after Lemma 2.3 by Kong and Teo [2] and goes roughly like this:

- We consider the Hankel-style integration contour sketched in Figure 1.
- Since two of the horizontal lines lie directly on the branch cut, this integral is very awkward to handle. We instead only consider the part of the contour in the upper half plane, adding a horizontal line on the positive real axis to connect the part we cut off, and then also move the branch cut out of the way to the negative real axis by using the alternative branch of the logarithm $\widehat{\ln} z = \ln(-iz) + \frac{1}{2}i\pi$.

- We apply the Cauchy integral theorem to this contour and note that the integrand has no singularities inside, so its value must be 0.
- We do the same for the portion of the contour that is in the lower half of the complex plane and add the two values, noting that the horizontal lines we added cancel out. To save some work, we note that we can get this result for the lower half plane from that for the upper half plane (which we already have) by complex conjugation due to the fact that the integrand is symmetric under conjugation.
- Denote the vertical integral at $\mathrm{Re}(z) = c$ as $\mathrm{lhs}(T, s)$ and the sum of the other ones as $\mathrm{rhs}(r, T, s)$. The above shows that $\mathrm{lhs}(T, s) = \mathrm{rhs}(r, T, s)$ holds for all s with $\mathrm{Re}(s) > 0$.
- This immediately means that the value of $\mathrm{rhs}(r, T, s)$ is actually independent of r.
- Note also that $\lim_{T\to\infty} \mathrm{rhs}(r, T, s)$ exists for any s with $\mathrm{Re}(s) > 0$, and of course this limit is also independent of r. Write $\mathrm{RHS}(r, s)$ for this limit. In particular, this already shows that $\mathrm{lhs}(T, s)$ also has a limit for $T \to \infty$ and its value is $\mathrm{RHS}(r, s)$.
- In fact, $\mathrm{RHS}(r, s)$ can be expressed in terms of the incomplete Gamma function $\Gamma(1 - s, r)$ and some error terms corresponding to the small circle around the origin.
- Thus, if $\mathrm{Re}(s) < 1$, we can let $r \to 0$ and find that the error terms in $\mathrm{RHS}(r, s)$ vanish and $\lim_{r\to 0} \Gamma(1 - s, r) = \Gamma(1 - s)$. Together with the reflection formula for Γ, we obtain $\mathrm{RHS}(s) = 2\pi/\Gamma(s)$, exactly as we wanted. This establishes the result we wanted in the strip $0 < \mathrm{Re}(s) < 1$.
- Moreover, both $\Gamma(1 - s, r)$ and the error terms and other factors occurring in $\mathrm{RHS}(r, s)$ are holomorphic functions in s for $\mathrm{Re}(s) > 0$, so we can use analytic continuation to lift the above equality to all s with $\mathrm{Re}(s) > 0$ and we are done.

```
theorem rGamma_hankel_strong:
  assumes c: "c > 0" and s: "Re s > 0"
  shows   "((λT. integral {-T..T} ((λz. exp z * z powr (-s)) ∘ (λt. Complex
c t)))
              ⟶ (2 * of_real pi * rGamma s)) at_top"
proof -
  define A where "A = {z. Re z = 0 ∧ Im z ≤ 0}"
  define Ln' where "Ln' ≡ (λz. ln (-i * z) + i * pi / 2)"
  have [holomorphic_intros]: "Ln' holomorphic_on X" if "X ∩ A = {}" for
X
```

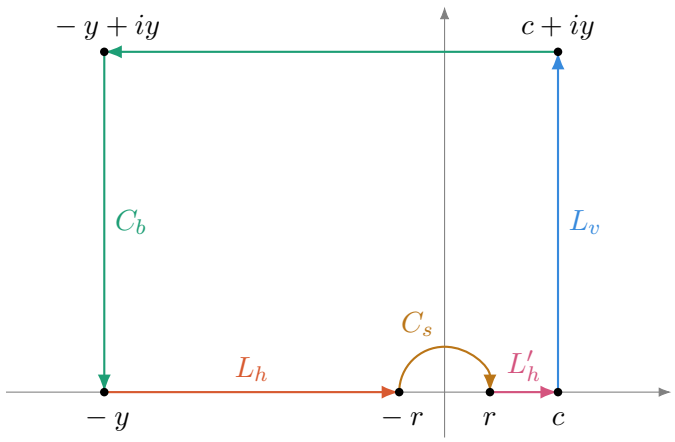

Figure 1: The Hankel-style integration contour used in the proof of the $\int_{c-i\infty}^{c+i\infty}$-style integral representation for $1/\Gamma(s)$.

```
    unfolding Ln'_def using that
    by (auto intro!: holomorphic_intros simp: A_def complex_nonpos_Reals_iff)
  have [analytic_intros]: "Ln' analytic_on X" if "X ∩ A = {}" for X
    unfolding Ln'_def using that
    by (auto intro!: analytic_intros simp: A_def complex_nonpos_Reals_iff)

  have Ln'_of_real_pos: "Ln' (of_real x) = of_real (ln x)" if "x > 0"
for x
  proof -
    have "Ln' (of_real x) = Ln (of_real x * (-i)) + i * pi / 2"
      by (simp add: Ln'_def mult_ac)
    also have "... = of_real (ln x)" using that
      by (subst Ln_times_of_real) (auto simp: Ln_of_real)
    finally show ?thesis .
  qed

  have Ln'_of_real_neg: "Ln' (of_real x) = of_real (ln (-x)) + i * pi"
if "x < 0" for x
  proof -
    have "Ln' (of_real x) = Ln (of_real (-x) * i) + i * pi / 2"
      by (simp add: Ln'_def mult_ac)
    also have "... = of_real (ln (-x)) + i * pi" using that
      by (subst Ln_times_of_real) (auto simp: Ln_Reals_eq)
    finally show ?thesis .
  qed

  have Ln'_eq_Ln: "Ln' z = Ln z" if z: "z ≠ 0" "Im z ≥ 0" for z
  proof -
    have "0 ≤ Im (Ln z)" "Im (Ln z) ≤ pi"
```

```
    using z Im_Ln_pos_le[of z] by auto
  moreover from this have "Im (Ln z) > -pi/2"
    using pi_gt_zero by linarith
  ultimately show ?thesis
  unfolding Ln'_def using z
  by (subst Ln_times_simple) auto
qed

define f where "f = (λs z. exp z * z powr (-s) :: complex)"
define fr where "fr = (λs z. exp z * complex_of_real z powr (-s))"
define g where "g = (λs z. exp (z - s * Ln' z))"
define h :: "complex ⇒ real ⇒ complex"
  where "h = (λs t. exp (of_real t - s * of_real (ln (-t))))"

define Lv :: "real ⇒ real ⇒ complex" where "Lv = (λy. linepath (of_real
c) (Complex c y))"
define Lh :: "real ⇒ real ⇒ real ⇒ complex"
  where "Lh = (λr y. linepath (-of_real y) (-of_real r))"
define Lh' :: "real ⇒ real ⇒ complex"
  where "Lh' = (λr. linepath (of_real r) (of_real c))"
define Cb :: "real ⇒ real ⇒ complex"
  where "Cb = (λy. linepath (Complex c y) (Complex (-y) y) +++ linepath
(Complex (-y) y) (-of_real y))"
define Cs :: "real ⇒ real ⇒ complex"
  where "Cs = (λr. part_circlepath 0 r pi 0)"
define err1 where "err1 = (λs y. contour_integral (Cb y) (g s))"
define err2 where "err2 = (λs r. contour_integral (Cs r) (g s))"

have "pathstart (Lv y) = of_real c" "pathfinish (Lv y) = Complex c y"
for y
  by (auto simp: Lv_def)
have "pathstart (Cb y) = Complex c y" "pathfinish (Cb y) = of_real (-y)"
for y
  by (auto simp: Cb_def complex_eq_iff Im_exp Re_exp)

have path_image_Cb: "z ∉ A" if z: "z ∈ path_image (Cb T)" and T: "T
> c" for z T
  using that T c by (auto simp: A_def Cb_def path_image_join closed_segment_same_Re
closed_segment_same_Im)

have path_image_Cs: "z ∉ A" if z: "z ∈ path_image (Cs r)" and r: "r
> 0" for z r
proof
  assume "z ∈ A"
  have "z ∈ path_image (part_circlepath 0 r 0 pi)"
    using z unfolding Cs_def by (metis reversepath_part_circlepath
path_image_reversepath)
  then obtain t where t: "z = rcis r t" "t ∈ {0..pi}"
    by (auto simp: path_image_part_circlepath rcis_def cis_conv_exp)
```

```
    with ‹z ∈ A› and r have "cos t = 0"
      by (auto simp: A_def t(1))
    with t(2) have "t = pi / 2"
      by (metis arccos_0 arccos_cos atLeastAtMost_iff)
    with t(2) and ‹z ∈ A› and r show False
      by (auto simp: t(1) A_def)
  qed
```

The horizontal line integral on the left half of the real axis can be extended to infinity for any s due to the exponential decay of the integrand.

```
  have integrable: "set_integrable lborel {r..} (λt. h s (-t))" if r:
"r > 0" for r s
    using absolutely_integrable_incomplete_Gamma[of r s] r
    by (simp add: h_def set_integrable_def integrable_completion)
```

We first consider a "half-Hankel" contour consisting only of the half of the Hankel contour lying in the upper half plane.

```
  define γ where "γ = (λr y. Lv y +++ Cb y +++ Lh r y +++ Cs r +++ Lh'
r)"
  have γ: "valid_path (γ r y)" "pathfinish (γ r y) = pathstart (γ r y)"
    if "y ≥ c" "r ≥ 0" for r y using that c
      by (auto simp: γ_def Lv_def Lh_def Cb_def Cs_def Lh'_def A_def complex_eq_iff
Re_exp Im_exp
                    path_image_join closed_segment_same_Re closed_segment_same_Im
                    closed_segment_eq_real_ivl intro!: valid_path_join)
  have path_image_γ: "path_image (γ r T) ⊆ -A"
    if "T > c" "r > 0" "r < c" for r T
    using path_image_Cs[of _ r] path_image_Cb[of _ T] that
    by (auto simp: γ_def Cs_def Cb_def Lv_def Lh_def Lh'_def path_image_join

          closed_segment_same_Re closed_segment_same_Im closed_segment_eq_real_ivl
A_def)
```

Applying Cauchy's theorem to this contour, we obtain a term relating the horizontal integral on the real axis with the vertical one, modulo some error terms.

```
  have 1: "i * integral {0..y} (λx. f s (Complex c x)) + integral {-y..-r}
(fr s) =
             -(integral {r..c} (fr s) + err1 s y + err2 s r)"
     if yr: "y > c" "r > 0" "r < c" for s :: complex and r y :: real
  proof -
    have "(g s has_contour_integral 0) (γ r y)"
    proof (rule Cauchy_theorem_simply_connected)
      show "open (-A)"
      proof -
        have "closed ({z. Re z = 0} ∩ {z. Im z ≤ 0})"
          by (intro closed_Int closed_halfspace_Re_eq closed_halfspace_Im_le)
        also have "... = A"
```

```
        by (auto simp: A_def)
      finally show ?thesis
        by auto
    qed
  next
    show "simply_connected (-A)"
    proof (rule simply_connected_retraction_gen)
      show "simply_connected (-complex_of_real ` {..0})"
        by (rule simply_connected_slotted_complex_plane_left)
    next
      show "continuous_on (-A) (λz. -i * z)" "continuous_on (-complex_of_real
` {..0}) (λz. i * z)"
        by (auto intro!: continuous_intros)
    next
      have "bij_betw (λz. i * z) (- complex_of_real ` {..0}) (-A)"
        by (rule bij_betwI[of _ _ _ "λz. -i * z"]) (auto simp: A_def
complex_eq_iff)
      thus "(*) i ` (- complex_of_real ` {..0}) = - A"
        by (simp add: bij_betw_def)
    qed (auto simp: complex_eq_iff A_def)
  next
    show "g s holomorphic_on -A"
      by (auto simp: g_def intro!: holomorphic_intros)
  qed (use γ[of y r] path_image_γ[of y r] yr in simp_all)
  hence "contour_integral (γ r y) (g s) = 0"
    using contour_integral_unique by blast

  also have "contour_integral (γ r y) (g s) =
             contour_integral (Lv y) (g s) + contour_integral (Cb y)
(g s) +
             contour_integral (Lh r y) (g s) + contour_integral (Cs
r) (g s) +
             contour_integral (Lh' r) (g s)"
    unfolding γ_def
  proof path
    show "g s analytic_on path_image (Lv y +++ Cb y +++ Lh r y +++
Cs r +++ Lh' r)"
    proof (rule analytic_on_subset)
      show "path_image (Lv y +++ Cb y +++ Lh r y +++ Cs r +++ Lh' r)
⊆ -A"
        using path_image_Cb[of _ y] path_image_Cs[of _ r] yr c
        unfolding Lv_def Cb_def Lh_def Cs_def Lh'_def
        by (fastforce simp: path_image_join A_def closed_segment_same_Re
closed_segment_same_Im
                            closed_segment_eq_real_ivl)
    qed (auto intro!: analytic_intros simp: g_def)
  qed (auto simp: Cs_def Lh'_def Lh_def Cb_def Lv_def)

  also have "contour_integral (Lh r y) (g s) = integral {-y..-r} (λx.
```

```
g s (of_real x))"
      unfolding Lh_def by (subst contour_integral_linepath_Reals_eq) (use
yr c in simp_all)
    also have "... = integral {-y..-r} (fr s)"
      unfolding g_def fr_def using c yr
      by (intro integral_cong)
         (auto simp: Ln'_of_real_neg powr_def Ln_of_real' field_simps
exp_diff exp_minus exp_add)
    also have "contour_integral (Lh' r) (g s) = integral {r..c} (λx. g
s (of_real x))"
      unfolding Lh'_def by (subst contour_integral_linepath_Reals_eq)
(use yr c in simp_all)
    also have "... = integral {r..c} (fr s)"
      unfolding g_def fr_def using c yr
      by (intro integral_cong)
         (auto simp: Ln'_of_real_pos powr_def Ln_of_real' field_simps
exp_diff exp_minus exp_add)
    also have "contour_integral (Lv y) (g s) = i * integral {0..y} (λx.
g s (Complex c x))"
      unfolding Lv_def by (rule contour_integral_linepath_same_Re) (use
yr in auto)
    also have "integral {0..y} (λx. g s (Complex c x)) = integral {0..y}
(λx. f s (Complex c x))"
      unfolding g_def f_def
      by (intro integral_cong, subst Ln'_eq_Ln)
         (use c in ‹auto simp: g_def powr_def field_simps exp_diff exp_minus
exp_add complex_eq_iff›)
    finally show ?thesis
      unfolding err1_def err2_def by (Groebner_Basis.algebra)
  qed
```

We combine this result with a flipped copy of itself to get the full Hankel contour. One of the error terms, namely the horizontal line along the positive real axis, cancels.

```
  have 2: "integral {-y..y} (λt. f s (Complex c t)) =
             2 * sin (pi * s) * (CLBINT t=r..y. h s (-t)) -
             i * (cnj (err1 (cnj s) y) + cnj (err2 (cnj s) r) - err2 s
r - err1 s y)"
    if yr: "y > c + r" "r > 0" "r < c" for s :: complex and y r :: real
  proof -
    have "i * integral {0..y} (λx. f s (Complex c x)) + integral {-y..-r}
(fr s) -
          cnj (i * integral {0..y} (λx. f (cnj s) (Complex c x)) + integral
{-y..-r} (fr (cnj s))) =
          -(integral {r..c} (fr s) + err1 s y + err2 s r) -
          cnj (-(integral {r..c} (fr (cnj s)) + err1 (cnj s) y + err2
(cnj s) r))"
      by (subst (1 2) 1) (use yr in auto)
    also have "cnj (-(integral {r..c} (fr (cnj s)) + err1 (cnj s) y +
```

```
err2 (cnj s) r)) =
              -integral {r..c} (λx. cnj (fr (cnj s) x)) - cnj (err1 (cnj
s) y) - cnj (err2 (cnj s) r)"
      by (simp add: integral_cnj)
    also have "integral {r..c} (λx. cnj (fr (cnj s) x)) = integral {r..c}
(fr s)"
      by (intro integral_cong) (use c yr in ‹auto simp: exp_cnj fr_def
cnj_powr›)
    also have "-(integral {r..c} (fr s) + err1 s y + err2 s r) -
              (-... - cnj (err1 (cnj s) y) - cnj (err2 (cnj s) r)) =
              cnj (err1 (cnj s) y) + cnj (err2 (cnj s) r) - err2 s r
- err1 s y"
      by simp

    also have "cnj (i * integral {0..y} (λx. f (cnj s) (Complex c x))
+ integral {-y..-r} (fr (cnj s))) =
              integral {-y..-r} (λx. cnj (fr (cnj s) x)) -
              i * integral {0..y} (λx. cnj (f (cnj s) (Complex c x)))"
      by (simp add: integral_cnj)
    also have "i * integral {0..y} (λx. f s (Complex c x)) + integral
{-y..-r} (fr s) - ... =
              i * (integral {0..y} (λx. cnj (f (cnj s) (Complex c x)))
+ integral {0..y} (λx. f s (Complex c x))) +
              (integral {-y..-r} (fr s) - integral {-y..-r} (λx. cnj
(fr (cnj s) x)))"
      by (Groebner_Basis.algebra)
    also have "integral {-y..-r} (fr s) =
              integral {-y..-r} (λt. exp (t - s * complex_of_real (ln
(-t)) - i * s * pi))"
      by (intro integral_cong)
         (use yr in ‹auto simp: fr_def powr_def exp_cnj Ln_of_real' ring_distribs
exp_diff
                                exp_minus exp_add field_simps simp flip:
of_real_minus›)
    also have "integral {-y..-r} (λx. cnj (fr (cnj s) x)) =
              integral {-y..-r} (λt. exp (t - s * complex_of_real (ln
(-t)) + i * s * pi))"
      by (intro integral_cong)
         (use yr in ‹auto simp: fr_def powr_def exp_cnj Ln_of_real' ring_distribs
exp_diff
                                exp_minus exp_add field_simps simp flip:
of_real_minus›)
    also have "integral {-y..-r} (λt. exp (t - s * complex_of_real (ln
(-t)) - i * s * pi)) - ... =
              integral {-y..-r} (λt. exp (t - s * complex_of_real (ln
(-t)) - i * s * pi) -
                                     exp (t - s * complex_of_real (ln
(-t)) + i * s * pi))"
      by (rule integral_diff [symmetric]; rule integrable_continuous_interval)
```

```
        (use yr c in ‹auto intro!: continuous_intros simp: fr_def›)
    also have "(λt. exp (t - s * complex_of_real (ln (-t)) - i * s * pi)
-
                   exp (t - s * complex_of_real (ln (-t)) + i * s * pi))
=
               (λt. exp (t - s * complex_of_real (ln (-t))) * (exp (-i
* s * pi) - exp (i * s * pi)))"
      by (simp add: exp_diff exp_add exp_minus field_simps)
    also have "... = (λt. -2 * i * sin (pi * s) * h s t)"
      by (simp add: h_def sin_exp_eq algebra_simps add_divide_distrib
diff_divide_distrib)
    also have "integral {-y..-r} ... = -2 * i * sin (pi * s) * integral
{-y..-r} (h s)"
      unfolding h_def by (rule integral_mult_right)
    also have "integral {-y..-r} (h s) = integral {r..y} (λt. h s (-t))"
      by (subst Henstock_Kurzweil_Integration.integral_reflect_real [symmetric])
auto
    also have "... = (CLBINT t=r..y. h s (-t))"
      by (intro interval_integral_eq_integral [symmetric] borel_integrable_atLeastAtMost')
         (use yr in ‹auto simp: h_def complex_eq_iff intro!: continuous_intros›)

    also have "integral {0..y} (λx. cnj (f (cnj s) (Complex c x))) =
               integral {-y..0} (λx. cnj (f (cnj s) (Complex c (-x))))"
      by (subst Henstock_Kurzweil_Integration.integral_reflect_real [symmetric])
simp_all
    also have "... = integral {-y..0} (λx. f s (Complex c x))"
      using c yr by (intro integral_cong) (auto simp: f_def exp_cnj cnj_powr
complex_cnj)
    also have "integral {-y..0} (λx. f s (Complex c x)) + integral {0..y}
(λx. f s (Complex c x)) =
               integral {-y..y} (λx. f s (Complex c x))"
      by (intro Henstock_Kurzweil_Integration.integral_combine integrable_continuous_real)
         (use yr in ‹auto intro!: continuous_intros simp: f_def complex_eq_iff›)

    finally show ?thesis using power2_i
      by (Groebner_Basis.algebra)
  qed
```

The four error terms vanish under diffrent preconditions: some only for $\mathrm{Re}(s) > 0$ and some for $\mathrm{Re}(s) < 1$.

```
  have lim_err1: "(err1 s ⟶ 0) at_top" if s: "Re s > 0" for s
  proof (rule Lim_null_comparison)
    define C1 where "C1 = exp (c + pi * |Im s|)"
    define C2 where "C2 = exp (pi * |Im s|)"
    show "eventually (λT. norm (err1 s T) ≤ C1 * T powr (-Re s) + C2
* exp (-T) * T powr (-Re s) * T) at_top"
      using eventually_gt_at_top[of c]
    proof eventually_elim
      case (elim T)
```

```
      define err1_1 where "err1_1 = contour_integral (linepath (Complex
c T) (Complex (-T) T)) (g s)"
      define err1_2 where "err1_2 = contour_integral (linepath (Complex
(- T) T) (-of_real T)) (g s)"

      have "err1 s T = err1_1 + err1_2"
        unfolding err1_def err1_1_def err1_2_def Cb_def g_def using path_image_Cb[of
_ T] elim
        by (subst contour_integral_join)
           (auto intro!: analytic_imp_contour_integrable analytic_intros
                 simp: A_def Cb_def path_image_join)
      also have "norm ... ≤ norm err1_1 + norm err1_2"
        by norm
      also have "err1_1 = -contour_integral (linepath (Complex (-T) T)
(Complex c T)) (g s)"
        unfolding err1_1_def using contour_integral_reversepath by fastforce
      also have "norm ... = norm (integral {-T..c} (λx. g s (of_real x
+ of_real T * i)))"
        using elim c by (subst contour_integral_linepath_same_Im) auto
      also have "... ≤ T powr (-Re s) * exp (|Im s| * pi) * (exp c - exp
(-T))"
      proof (rule integral_norm_bound_integral')
        fix t assume t: "t ∈ {-T..c}"
        define z where "z = (of_real t + of_real T * i)"
        have "z ≠ 0"
          using c elim by (auto simp: complex_eq_iff z_def)
        hence "g s z = f s z" using t elim
          unfolding f_def g_def powr_def
          by (subst Ln'_eq_Ln) (auto simp: z_def exp_diff exp_minus field_simps)
        also have "norm ... = exp t * (norm z) powr (-Re s) * exp (Im
s * Arg z)"
          by (simp add: z_def f_def norm_mult norm_powr_complex)
        also have "... ≤ exp t * T powr (-Re s) * exp (|Im s| * pi)"
        proof (intro mult_mono powr_mono2')
          have "T = Im z"
            by (auto simp: z_def)
          also have "... ≤ norm z"
            using abs_Im_le_cmod[of z] by simp
          finally show "T ≤ norm z" .
        next
          have "Im s * Arg z ≤ |Im s * Arg z|"
            by linarith
          also have "... ≤ |Im s| * pi"
            unfolding abs_mult using Arg_bounded[of z] by (intro mult_left_mono)
auto
          finally show "exp (Im s * Arg z) ≤ exp (|Im s| * pi)"
            by simp
        qed (use s elim c in auto)
        also have "... = T powr (-Re s) * exp (|Im s| * pi) * exp t"
```

```
        by (simp add: C1_def algebra_simps)
      finally show "norm (g s z) ≤ T powr (-Re s) * exp (|Im s| * pi)
* exp t" .
    next
      have "((λx. T powr (-Re s) * exp (|Im s| * pi) * exp x) has_integral

              (T powr (-Re s) * exp (|Im s| * pi) * exp c -
               T powr (-Re s) * exp (|Im s| * pi) * exp (-T))) {-T..c}"
using elim c
        by (intro fundamental_theorem_of_calculus)
           (auto simp flip: has_real_derivative_iff_has_vector_derivative
                 intro!: derivative_eq_intros)
      thus "((λx. T powr (-Re s) * exp (|Im s| * pi) * exp x) has_integral

              (T powr (-Re s) * exp (|Im s| * pi) * (exp c - exp (-T))))
{-T..c}"
        by (simp add: algebra_simps)
    qed (auto simp: g_def Ln'_def intro!: measurable_completion borel_measurable_if_D)
    also have "... ≤ C1 * T powr (-Re s)"
      by (simp add: C1_def algebra_simps exp_add)

    also have "norm err1_2 ≤ C2 * exp (-T) * T powr (-Re s) * norm
(-of_real T - Complex (-T) T)"
      unfolding err1_2_def
    proof (rule contour_integral_bound_linepath)
      show "g s contour_integrable_on linepath (Complex (- T) T) (-of_real
T)" using c elim
        by (auto intro!: analytic_imp_contour_integrable analytic_intros
                 simp: g_def A_def Cb_def path_image_join closed_segment_same_Re)
    next
      fix z assume "z ∈ closed_segment (Complex (-T) T) (-complex_of_real
T)"
      then obtain t where t: "t ∈ {0..T}" "z = Complex (-T) t"
        using c elim by (auto simp: closed_segment_same_Re closed_segment_eq_real_ivl
complex_eq_iff)
      have "z ≠ 0"
        using c elim by (auto simp: complex_eq_iff t(2))
      hence "g s z = f s z" using t elim
        unfolding f_def g_def powr_def
        by (subst Ln'_eq_Ln) (auto simp: t(2) exp_diff exp_minus field_simps)
      also have "norm (f s z) = exp (-T) * (norm z) powr (-Re s) * exp
(Im s * Arg z)"
        by (simp add: f_def norm_mult t(2) norm_powr_complex)
      also have "... ≤ exp (-T) * T powr (-Re s) * exp (|Im s| *pi)"
      proof (intro mult_mono powr_mono2')
        from c elim have "T = |Re z|"
          by (auto simp: t)
        also have "... ≤ norm z"
          using abs_Re_le_cmod[of z] by simp
```

```
        finally show "T ≤ norm z" .
      next
        have "Im s * Arg z ≤ |Im s * Arg z|"
          by linarith
        also have "... ≤ |Im s| * pi"
          unfolding abs_mult using Arg_bounded[of z] by (intro mult_left_mono)
auto
        finally show "exp (Im s * Arg z) ≤ exp (|Im s| * pi)"
          by simp
      qed (use c elim s in auto)
      finally show "norm (g s z) ≤ C2 * exp (-T) * T powr (-Re s)"
        by (simp add: C2_def algebra_simps)
    qed (auto simp: C2_def)
    also have "norm (-of_real T - Complex (-T) T) = T"
      using elim c by (simp add: cmod_def)
    finally show ?case
      by simp_all
  qed

  show "((λT. C1 * T powr (-Re s) + C2 * exp (-T) * T powr (-Re s)
* T) ⟶ 0) at_top"
    using s by real_asymp
qed

have lim_err2: "(err2 s ⟶ 0) (at_right 0)" if s: "Re s < 1" for
s
proof (rule Lim_null_comparison)
  define C where "C = pi * exp (c + |Im s| * pi)"
  have "eventually (λr::real. r > 0) (at_right 0)"
       "eventually (λr::real. r < c) (at_right 0)"
    using c by real_asymp+
  thus "eventually (λr. norm (err2 s r) ≤ C * r powr (1 - Re s)) (at_right
0)"
  proof eventually_elim
    case (elim r)
    have "norm (err2 s r) ≤ exp c * r powr - Re s * exp (|Im s| * pi)
* r * |0 - pi|"
      unfolding err2_def Cs_def
    proof (rule contour_integral_bound_part_circlepath)
      show "g s contour_integrable_on part_circlepath 0 r pi 0"
        using path_image_Cs[of _ r] elim unfolding g_def Cs_def
        by (auto intro!: analytic_imp_contour_integrable analytic_intros
simp: A_def)
    next
      show "norm (g s z) ≤ exp c * r powr - Re s * exp (|Im s| * pi)"

        if z: "z ∈ path_image (part_circlepath 0 r pi 0)" for z
      proof -
        have "z ∈ path_image (part_circlepath 0 r 0 pi)"
```

```
        using z by (metis reversepath_part_circlepath path_image_reversepath)
      then obtain t where t: "z = rcis r t" "t ∈ {0..pi}"
        by (auto simp: path_image_part_circlepath rcis_def cis_conv_exp)
      have "g s z = f s z" unfolding g_def f_def using t(2) elim
        by (subst Ln'_eq_Ln)
           (auto simp: t(1) exp_diff powr_def exp_minus field_simps
sin_ge_zero)
      also have "norm (f s z) = exp (r * cos t) * r powr -Re s * exp
(Im s * Arg z)"
        using elim t by (simp add: t f_def norm_mult norm_powr_complex
Arg_rcis)
      also have "... ≤ exp c * r powr -Re s * exp (|Im s| * pi)"
      proof (intro mult_mono)
        have "r * cos t ≤ r * 1"
          by (rule mult_left_mono) (use elim in auto)
        also have "... ≤ c"
          using elim by simp
        finally show "exp (r * cos t) ≤ exp c"
          by simp
      next
        have "Im s * Arg z ≤ |Im s * Arg z|"
          by linarith
        also have "... ≤ |Im s| * pi"
          unfolding abs_mult using Arg_bounded[of z] by (intro mult_left_mono)
auto
        finally show "exp (Im s * Arg z) ≤ exp (|Im s| * pi)"
          by simp
      qed (use elim in auto)
      finally show "norm (g s z) ≤ exp c * r powr (-Re s) * exp (|Im
s| * pi)" .
    qed
  qed (use elim in auto)
  also have "exp c * r powr (-Re s) * exp (|Im s| * pi) * r * |0 -
pi| = C * r powr (1-Re s)"
    using elim by (auto simp: C_def powr_diff powr_minus field_simps
exp_add)
  finally show "norm (err2 s r) ≤ C * r powr (1 - Re s)" .
qed

show "((λr. C * r powr (1 - Re s)) ⟶ 0) (at_right 0)"
  using s by real_asymp
qed
```

We now arbitrarily pick some suitable radius (we will soon see that all the values are independent of it anyway).

```
define r where "r = c / 2"
have r: "r > 0" "r < c"
  using c by (auto simp: r_def)
```

```
  define lhs where "lhs = (λs T. integral {-T..T} (λt. f s (Complex c
t)))"
  define rhs where "rhs = (λs r T. 2 * sin (complex_of_real pi * s) *
(CLBINT t=r..T. h s (-t)) -
                                  i * (cnj (err1 (cnj s) T) + cnj (err2
(cnj s) r) -
                                  err2 s r - err1 s T))"
  define RHS where "RHS = (λs r. 2 * sin (complex_of_real pi * s) * (CLBINT
t:{r..}. h s (- t)) +
                                      i * (err2 s r - cnj (err2 (cnj s)
r)))"
```

For $\mathrm{Re}(s) > 0$, we now let $T \to \infty$ and let two of the error terms vanish. What we obtain from this is that the vertical integral we are interested in (`lhs`) converges to a slightly deformed version of the horizontal integrals we want (`RHS`). The deformation is the small circle around the origin to avoid the singularity there.

```
  have lim_lhs: "(lhs s ⟶ RHS s r) at_top"
    if r: "r > 0" "r < c" and s: "Re s > 0" for s :: complex and r ::
real
  proof -
    have "(rhs s r ⟶ 2 * sin (pi * s) *
            (CLBINT t:{r..}. h s (-t)) - i * (cnj 0 + cnj (err2 (cnj s)
r) - err2 s r - 0)) at_top"
      unfolding rhs_def
    proof (intro tendsto_intros lim_err1 lim_err2)
      have "((λx. set_lebesgue_integral lborel {r..x} (λt. h s (-t)))
⟶
              set_lebesgue_integral lborel {r..} (λt. h s (-t))) at_top"
      proof (rule tendsto_set_lebesgue_integral_at_top)
        show "set_integrable lborel {r..} (λt. h s (- t))"
          by (rule integrable) (use r in auto)
      qed auto
      moreover have "eventually (λx. set_lebesgue_integral lborel {r..x}
(λt. h s (-t)) =
                                    (CLBINT t=ereal r..ereal x. h s (-t)))
at_top"
        using eventually_ge_at_top[of r] by eventually_elim (simp_all
add: interval_integral_Icc)
      ultimately have *: "((λx. (CLBINT t=ereal r..ereal x. h s (-t)))
⟶
                          set_lebesgue_integral lborel {r..} (λt. h s
(-t))) at_top"
        by (rule Lim_transform_eventually)
      show "((λx. CLBINT t=ereal r..ereal x. h s (- t)) ⟶
                  CLBINT t:{r..}. h s (- t)) at_top"
        by (rule filterlim_compose[OF *]) real_asymp
    qed (use s in auto)
```

```
    hence "(rhs s r ⟶ 2 * sin (pi * s) * (CLBINT t:{r..}. h s (-t))
+
                                i * (err2 s r - cnj (err2 (cnj s) r))) at_top"
      by (simp add: algebra_simps)
    moreover have "eventually (λT. lhs s T = rhs s r T) at_top"
    proof -
      have "eventually (λT. c + r < T) at_top"
        by real_asymp
      thus ?thesis
      proof eventually_elim
        case (elim T)
        thus ?case using 2[of r T s] r
          by (simp add: lhs_def rhs_def)
      qed
    qed
    ultimately show "(lhs s ⟶ RHS s r) at_top"
      using filterlim_cong unfolding RHS_def by blast
  qed
```

Next, we show that `RHS s r` is equal to $2\pi/\Gamma(s)$ for all s with $\mathrm{Re}(s) < 1$, independently of the radius r.

```
  have RHS_eq: "RHS s r = 2 * pi * rGamma s" if s: "Re s ∈ {0<..<1}" for
s :: complex
  proof -
    have "((λr. LBINT t. indicator {r..} t * h s (-t)) ⟶
           (LBINT t. indicator {0..} t * h s (-t))) (at_right 0)"
    proof (rule tendsto_at_right_sequentially)
      show "0 < (1::real)"
        by simp
    next
      fix l :: "nat ⇒ real"
      assume l: "l ⟶ 0" "⋀n. l n > 0"
      show "(λx. LBINT t. indicator {l x..} t * h s (-t)) ⟶ (LBINT
t. indicator {0..} t * h s (-t))"
      proof (rule integral_dominated_convergence)
        have "integrable lebesgue (λt. norm (indicator {0<..} t * complex_of_real
t powr (-s) / of_real (exp t)))"
          by (intro integrable_norm)
             (use absolutely_integrable_Gamma_integral'[of "1-s"] s
               in ‹simp_all add: set_integrable_def scaleR_conv_of_real
of_real_indicator›)
        hence "integrable lborel (λt. norm (indicator {0<..} t * complex_of_real
t powr (-s) / of_real (exp t)))"
          by (subst (asm) integrable_completion) auto
        also have "?this ⟷ integrable lborel (λt. indicator {0<..}
t * norm (h s (-t)))"
          by (intro Bochner_Integration.integrable_cong)
             (auto simp: norm_mult indicator_def norm_divide h_def powr_def
```

```
field_simps exp_diff exp_minus)
        finally show "integrable lborel (λt. indicator {0<..} t * norm
(h s (-t)))" .
      next
        show "AE t in lborel. norm (indicator {l n..} t * h s (- t))
                              ≤ indicat_real {0<..} t * cmod (h s (- t))"
for n
          using l(2)[of n] by (intro always_eventually) (auto simp: indicator_def)
      next
        have "AE t in lborel. t ≠ 0"
          by (metis AE_lborel_singleton)
        thus "AE t in lborel. (λn. indicator {l n..} t * h s (-t)) ⟶
indicator {0..} t * h s (-t)"
        proof eventually_elim
          case (elim t)
          show ?case
          proof (cases "t > 0")
            case True
            hence "eventually (λn. l n < t) at_top"
              using l(1) order_tendsto_iff by blast
            hence "eventually (λn. indicator {l n..} t * h s (-t) =
                                   indicator {0..} t * h s (-t)) sequentially"
              by eventually_elim (use True in auto)
            thus ?thesis
              by (metis tendsto_eventually)
          next
            case False
            hence "indicator {l n..} t * h s (-t) = indicator {0..} t
* h s (-t)" for n
              using l(2)[of n] elim by (auto simp: indicator_def)
            thus ?thesis
              by fastforce
          qed
        qed
      qed (auto simp: h_def)
    qed
    hence lim_integral: "((λr. CLBINT t:{r..}. h s (- t)) ⟶ CLBINT
t:{0..}. h s (-t)) (at_right 0)"
      by (simp add: set_lebesgue_integral_def scaleR_conv_of_real of_real_indicator)
```

For $r \to 0$, the integral in `RHS s r` becomes the integral over the entire positive real line. This integral is also exactly the one from the definition of Γ.

```
    have "filterlim (λr. RHS s r)
            (nhds (2 * sin (pi * s) * (CLBINT t:{0..}. h s (-t)) + i *
(0 - cnj 0))) (at_right 0)"
      unfolding RHS_def by (intro tendsto_intros lim_err2 lim_integral)
(use s in auto)
    hence "filterlim (λr. RHS s r) (nhds (2 * sin (pi * s) * (CLBINT t:{0..}.
```

```
h s (-t)))) (at_right 0)"
      by simp
```

Moreover, since `lhs s T` is independent of r but also tends to `RHS s r` for $T \to \infty$ for any r, we know that the value of `RHS s r` is actually also independent of r (at least for sufficiently small r).

```
    moreover have "eventually (λr'::real. r' > 0) (at_right 0)" "eventually
(λr'. r' < c) (at_right 0)"
      using c by real_asymp+
    hence "eventually (λr'. RHS s r' = RHS s r) (at_right 0)"
    proof eventually_elim
      case (elim r')
      have "(lhs s ⟶ RHS s r') at_top"
        by (rule lim_lhs) (use elim s in auto)
      moreover have "(lhs s ⟶ RHS s r) at_top"
        by (rule lim_lhs) (use r s in auto)
      ultimately show "RHS s r' = RHS s r"
        using tendsto_unique trivial_limit_at_top_linorder by blast
    qed
    ultimately have "filterlim (λr'. RHS s r)
                       (nhds (2 * sin (pi * s) * (CLBINT t:{0..}. h s
(-t)))) (at_right (0::real))"
      by (rule Lim_transform_eventually)
    hence "RHS s r = 2 * sin (pi * s) * (CLBINT t:{0..}. h s (-t))"
      by (simp add: tendsto_const_iff)
```

All that remains now is some massaging of terms and using the definition of the Gamma function as well as the reflection identity.

```
    also have "(CLBINT t:{0..}. h s (-t)) = (CLBINT t:{0<..}. of_real
t powr (-s) / of_real (exp t))"
      unfolding set_lebesgue_integral_def
    proof (intro integral_cong_AE)
      have "AE t in lborel. t ≠ (0::real)"
        using AE_lborel_singleton by blast
      thus "AE t in lborel. indicator {0..} t *_R h s (-t) =
                            indicator {0<..} t *_R (of_real t powr -s /
of_real (exp t))"
        by eventually_elim
           (auto simp: powr_def h_def exp_add exp_diff exp_minus field_simps

                       Ln_of_real exp_of_real indicator_def)
    qed (auto simp: h_def)
    also have "... = integral {0<..} (λt. of_real t powr (-s) / of_real
(exp t))"
      by (rule set_borel_integral_eq_integral(2))
         (use absolutely_integrable_Gamma_integral'[of "1-s"] s
          in ‹simp_all add: set_integrable_def integrable_completion›)
    also have "... = Gamma (1-s)"
      using Gamma_integral_complex'[of "1-s"] s by (simp add: has_integral_iff)
```

```
    also have "2 * sin (pi * s) * Gamma (1 - s) = 2 * pi * rGamma s *
(Gamma (1 - s) * rGamma (1 - s))"
      using rGamma_reflection_complex[of s] by (simp add: field_simps)
    also have "1 - s ∉ ℤ≤0"
      using s by (auto elim!: nonpos_Ints_cases simp: complex_eq_iff)
    hence "Gamma (1 - s) * rGamma (1 - s) = 1"
      by (simp add: rGamma_inverse_Gamma field_simps Gamma_eq_zero_iff)
    finally show "RHS s r = 2 * pi * rGamma s"
      by simp
  qed
```

Also, for any $r > 0$, the term `RHS s r` defines a holomorphic function on the half-plane with $\mathrm{Re}(s) > 0$.

```
  have RHS_holo: "(λs. RHS s r) holomorphic_on {s. Re s > 0}"
  proof -
    have "(λz. err2 z r) = (λz. -contour_integral (part_circlepath 0 r
0 pi) (g z))"
      unfolding err2_def Cs_def by (metis contour_integral_part_circlepath_reverse)
    also have "... = (λz. -integral {0..pi} (λt. i * g z (rcis r t) * rcis
r t))"
      by (subst contour_integral_part_circlepath_eq) (simp_all add: mult_ac
rcis_def)
    finally have err2_eq: "err2 z r = -integral (cbox 0 pi) (λt. i * g
z (rcis r t) * rcis r t)" for z
      by (auto simp: fun_eq_iff)

    define err2' where
      "err2' = (λa b s. integral {a..b} (λt. i * exp (rcis r t - s * Complex
(ln r) t) * rcis r t))"

    have err2'_holomorphic: "err2' a b holomorphic_on {s. Re s > 0}" for
a b
    proof -
      define l where "l = (λs t. i * exp (rcis r t + (1 - s) * Complex
(ln r) t))"
      define l' where "l' = (λs t. -i * exp (rcis r t + (1 - s) * Complex
(ln r) t) * Complex (ln r) t)"
      have "(λs. integral (cbox a b) (l s)) holomorphic_on {s. Re s >
0}"
      proof (rule leibniz_rule_holomorphic)
        show "((λs. l s t) has_field_derivative l' s t) (at s within
{s. Re s > 0})"
          if s: "s ∈ {s. Re s > 0}" and t: "t ∈ cbox a b" for s t
          unfolding l_def l'_def by (auto intro!: derivative_eq_intros)
      next
        show "l s integrable_on cbox a b" if "s ∈ {s. Re s > 0}" for
s
          unfolding l_def by (intro integrable_continuous continuous_intros)
      next
```

```
      show "continuous_on ({s. 0 < Re s} × cbox a b) (λ(s, t). l' s
t)"
        unfolding l'_def case_prod_unfold by (auto intro!: continuous_intros)
    qed (auto simp: convex_halfspace_Re_gt)
    also have "(λs. integral (cbox a b) (l s)) = err2' a b"
      unfolding err2'_def cbox_interval using r
      by (intro ext integral_cong)
         (auto simp: exp_diff l_def exp_add ring_distribs rcis_def cis_conv_exp
Complex_eq exp_of_real)
    finally show ?thesis .
  qed

  have "err2 s r = -err2' 0 pi s" for s
  proof -
    have "err2 s r = -integral {0..pi} (λx. i * exp (rcis r x - s *
Ln' (rcis r x)) * rcis r x)"
      unfolding err2_eq by (simp add: integral_cnj g_def exp_cnj rcis_def
cis_cnj)
    also have "integral {0..pi} (λt. i * exp (rcis r t - s * Ln' (rcis
r t)) * rcis r t) =
                integral {0..pi} (λt. i * exp (rcis r t - s * Complex
(ln r) t) * rcis r t)"
    proof (intro integral_cong)
      fix t assume t: "t ∈ {0..pi}"
      have "Ln' (rcis r t) = Ln (rcis r (t - pi / 2)) + i * pi / 2"
        unfolding Ln'_def by (simp add: rcis_def mult_ac flip: cis_divide)
      also have "... = Complex (ln r) t"
        by (subst Ln_rcis) (use r t pi_gt_zero in ‹auto simp del: pi_gt_zero
simp: complex_eq_iff›)
      finally show "i * exp (rcis r t - s * Ln' (rcis r t)) * rcis r
t =
                    i * exp (rcis r t - s * Complex (ln r) t) * rcis
r t" by simp
    qed
    finally show ?thesis
      by (simp add: err2'_def)
  qed
  hence err2_holomorphic1: "(λs. err2 s r) holomorphic_on {s. Re s >
0}"
    using holomorphic_on_minus[OF err2'_holomorphic[of 0 pi]] by simp

  have "cnj (err2 (cnj s) r) = err2' (-pi) 0 s" for s
  proof -
    have "cnj (err2 (cnj s) r) =
          integral {0..pi} (λx. i * exp (rcis r (-x) - s * cnj (Ln'
(rcis r x))) * (rcis r (-x)))"
      unfolding err2_eq by (simp add: integral_cnj g_def exp_cnj rcis_def
cis_cnj)
    also have "... = integral {0..pi} (λt. i * exp (rcis r (-t) - s *
```

```
Complex (ln r) (-t)) * rcis r (-t))"
      proof (intro integral_cong)
        fix t assume t: "t ∈ {0..pi}"
        have "Ln' (rcis r t) = Ln (rcis r (t - pi / 2)) + i * pi / 2"
          unfolding Ln'_def by (simp add: rcis_def mult_ac flip: cis_divide)
        also have "... = Complex (ln r) t"
          by (subst Ln_rcis) (use r t pi_gt_zero in <auto simp del: pi_gt_zero
simp: complex_eq_iff>)
        also have "cnj ... = Complex (ln r) (-t)"
          by (simp add: complex_eq_iff)
        finally show "i * exp (rcis r (-t) - s * cnj (Ln' (rcis r t)))
* rcis r (-t) =
                        i * exp (rcis r (-t) - s * Complex (ln r) (-t)) *
rcis r (-t)" by simp
      qed
      also have "... = integral {-pi..0} (λt. i * exp (rcis r t - s * Complex
(ln r) t) * rcis r t)"
        by (subst Henstock_Kurzweil_Integration.integral_reflect_real
[symmetric]) simp_all
      finally show ?thesis
        by (simp add: err2'_def)
    qed
    hence err2_holomorphic2: "(λs. cnj (err2 (cnj s) r)) holomorphic_on
{s. Re s > 0}"
      using err2'_holomorphic[of "-pi" 0] by simp

    have holo_integral: "(λz. CLBINT t:{r..}. h z (-t)) holomorphic_on
UNIV"
    proof -
      have "(λz. Gamma_incu (1-z) (of_real r)) holomorphic_on UNIV"
        using r by (auto intro!: analytic_imp_holomorphic analytic_intros)
      also have "(λz. Gamma_incu (1-z) (of_real r)) = (λz. LBINT t:{r..}.
h z (-t))"
      proof
        fix z :: complex
        have "Gamma_incu (1-z) (of_real r) = integral {r..} (λt. h z (-t))"
          using has_integral_Gamma_incu_complex_of_real'[of r "1-z"] r
          by (simp add: h_def minus_diff_commute has_integral_iff)
        also have "... = (LBINT t:{r..}. h z (-t))"
          by (intro set_borel_integral_eq_integral(2) [symmetric] integrable
r)
        finally show "Gamma_incu (1 - z) (complex_of_real r) = (LBINT t:{r..}.
h z (- t))" .
      qed
      finally show ?thesis .
    qed
    show "(λs. RHS s r) holomorphic_on {s. Re s > 0}" unfolding RHS_def
      by (intro holomorphic_intros err2_holomorphic1 err2_holomorphic2
               holomorphic_on_subset[OF holo_integral]) auto
```

```
  qed
```

Thus, we can extend the validity of our equation to the half-plane $\mathrm{Re}(s) > 0$ by analytic continuation.

```
  have RHS_eq': "RHS s r = 2 * pi * rGamma s"
  proof (rule analytic_continuation_open[where f = "λs. RHS s r"])
    have "Re (1/2) ∈ {0<..<1}"
      by simp
    thus "{s. Re s ∈ {0<..<1}} ≠ {}"
      by blast
  next
    have "open ({s. Re s > 0} ∩ {s. Re s < 1})"
      by (intro open_Int open_halfspace_Re_lt open_halfspace_Re_gt)
    also have "... = {s. Re s ∈ {0<..<1}}"
      by auto
    finally show "open {s. Re s ∈ {0<..<1}}" .
  next
    show "(λa. complex_of_real (2 * pi) * rGamma a) holomorphic_on {s.
Re s > 0}"
      by (intro holomorphic_intros)
  next
    show "RHS s r = complex_of_real (2 * pi) * rGamma s" if s: "s ∈ {s.
Re s ∈ {0<..<1}}" for s
      using RHS_eq[of s] s r by simp
  qed (use s RHS_holo in ‹auto simp: open_halfspace_Re_gt›)

  show ?thesis
    using lim_lhs[of r s] RHS_eq' r s by (simp add: RHS_def lhs_def f_def
o_def)
qed
```

If $\mathrm{Re}(s) > 1$, the integral exists not only as an improper/Cauchy principal value style integral, but as a Lebesgue integral.

```
corollary rGamma_hankel:
  assumes c: "c > 0" and s: "Re s > 1"
  shows   "(((λz. exp z * z powr (-s)) ∘ (λt. Complex c t))
             has_integral (2 * of_real pi * rGamma s)) UNIV"
proof -
  have "((λT. integral {-T..T} ((λz. exp z * z powr - s) ∘ Complex c))
⟶
          2 * of_real pi * rGamma s) at_top"
    by (rule rGamma_hankel_strong) (use assms in auto)
  also have "(λT. integral {-T..T} ((λz. exp z * z powr - s) ∘ Complex
c)) =
             (λT. set_lebesgue_integral lebesgue {-T..T} ((λz. exp z *
z powr - s) ∘ Complex c))"
    by (intro ext set_lebesgue_integral_eq_integral(2) [symmetric]
                absolutely_integrable_continuous_real)
       (use c s in ‹auto simp: complex_eq_iff intro!: continuous_intros›)
```

```
  finally have "((λT. set_lebesgue_integral lebesgue {-T..T} ((λz. exp
z * z powr - s) ∘ Complex c)) ⟶
                  2 * of_real pi * rGamma s) at_top" .
  hence "((λT. set_lebesgue_integral lebesgue {-real T..real T} ((λz.
exp z * z powr - s) ∘ Complex c)) ⟶
                2 * of_real pi * rGamma s) at_top"
    by (rule filterlim_compose) (rule filterlim_real_sequentially)

  moreover have
    "(λT. set_lebesgue_integral lebesgue {-T..T} ((λz. exp z * z powr
- s) ∘ Complex c)) ⟶
       lebesgue_integral lebesgue ((λz. exp z * z powr - s) ∘ Complex
c)"
    unfolding set_lebesgue_integral_def
  proof (intro integral_dominated_convergence always_eventually allI)
    show "integrable lebesgue (λt. norm (exp (Complex c t) * Complex
c t powr -s))"
      using rGamma_hankel_integrable[of c s] c s
      by (intro integrable_norm) (simp_all add: set_integrable_def)
  next
    fix t :: real
    have "eventually (λT. real T ≥ |t|) at_top"
      by real_asymp
    hence "eventually (λT. indicat_real {-real T..real T} t *R ((λz.
exp z * z powr - s) ∘ Complex c) t =
                          ((λz. exp z * z powr - s) ∘ Complex c) t) at_top"
      by eventually_elim (auto simp: indicator_def)
    thus "(λT. indicat_real {-real T..real T} t *R ((λz. exp z * z powr
- s) ∘ Complex c) t)
            ⟶ ((λz. exp z * z powr - s) ∘ Complex c) t"
      by (rule tendsto_eventually)
  qed (auto intro!: measurable_completion simp: indicator_def o_def Complex_eq)

  ultimately have "2 * of_real pi * rGamma s =
                     lebesgue_integral lebesgue ((λz. exp z * z powr -
s) ∘ Complex c)"
    using LIMSEQ_unique by blast
  also have "... = integral UNIV ((λz. exp z * z powr - s) ∘ Complex c)"
    using rGamma_hankel_integrable[OF c s]
    by (intro integral_lebesgue [symmetric]) (simp_all add: set_integrable_def)
  moreover have "((λz. exp z * z powr - s) ∘ Complex c) integrable_on
UNIV"
    using rGamma_hankel_integrable[OF c s] absolutely_integrable_on_def
by blast
  ultimately show ?thesis
    by (simp add: has_integral_iff)
qed
```

2.3 The integral for the Bessel function

We now show the main result of this section: A Hankel-style contour-integral representation for the modified Bessel function of the first kind $I(\nu, s)$, valid for all s and ν with $\mathrm{Re}(\nu) > 0$.

The integral is of the form $\int_{c-i\infty}^{c+i\infty}$ for any real $c > 0$ and converges absolutely. The proof is modelled after Proposition 2.4 by Kong and Teo [2] and works by taking the similar integral formula for $1/\Gamma(s)$, summing over it, and exchanging the order of summation and integration.

The proof is fairly quick since the main work was already done in other lemmas. The main bit of work that we still have to do here is to justify exchanging summation and integration. This is done by showing that the integrals over the absolute value exist for every summand, and then the sum over these integrals also exists.

The integrals over the summand can be brought into the form

$$\int_{-\infty}^{\infty} (c^2 + t^2)^a \, \mathrm{d}t$$

for some suitable $a > \frac{1}{2}$, which we showed earlier to have a closed form in terms of the Beta function.

```
theorem
  fixes c :: real and s v :: complex
  assumes v: "Re v > 0" and c: "c > 0"
  shows   has_integral_Bessel_I_complex:
           "(((λz. 1 / (2*pi) * (s/2) powr v * exp (z + s^2/(4*z)) / z
powr (v+1)) ∘ Complex c)
                has_integral Bessel_I v s) UNIV"
    and   absolutely_integrable_Bessel_I_complex:
           "((λz. 1 / (2*pi) * (s/2) powr v * exp (z + s^2/(4*z)) / z
powr (v+1)) ∘ Complex c)
              absolutely_integrable_on UNIV"
proof -
  from v have v': "v ∉ ℤ≤0"
    by (auto elim!: nonpos_Ints_cases)
  define LINTL :: "(real ⇒ complex) ⇒ complex" ("∫ ")
    where "LINTL = lebesgue_integral lebesgue"

  have 1: "integrable lebesgue (λx. (s/2)^(2*j) / fact j * exp (Complex
c x) *
                                      Complex c x powr - (v + of_nat
j + 1))" for j
  proof -
    have "integrable lebesgue (λx. (s/2)^(2*j) / fact j *
              (exp (Complex c x) * Complex c x powr - (v + of_nat j +
1)))"
      using rGamma_hankel_integrable[of c "v + of_nat j + 1"] c v
```

```
    by (intro integrable_mult_right) (simp add: set_integrable_def)
  thus ?thesis
    by (simp add: algebra_simps)
qed

have 2: "summable (λj. norm ((s/2)^(2*j) / fact j * exp (Complex c t)
*
                       Complex c t powr -(v + of_nat j + 1)))" for t
:: real
proof -
  define z where "z = Complex c t"
  define C where "C = exp c * exp (Im v * Arg z) * norm z powr (-Re
v-1)"
  have "summable (λj. C * (inverse (fact j) * (norm s ^ 2 / (4 * norm
z)) ^ j))"
    by (intro summable_mult summable_exp)
  also have "... = (λj. norm ((s/2)^(2*j) / fact j * exp (Complex c
t) *
                         Complex c t powr -(v + of_nat j + 1)))"
    by (intro ext)
       (simp_all add: norm_mult norm_divide norm_power power_mult norm_powr_complex
z_def
                       powr_diff powr_minus divide_simps exp_minus powr_add
C_def powr_realpow)
  finally show "summable ..." .
qed

have 3: "summable (λj. LINT t|lebesgue. norm ((s/2)^(2*j) / fact j *

                      exp (Complex c t) * Complex c t powr - (v + of_nat
j + 1)))"
proof -
  define C where "C = exp c * (Beta (1 / 2) (Re v / 2) * exp (|Im v|
* pi)) * c powr (-Re v)"
  show ?thesis
  proof (rule summable_comparison_test')
    fix j :: nat
    assume j: "j ≥ 0"
    show "norm (LINT t|lebesgue. norm ((s/2)^(2*j) / fact j *
                    exp (Complex c t) * Complex c t powr - (v + of_nat
j + 1))) ≤
            C * (inverse (fact j) * (norm s ^ 2 / (4 * c)) ^ j)"
    proof -
      define e where "e = (Re v + real j + 1)/2"
      define I where "I = c powr (1-2*e) / 2 * Beta (1/2) (e - 1/2)"

      have integrable': "integrable lebesgue (λt. (c^2 + t^2) powr -e)"
       and integral': "lebesgue_integral lebesgue (λt. (c^2 + t^2) powr
-e) = 2 * I"
```

```
        using Bessel_I_aux_integral2[of c e] assms by (simp_all add:
e_def I_def)

      have "norm (LINT t|lebesgue. norm ((s/2)^(2*j) / fact j * exp (Complex
c t) *
                                    Complex c t powr -(v + of_nat j
+ 1))) =
            (LINT t|lebesgue. norm ((s/2)^(2*j) / fact j * exp (Complex
c t) *
                                  Complex c t powr -(v + of_nat j + 1)))"
        unfolding real_norm_def
        by (intro abs_of_nonneg Bochner_Integration.integral_nonneg)
auto
       also have "... = exp c * (norm s / 2) ^ (2 * j) / fact j *
                         (LINT t|lebesgue. norm (Complex c t) powr (-Re
v - real j - 1) *
                                             exp (Im v * Arg (Complex c
t)))"
        by (simp add: norm_power norm_mult norm_divide norm_powr_complex)
       also have "(LINT t|lebesgue. norm (Complex c t) powr (-Re v -
real j - 1) *
                                      exp (Im v * Arg (Complex c t)))
=
                  (LINT t|lebesgue. (c^2 + t^2) powr (-e) * exp (Im v *
Arg (Complex c t)))"
        by (intro Bochner_Integration.integral_cong)
           (auto simp: cmod_def powr_powr e_def add_divide_distrib diff_divide_distrib
                 simp flip: powr_half_sqrt)
       also have "... ≤ (LINT t|lebesgue. (c^2 + t^2) powr (-e) * exp (|Im
v| * pi))"
       proof (rule integral_mono')
         show "integrable lebesgue (λx. (c^2 + x^2) powr -e * exp (|Im
v| * pi))"
           using integrable' unfolding set_integrable_def by (intro integrable_mult_left)
         show "(c^2 + t^2) powr -e * exp (Im v * Arg (Complex c t)) ≤

                 (c^2 + t^2) powr -e * exp (|Im v| * pi)" for t
         proof -
           have "Im v * Arg (Complex c t) ≤ |Im v * Arg (Complex c t)|"
             by linarith
           also have "... ≤ |Im v| * pi"
             unfolding abs_mult using Arg_bounded[of "Complex c t"] by
(intro mult_left_mono) auto
           finally show ?thesis
             by (intro mult_mono) auto
         qed
       qed auto
       also have "... = exp (|Im v| * pi) * (LINT t|lebesgue. (c^2 + t^2)
powr (-e))"
```

```
        by simp
      also have "... = 2 * I * exp (|Im v| * pi)"
        by (subst integral') simp
      also have "I ≤ Beta (1 / 2) (Re v / 2) / 2 * c powr (1 - 2 *
e)"
      proof -
        have "I = Beta (1 / 2) (e - 1 / 2) / 2 * c powr (1 - 2 * e)"
          by (simp add: I_def mult_ac)
        also have "Beta (1 / 2) (e - 1 / 2) ≤ Beta (1 / 2) (Re v / 2)"
          by (rule Beta_real_mono) (use v in ‹simp_all add: e_def add_divide_distrib›)
        finally show ?thesis
          using c by simp
      qed
      also have "exp c * (cmod s / 2) ^ (2 * j) / fact j *
                   (2 * (Beta (1 / 2) (Re v / 2) / 2 * c powr (1 - 2
* e)) * exp (|Im v| * pi)) =
                 C * (inverse (fact j) * ((cmod s)^2 / (4 * c)) ^ j)"
using c
        by (simp add: C_def divide_simps e_def powr_diff powr_minus
powr_realpow power_mult diff_divide_distrib)
      finally show ?thesis
        by (simp add: mult_left_mono mult_right_mono divide_right_mono)
    qed
  next
    show "summable (λj. C * (inverse (fact j) * (norm s ^ 2 / (4 *
c)) ^ j))"
      by (intro summable_mult summable_exp)
  qed
qed

have sum_eq:
   "(λt. ∑j. (s/2)^(2*j) / fact j * exp (Complex c t) * Complex c t
powr -(v + of_nat j + 1)) =
    (λz. exp (z + s^2/(4*z)) * z powr -(v+1)) ∘ (λt. Complex c t)"
  (is "?lhs = ?rhs")
proof
  fix t :: real
  define z where "z = Complex c t"
  have "(λj. exp z * z powr -(v+1) * ((s^2/(4 * z))^j /R fact j)) sums
          ((exp z * z powr -(v+1)) * exp (s^2/(4 * z)))"
    by (intro sums_mult exp_converges)
  also have "(exp z * z powr -(v+1)) * exp (s^2/(4*z)) = exp (z + s^2/(4*z))
* z powr -(v+1)"
    by (simp add: exp_add field_simps)
  also have "(λj. exp z * z powr -(v+1) * ((s^2/(4 * z))^j /R fact j))
=
              (λj. (s/2)^(2*j) / fact j * exp z * z powr -(v + of_nat
j + 1))"
    apply (rule ext)
```

```
    apply (simp add: powr_diff powr_minus power_divide power_mult powr_nat
scaleR_conv_of_real)
    apply (simp add: field_simps)?
    done
  finally show "?lhs t = ?rhs t"
    by (simp add: sums_iff z_def)
qed


have "integrable lebesgue
        (λt. ∑j. (s/2)^(2*j) / fact j * exp (Complex c t) * Complex
c t powr -(v + of_nat j + 1))"
  by (intro integrable_suminf always_eventually allI 1 2 3)
hence "integrable lebesgue ((λz. exp (z + s^2 / (4 * z)) * z powr -
(v + 1)) ∘ Complex c)"
  unfolding sum_eq .
hence "integrable lebesgue ((λz. ((s/2) powr v / (2*pi)) *
                                (exp (z + s^2 / (4 * z)) * z powr - (v
+ 1))) ∘ Complex c)"
  unfolding o_def by (intro integrable_mult_right)
thus integrable:
   "((λz. 1 / (2*pi) * (s/2) powr v * exp (z + s^2 / (4 * z)) / z powr
(v + 1))) ∘ Complex c
      absolutely_integrable_on UNIV"
  by (simp add: o_def set_integrable_def powr_minus powr_diff powr_add
field_simps)


have "(λj. ∫ ((λz. (s/2)^(2*j) / fact j * exp z * z powr -(v + of_nat
j + 1)) ∘ Complex c)) sums
        ∫ (λt. ∑j. (s/2)^(2*j) / fact j * exp (Complex c t) * Complex
c t powr -(v + of_nat j + 1))"
  unfolding LINTL_def o_def by (intro sums_integral always_eventually
allI 1 2 3)
also have "(λj. ∫ ((λz. (s/2)^(2*j) / fact j * exp z * z powr -(v +
of_nat j + 1)) ∘ Complex c)) =
            (λj. (s/2)^(2*j) / fact j * 2 * pi * rGamma (v + of_nat j
+ 1))" (is "?lhs = ?rhs")
proof
  fix j :: nat
  have "∫ ((λz. (s/2)^(2*j) / fact j * exp z * z powr -(v + of_nat
j + 1)) ∘ Complex c) =
          (s/2)^(2*j) / fact j * ∫ ((λz. exp z * z powr -(v + of_nat
j + 1)) ∘ Complex c)"
    unfolding o_def LINTL_def
    by (subst integral_mult_right_zero [symmetric], rule Bochner_Integration.integral_con
simp_all
  also have "∫ ((λz. exp z * z powr -(v + of_nat j + 1)) ∘ Complex
c) =
```

```
              integral UNIV ((λz. exp z * z powr -(v + of_nat j + 1))
∘ Complex c)" unfolding LINTL_def
      by (rule integral_lebesgue [symmetric])
         (use rGamma_hankel_integrable[of c "v + of_nat j + 1"] c v in
‹simp_all add: set_integrable_def›)
    also have "... = 2 * pi  * rGamma (v + of_nat j + 1)"
      using rGamma_hankel[of c "v + of_nat j + 1"] c v
      by (simp add: has_integral_iff field_simps)
    finally show "?lhs j = ?rhs j" by (simp add: o_def)
  qed
  also note sum_eq
  finally have "(λj. ((s/2) powr v / (2*pi)) *
                      ((s/2)^(2*j) / fact j * 2 * of_real pi * rGamma
(v + of_nat j + 1))) sums
                  (((s/2) powr v / (2*pi)) * (∫ ((λz. exp (z + s^2/(4*z))
* z powr -(v+1)) ∘ Complex c)))"
    by (intro sums_mult)

  also have "(λj. ((s/2) powr v / (2*pi)) * ((s/2)^(2*j) / fact j * 2
* of_real pi * rGamma (v + of_nat j + 1))) =
             (λj. (s/2) powr (v + 2 * of_nat j) * rGamma (v + of_nat j
+ 1) / fact j)"
    by (intro ext; cases "s = 0") (auto simp: powr_add simp flip: powr_nat')
  finally have "(λj. (s/2) powr (v + 2 * of_nat j) * rGamma (v + of_nat
j + 1) / fact j) sums
                  (((s/2) powr v / (2 * pi)) * (∫ ((λz. exp (z + s^2 /
(4 * z)) * z powr - (v + 1)) ∘ Complex c)))" .
  also have "((s/2) powr v / (2 * pi)) * ∫ ((λz. exp (z + s^2 / (4 * z))
* z powr - (v + 1)) ∘ Complex c) =
             (∫ ((λz. 1 / (2 * pi) * (s/2) powr v * exp (z + s^2 / (4
* z)) / z powr (v+1)) ∘ Complex c))"
    unfolding LINTL_def
    by (subst integral_mult_right_zero [symmetric], rule Bochner_Integration.integral_cong)

       (simp_all add: o_def powr_diff powr_add powr_minus field_simps)
  finally have "(λj. (s/2) powr (v + 2 * of_nat j) * rGamma (v + of_nat
j + 1) / fact j) sums
                  (∫ ((λz. 1 / (2 * pi) * (s/2) powr v * exp (z + s^2/(4*z))
/ z powr (v+1)) ∘ Complex c))" .

  moreover have "(λj. (s/2) powr (v + 2 * of_nat j) * rGamma (v + of_nat
j + 1) / fact j) sums
                  Bessel_I v s"
  proof -
    have "(λj. (s/2) powr' (v + 2 * of_nat j) * rGamma (v + of_nat j +
1) / fact j) sums
                 Bessel_I v s"
      using sums_Bessel_I_complex[of s v] v' by (simp add: rGamma_inverse_Gamma
field_simps)
```

```
    also have "(λj. (s/2) powr' (v + 2 * of_nat j)) = (λj. (s/2) powr
(v + 2 * of_nat j))"
      (is "?lhs = ?rhs")
    proof
      fix j :: nat
      from v' have "v + 2 * complex_of_nat j ≠ 0"
        by (metis v' add.commute left_add_twice plus_of_nat_eq_0_imp plus_of_nat_notin_nonp
      thus "?lhs j = ?rhs j"
        by (cases "s = 0") (auto simp: powr'_complex powr'_0_left_if)
    qed
    finally show ?thesis .
  qed
  ultimately have eq:
    "Bessel_I v s =
       (∫ ((λz. 1 / (2 * pi) * (s/2) powr v * exp (z + s^2/(4*z)) / z
powr (v+1)) ∘ Complex c))"
    using sums_unique2 by blast
  also have "... = integral UNIV ((λz. 1 / (2 * pi) * (s/2) powr v *
                   exp (z + s^2/(4*z)) / z powr (v+1)) ∘ Complex c)"
    unfolding LINTL_def using integrable unfolding set_integrable_def
    by (intro integral_lebesgue [symmetric]) simp
  finally show "(((λz. 1 / (2*pi) * (s/2) powr v * exp (z + s^2/(4*z))
/ z powr (v+1)) ∘ Complex c)
                has_integral Bessel_I v s) UNIV"
    using set_lebesgue_integral_eq_integral(1)[OF integrable] assms
    by (simp add: has_integral_iff)
qed

lemma has_integral_Bessel_I_complex':
  fixes x :: complex and c :: real and v :: complex and f :: "complex
⇒ complex"
  defines "f ≡ (λt. exp (t + x^2 / t) / t powr v)"
  assumes c: "c > 0" and v: "Re v > 1" and x: "x ≠ 0"
  shows   "((f ∘ (λt. Complex c t)) has_integral
             (2 * pi * Bessel_I (v-1) (2*x)) / (x powr (v-1))) UNIV"
proof -
  define a where "a = 2 * pi / (x powr (v - 1))"
  define g where "g = (λt. (1 / (2 * pi)) * x powr (v - 1) * exp (t +
x^2 / t) / t powr v) ∘ (λt. Complex c t)"
  have "(g has_integral Bessel_I (v - 1) (2 * x)) UNIV"
    using has_integral_Bessel_I_complex[of "v-1" c "2*x"] assms unfold-
ing g_def by (simp add: o_def)
  hence "((λt. a * g t) has_integral (a * Bessel_I (v - 1) (2 * x))) UNIV"
    by (rule has_integral_mult_right)
  also have "(λt. a * g t) = f ∘ (λt. Complex c t)"
    using x by (auto simp: a_def g_def f_def fun_eq_iff)
  finally show "((f ∘ (λt. Complex c t)) has_integral
                (2 * pi * Bessel_I (v - 1) (2 * x)) / (x powr (v - 1)))
UNIV"
```

```
    by (simp add: a_def o_def)
qed

lemma absolutely_integrable_Bessel_I_complex':
  fixes c :: real and s v :: complex
  assumes v: "Re v > 1" and c: "c > 0"
  shows   "((λz. exp (z + s/z) / z powr v) ∘ Complex c) absolutely_integrable_on
UNIV"
proof (cases "s = 0")
  case [simp]: False
  define x where "x = csqrt s"
  define C where "C = 1 / (2*pi) * (2 * csqrt s / 2) powr (v-1)"
  have [simp]: "C ≠ 0"
    by (simp add: C_def)
  have "(λz. inverse C * ((λz. C * exp (z + (2 * csqrt s)^2/(4*z)) / z
powr (v-1+1)) ∘ Complex c) z)
          absolutely_integrable_on UNIV" unfolding C_def
    by (intro set_integrable_mult_right absolutely_integrable_Bessel_I_complex)
(use v c in auto)
  also have "(λz. inverse C * ((λz. C * exp (z + (2 * csqrt s)^2/(4*z))
/ z powr (v-1+1)) ∘ Complex c) z) =
             (λz.  exp (z + s / z) / z powr v) ∘ Complex c"
    by (simp add: fun_eq_iff)
  finally show ?thesis .
next
  case [simp]: True
  show ?thesis using rGamma_hankel_integrable[of c v] c v
    by (simp add: powr_minus field_simps)
qed

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.

[2] Z.-Y. Kong and L.-P. Teo. Rademacher's formula for the partition function, 2023.