Generalised Hypergeometric Series

Manuel Eberl

July 8, 2026

Abstract

This entry provides a formalisation of the generalized hypergeometric series ${}_pF_q$, both as a formal power series and as a function in a Banach space. It is defined, for parameters $\mathbf{a} = (a_1, \ldots, a_p)$ and $\mathbf{b} = (b_1, \ldots, b_q)$ with $b_i \notin \mathbb{Z}_{\leq 0}$ as the exponential power series

$$F(\mathbf{a}; \mathbf{b}; z) = \sum_{n \geq 0} \frac{a_1^{\overline{n}} \cdots a_p^{\overline{n}}}{b_1^{\overline{n}} \cdots b_q^{\overline{n}}} \frac{z^n}{n!}$$

where $a^{\overline{n}} = a(a+1)\ldots(a+n-1)$ is the Pochhammer symbol.

Basic properties of ${}_pF_q$ are proven (uniform convergence, continuity, holomorphicity), as well as some important properties for specific instances, such as:

- representations of various trigonometric and hyperbolic functions in terms of ${}_0F_1$ and ${}_2F_1$
- the contiguous identities for ${}_2F_1$ and ${}_1F_1$
- the transformation identity ${}_1F_1(a; b; z) = e^z {}_1F_1(b-a; b; -z)$ for Kummer's confluent hypergeometric function
- the fact that ${}_1F_1$ is a solution of the ODE $aW(x) - (b-x)W'(x) - xW''(x) = 0$
- the fact that the error function can be expressed in terms of ${}_1F_1$ as $\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} z {}_1F_1(\frac{1}{2}; \frac{3}{2}; -z^2)$

A regularised variant of ${}_pF_q$ that is also defined if $b_i \in \mathbb{Z}_{\leq 0}$ is also provided.

Contents

1 Generalized Hypergeometric Series

theory *Generalized_Hypergeometric_Series*
imports
"HOL-Complex_Analysis.Complex_Analysis"
"HOL-Real_Asymp.Real_Asymp"
"Error_Function.Error_Function"
begin

1.1 Definition as a formal power series

Let $\mathbf{a} = (a_1, \ldots, a_p)$ and $\mathbf{b} = (b_1, \ldots, b_q)$. The typical notation for the hypergeometric function is

$$ {}_pF_q(a_1, \ldots, a_p; b_1, \ldots, b_q; z) \ .$$

We will instead use the somewhat more compact $F(\mathbf{a}, \mathbf{b}, z)$. We will always let p and q implicitly denote the length of the vectors $\mathbf{a}$ and $\mathbf{b}$, respectively. Note that $\mathbf{b}$ must not contain non-positive integers to avoid division by zero.

We first look at the hypergeometric series as an exponential generating function

$$F(\mathbf{a}, \mathbf{b}, z) = \sum_{n \geq 0} \frac{a_1^{\overline{n}} \cdots a_p^{\overline{n}}}{b_1^{\overline{n}} \cdots b_q^{\overline{n}}} \frac{z^n}{n!}$$

where $m^{\overline{n}} = m(m+1)\cdots(m+n-1)$ is the Pochhammer symbol.

Note that to be consistent with the rest of the library, we actually always consider ${}_pF_q(\mathbf{a}; \mathbf{b}; cz)$ for some arbitrary c rather than ${}_pF_q(\mathbf{a}; \mathbf{b}; z)$. It is, of course, easy to transfer results from each to the other, but this way we can express some things a bit more directly without using the composition operator.

lemmas *[simp del] = fps_hypergeo_nth*

lemma *fps_hypergeo_nth_aux: "foldl (λacc x. acc * f x) y xs = y * (∏ x←xs. f x)"*
by *(induction xs arbitrary: y) (auto simp: mult_ac)*

lemma *fps_hypergeo_nth:*
*"fps_nth (fps_hypergeo as bs c) n = (∏ a←as. pochhammer a n) / (∏ b←bs. pochhammer b n) * c ^ n / fact n"*
by *(simp add: fps_hypergeo_def fps_hypergeo_nth_aux)*

lemma *fps_hypergeo_cong:*
assumes *"mset as = mset as'" "mset bs = mset bs'" "c = c'"*
shows *"fps_hypergeo as bs c = fps_hypergeo as' bs' c'"*
using *assms* **by** *(simp add: fps_eq_iff fps_hypergeo_nth flip: prod_mset_prod_list)*

```
lemma fps_hypergeo_singleton_Nil [simp]:
  "fps_hypergeo [a] [] c = fps_compose (fps_binomial (-a)) (-fps_const
c * fps_X)"
  by (rule fps_ext) (simp_all add: fps_hypergeo_def gbinomial_pochhammer
flip: power_mult_distrib)
```

Parameters appearing in both of the lists can be cancelled.

```
lemma fps_hypergeo_cancel:
  assumes "a ∉ ℤ≤0"
  shows "fps_hypergeo (a # as) (a # bs) c = fps_hypergeo as bs c"
  using pochhammer_eq_0_imp_nonpos_Int[of a] assms by (auto simp: fps_eq_iff
fps_hypergeo_nth)

lemma fps_hypergeo_0_left [simp]:
  assumes "0 ∈ set as"
  shows   "fps_hypergeo as bs c = 1"
proof (rule fps_ext)
  fix n :: nat
  show "fps_nth (fps_hypergeo as bs c) n = fps_nth 1 n"
  proof (cases "n = 0")
    case False
    hence "pochhammer 0 n = 0"
      by (auto simp: pochhammer_0_left)
    with assms have "∃a∈set as. pochhammer a n = 0"
      by blast
   thus ?thesis using assms
     by (auto simp: prod_list_zero_iff fps_hypergeo_nth)
 qed auto
qed
```

Next, we show a number of different expressions for the derivative of the hypergeometric series.

```
lemma fps_deriv_hypergeo1:
  fixes as bs :: "'a :: field_char_0 list"
  assumes "set bs ∩ ℤ≤0 = {}"
  defines "as' ≡ map (λa. a + 1) as"
  defines "bs' ≡ map (λb. b + 1) bs"
  shows "fps_deriv (fps_hypergeo as bs c) =
         fps_const (c * prod_list as) / fps_const (prod_list bs) * fps_hypergeo
as' bs' c"
proof -
  from assms(1) have "prod_list bs ≠ 0"
    by (auto simp: prod_list_zero_iff)
  thus ?thesis unfolding as'_def bs'_def
    by (auto simp: fps_eq_iff divide_simps o_def pochhammer_rec prod_list_distrib
fps_hypergeo_nth
             simp del: of_nat_Suc)
qed
```

```
lemma fps_deriv_hypergeo1':
  fixes as bs :: "'a :: field_char_0 list"
  assumes "0 ∉ set bs"
  defines "as' ≡ map (λa. a + 1) as"
  defines "bs' ≡ map (λb. b + 1) bs"
  shows "fps_const (prod_list bs) * fps_deriv (fps_hypergeo as bs c) =
         fps_const (c * prod_list as) * fps_hypergeo as' bs' c"
  using assms(1)
  by (auto simp: fps_eq_iff fps_hypergeo_nth as'_def bs'_def o_def pochhammer_rec
prod_list_distrib
                prod_list_zero_iff simp del: of_nat_Suc)

lemma fps_deriv_hypergeo2:
  fixes as bs :: "'a :: field_char_0 list" and a c :: 'a
  assumes "set bs ∩ ℤ≤0 = {}"
  defines "F ≡ fps_hypergeo (a # as) bs c"
  defines "G ≡ fps_hypergeo ((a + 1) # as) bs c"
  shows "fps_X * fps_deriv F + fps_const a * F = fps_const a * G"
proof -
  show ?thesis
  proof (rule fps_ext)
    fix n :: nat
    have nz: "(∏ b←bs. pochhammer b n) ≠ 0"
      using assms(1) pochhammer_eq_0_imp_nonpos_Int[of _ n] unfolding
prod_list_zero_iff by force
    have "of_nat n * (pochhammer a n * (∏ a←as. pochhammer a n)) * c
^ n +
            a * (pochhammer a n * (∏ a←as. pochhammer a n)) * c ^ n =
          pochhammer a n * (a + of_nat n) * (∏ a←as. pochhammer a n)
* c ^ n"
      by (simp add: algebra_simps)
    also have "pochhammer a n * (a + of_nat n) = pochhammer a (Suc n)"
      by (simp add: pochhammer_rec')
    also have "... = a * pochhammer (a + 1) n"
      by (simp add: pochhammer_rec)
    finally show "fps_nth (fps_X * fps_deriv F + fps_const a * F) n = fps_nth
(fps_const a * G) n"
      using nz by (auto simp: fps_eq_iff F_def G_def field_simps fps_hypergeo_nth)
  qed
qed

lemma fps_deriv_hypergeo3:
  fixes as bs :: "'a :: field_char_0 list" and b c :: 'a
  assumes "set bs ∩ ℤ≤0 = {}" "b ∉ ℤ≤0" "b ≠ 1"
  defines "F ≡ fps_hypergeo as (b # bs) c"
  defines "G ≡ fps_hypergeo as ((b - 1) # bs) c"
  shows "fps_X * fps_deriv F + fps_const (b - 1) * F = fps_const (b -
1) * G"
proof -
```

```
  show ?thesis
  proof (rule fps_ext)
    fix n :: nat
    have nz: "(∏ b←bs. pochhammer b n) ≠ 0"
      using assms(1) pochhammer_eq_0_imp_nonpos_Int[of _ n] unfolding
prod_list_zero_iff by force
    have "b - 1 ∉ ℤ≤0"
    proof
      assume "b - 1 ∈ ℤ≤0"
      then obtain k where "b - 1 = of_int k" "k ≤ 0"
        by (cases rule: nonpos_Ints_cases)
      hence b_eq: "b = of_int (k + 1)"
        by (auto simp: algebra_simps)
      from ‹b ∉ ℤ≤0› have "k = 0"
        unfolding b_eq of_int_in_nonpos_Ints_iff using ‹k ≤ 0› by (auto
simp: not_le)
      with ‹b ≠ 1› show False
        unfolding b_eq by auto
    qed
    hence nz': "pochhammer (b - 1) n ≠ 0" "pochhammer b n ≠ 0"
      using assms(2,3) by (auto dest!: pochhammer_eq_0_imp_nonpos_Int)

    have "(of_nat n * ((∏ a←as. pochhammer a n) * c ^ n) +
            (b - 1) * ((∏ a←as. pochhammer a n) * c ^ n)) * pochhammer
(b - 1) n =
          pochhammer (b - 1) n * (b - 1 + of_nat n) * (∏ a←as. pochhammer
a n) * c ^ n"
      by (simp add: algebra_simps)
    also have "pochhammer (b - 1) n * (b - 1 + of_nat n) = pochhammer
(b - 1) (Suc n)"
      by (simp add: pochhammer_rec')
    also have "... = (b - 1) * pochhammer b n"
      by (simp add: pochhammer_rec)
    finally show "fps_nth (fps_X * fps_deriv F + fps_const (b - 1) * F)
n = fps_nth (fps_const (b - 1) * G) n"
      using nz nz' assms(2,3) by (auto simp: fps_eq_iff F_def G_def divide_simps
fps_hypergeo_nth)
  qed
qed
```

The radius of convergence of a hypergeometric series ${}_pF_q(\mathbf{a};\mathbf{b};z)$ is easy to determine: if $p \leq q$, it is ∞. If $p = q+1$, it is 1. If $p > q$, it is 0.

Note that the formulation here is slightly more general since, again, it talks about ${}_pF_q(\mathbf{a};\mathbf{b};cz)$.

```
lemma fps_conv_radius_hypergeo:
  fixes as bs :: "'a :: {banach, real_normed_field} list"
  shows   "fps_conv_radius (fps_hypergeo as bs c) =
             (if length as ≤ length bs ∨ c = 0 ∨ set (as@bs) ∩ ℤ≤0 ≠
{} then ∞
```

```
            else if length as = length bs + 1 then 1 / ereal (norm c)
            else 0)" (is "_ = ?r")
proof (cases "c = 0 ∨ set (as@bs) ∩ ℤ≤0 ≠ {}")
  case True
  hence "eventually (λn. fps_nth (fps_hypergeo as bs c) n = 0) at_top"
  proof
    assume [simp]: "c = 0"
    show ?thesis
      using eventually_gt_at_top[of 0] by eventually_elim (auto simp:
fps_hypergeo_nth)
  next
    assume "set (as @ bs) ∩ ℤ≤0 ≠ {}"
    then obtain n where n: "-of_nat n ∈ set (as @ bs)"
      by (auto elim!: nonpos_Ints_cases')
    show ?thesis
      using eventually_gt_at_top[of n]
      by eventually_elim
         (use n in <auto simp: fps_hypergeo_nth prod_list_zero_iff image_iff
pochhammer_eq_0_iff>)
  qed
  hence "fps_conv_radius (fps_hypergeo as bs c) = fps_conv_radius (0 ::
'a fps)"
    unfolding fps_conv_radius_def by (intro conv_radius_cong') auto
  also have "... = ∞"
    by simp
  finally show ?thesis
    using True by auto
next
  case *: False
  have nz1: "(∏ a←as. norm (pochhammer a n)) ≠ 0" for n
    using * pochhammer_eq_0_imp_nonpos_Int by (force simp: prod_list_zero_iff)
  have nz2: "(∏ b←bs. norm (pochhammer b n)) ≠ 0" for n
    using * pochhammer_eq_0_imp_nonpos_Int by (force simp: prod_list_zero_iff)
  from * have [simp]: "c ≠ 0"
    by auto
  have "(λn. norm (fps_nth (fps_hypergeo as bs c) n) / norm (fps_nth (fps_hypergeo
as bs c) (Suc n))) =
        (λn. (∏ b←1#bs. norm (b + of_nat n)) / (∏ a←as. norm (a + of_nat
n)) / norm c)"
    using nz1 nz2
    apply (simp add: norm_mult norm_divide norm_power norm_prod_list o_def
pochhammer_Suc fps_hypergeo_nth)
    apply (auto simp: field_simps prod_list_distrib fun_eq_iff)?
    done
  hence "(λn. ereal (norm (fps_nth (fps_hypergeo as bs c) n) / norm (fps_nth
(fps_hypergeo as bs c) (Suc n)))) =
         (λn. ereal ((∏ b←1#bs. norm (b + of_nat n)) / (∏ a←as. norm
(a + of_nat n)) / norm c))"
    by (intro ext) (simp only: fun_eq_iff)
```

```
  also have "... ⟶ ?r"
  proof -
    let ?m = "(1 + int (length bs) - int (length as))"
    have [asymp_equiv_intros]: "(λx. norm (b + of_nat x)) ∼[sequentially]
real" for b :: 'a
    proof -
      have "(λx. norm (b + of_real x)) ∘ real ∼[sequentially] (λx. x)
∘ real"
        by (rule asymp_equiv_compose norm_plus_of_real_asymp_equiv filterlim_real_sequentia
      thus ?thesis
        by (simp add: o_def)
    qed
    let ?lhs = "(λn. (∏b←1#bs. norm (b + of_nat n)) / (∏a←as. norm
(a + of_nat n)) / norm c)"
    have "?lhs ∼[at_top] (λn. (∏b←1#bs. real n) / (∏a←as. real n)
/ norm c)"
      by (intro asymp_equiv_intros norm_plus_of_real_asymp_equiv)
    also have "... ∼[at_top] (λn. real n powi ?m / norm c)"
    proof (rule asymp_equiv_refl_ev)
      have "eventually (λn::nat. n > 0) at_top"
        by (rule eventually_gt_at_top)
      thus "eventually (λn. (∏b←1#bs. real n) / (∏a←as. real n) /
norm c =
                              real n powi ?m / norm c) at_top"
        by eventually_elim (auto simp: power_int_diff power_int_add)
    qed
    finally have "(λn. ereal (?lhs n)) ⟶ ?r ⟷ (λn. ereal (real
n powi ?m / norm c)) ⟶ ?r"
      by (rule tendsto_ereal_asymp_equiv_transfer)
    also have "(λn. ereal (real n powi ?m / norm c)) ⟶ ?r"
    proof (cases "?m ≤ 0")
      case True
      from True have "(λn. 1 / real n ^ (nat (-?m)) / norm c) ⟶
(if ?m = 0 then 1 / norm c else 0)"
        by (cases "?m = 0") (real_asymp simp: field_simps)+
      also have "(λn. 1 / real n ^ (nat (-?m)) / norm c) = (λn. real n
powi ?m / norm c)"
        using ‹?m ≤ 0› unfolding power_int_def by (auto simp: fun_eq_iff
field_simps)
      finally have "(λn. ereal (real n powi ?m / norm c)) ⟶ ereal
(if ?m = 0 then 1 / norm c else 0)"
        by (intro tendsto_intros)
      also have "ereal (if ?m = 0 then 1 / norm c else 0) = ?r"
        using * True by (auto simp: one_ereal_def)
      finally show ?thesis .
    next
      case False
      hence "filterlim (λn. real n ^ (nat ?m) / norm c) at_top at_top"
        by real_asymp
```

```
      also have "(λn. real n ^ nat (?m) / norm c) = (λn. real n powi ?m
/ norm c)"
        using False unfolding power_int_def by (auto simp: fun_eq_iff
field_simps)
      finally have "(λn. ereal (real n powi ?m / norm c)) ⟶ ∞"
        by (simp add: tendsto_PInfty_eq_at_top)
      also have "∞ = ?r"
        using * False by (auto simp: one_ereal_def)
      finally show ?thesis .
    qed
    finally show ?thesis .
  qed
  finally have lim: "(λn. ereal (norm (fps_nth (fps_hypergeo as bs c) n)
/ norm (fps_nth (fps_hypergeo as bs c) (Suc n)))) ⟶ ?r" .
  have nz: "∀F n in sequentially. fps_nth (fps_hypergeo as bs c) n ≠
0"
    using nz1 nz2 by (intro always_eventually) (auto simp: prod_list_zero_iff
image_iff fps_hypergeo_nth)
  show ?thesis using conv_radius_ratio_limit_ereal[OF nz lim]
    by (simp add: fps_conv_radius_def)
qed
```

We also define the following notion, which corresponds to the ordinary generating function

$$\sum_{n\geq 0} \frac{a_1^{\overline{n}} \cdots a_p^{\overline{n}}}{b_1^{\overline{n}} \cdots b_q^{\overline{n}}} z^n$$

This is a bit easier to deal with for our convergence arguments later on.

One can express the "normal" hypergeometric series in terms of this easily by adding $a_{p+1} = 1$ to the first parameter vector.

```
definition fps_hypergeo_aux :: "'a :: field_char_0 list ⇒ 'a list ⇒ 'a
fps" where
  "fps_hypergeo_aux as bs = Abs_fps (λn. (∏a←as. pochhammer a n) / (∏b←bs.
pochhammer b n))"

definition hypergeo_F_aux where
  "hypergeo_F_aux as bs = eval_fps (fps_hypergeo_aux as bs)"

lemma fps_nth_hypergeo_aux:
  "fps_nth (fps_hypergeo_aux as bs) n = (∏a←as. pochhammer a n) / (∏b←bs.
pochhammer b n)"
  by (simp add: fps_hypergeo_aux_def)

lemma fps_hypergeo_conv_fps_hypergeo_aux:
  "fps_hypergeo as bs c = fps_compose (fps_hypergeo_aux as (1 # bs)) (fps_const
c * fps_X)"
  unfolding fps_hypergeo_nth fps_hypergeo_aux_def fps_eq_iff pochhammer_fact
fps_nth_compose_linear
```

```
  by (simp add: field_simps)

lemma fps_hypergeo_aux_conv_fps_hypergeo:
  "fps_hypergeo_aux as bs = fps_hypergeo (1 # as) bs 1"
  unfolding fps_hypergeo_nth fps_hypergeo_aux_def fps_eq_iff fps_nth_compose_linear
  by (simp add: field_simps flip: pochhammer_fact)

lemma fps_hypergeo_aux_split_head:
  "fps_hypergeo_aux as bs =
     1 + fps_const (prod_list as / prod_list bs) * fps_X *
           fps_hypergeo_aux (map (λa. a+1) as) (map (λb. b+1) bs)" (is
"?lhs = ?rhs ")
proof (rule fps_ext)
  fix n :: nat
  show "fps_nth ?lhs n =  fps_nth ?rhs n"
  proof (cases n)
    case (Suc m)
    show ?thesis
      apply (simp add: fps_hypergeo_aux_def mult.assoc)
      apply (auto simp: fps_hypergeo_nth o_def Suc pochhammer_rec prod_list_distrib
simp del: of_nat_Suc)
      done
  qed (auto simp: fps_hypergeo_aux_def)
qed

lemma fps_conv_radius_hypergeo_aux:
  fixes as bs :: "'a :: {banach, real_normed_field} list"
  shows   "fps_conv_radius (fps_hypergeo_aux as bs) =
             (if length as < length bs ∨ set (as@bs) ∩ ℤ≤0 ≠ {} then
∞
              else if length as = length bs then 1
              else 0)" (is "_ = ?r")
  unfolding fps_hypergeo_aux_conv_fps_hypergeo fps_conv_radius_hypergeo
  by (auto simp: one_ereal_def)

lemma hypergeo_F_aux_split_head:
  fixes as bs :: "'a :: {banach, real_normed_field, field_char_0} list"
  assumes "length as < length bs ∨ length as = length bs ∧ norm x <
1"
  shows   "hypergeo_F_aux as bs x =
             1 + prod_list as / prod_list bs * x * hypergeo_F_aux (map
(λa. a+1) as) (map (λb. b+1) bs) x"
proof -
  define c where "c = prod_list as / prod_list bs"
  define as' bs' where "as' = map (λa. a+1) as" "bs' = map (λb. b+1)
bs"
  have 1: "fps_conv_radius (fps_const c * fps_X) ≥ ∞"
    by (rule fps_conv_radius_mult_ge) auto
  have 2: "fps_conv_radius (fps_hypergeo_aux as' bs') ≥ (if length as
```

```
= length bs then 1 else ∞)"
    by (subst fps_conv_radius_hypergeo_aux) (use assms in ‹auto simp:
as'_bs'_def›)

  have "hypergeo_F_aux as bs x = eval_fps (fps_hypergeo_aux as bs) x"
    by (simp add: hypergeo_F_aux_def)
  also have "fps_hypergeo_aux as bs = 1 + fps_const c * fps_X * fps_hypergeo_aux
as' bs'"
    by (subst fps_hypergeo_aux_split_head) (simp_all add: c_def as'_bs'_def)
  also have "eval_fps ... x = 1 + eval_fps (fps_const c * fps_X * fps_hypergeo_aux
as' bs') x"
  proof (subst eval_fps_add)
    have "ereal (norm x) < (if length as = length bs then 1 else ∞)"
      using assms by auto
    also have "... ≤ fps_conv_radius (fps_const c * fps_X * fps_hypergeo_aux
as' bs')"
      by (rule fps_conv_radius_mult_ge) (use 1 2 in ‹auto simp: fps_conv_radius_mult›)
    finally show "ereal (norm x) < fps_conv_radius (fps_const c * fps_X
* fps_hypergeo_aux as' bs')" .
  qed auto
  also have "eval_fps (fps_const c * fps_X * fps_hypergeo_aux as' bs')
x =
             c * x * hypergeo_F_aux as' bs' x"
    unfolding hypergeo_F_aux_def
  proof (subst eval_fps_mult)
    have "ereal (norm x) < (if length as = length bs then 1 else ∞)"
      using assms by auto
    also have "... ≤ fps_conv_radius (fps_hypergeo_aux as' bs')"
      using 2 by auto
    finally show "ereal (norm x) < ..." .
  qed (use 1 assms in ‹auto simp: eval_fps_mult›)
  finally show ?thesis
    by (simp add: c_def as'_bs'_def)
qed
```

1.2 The hypergeometric function

We will now look at the case where the formal series converges to a function. The type constraints we consider here is a field extension of the reals with a complete norm defined on it. In some important lemmas (e.g. continuity), this will be further restricted to Heine–Borel spaces, which effectively means that we will only look at *finite-dimensional* real fields.

It is well-known that the only two such fields, up to isomorphism, are $\mathbb{R}$ and $\mathbb{C}$. These are the fields we care about anyway.

```
definition hypergeo_F :: "'a :: {banach, real_normed_field} list ⇒ 'a
list ⇒ 'a ⇒ 'a" where
  "hypergeo_F as bs = eval_fps (fps_hypergeo as bs 1)"
```

```
lemma hypergeo_F_conv_hypergeo_F_aux:
  "hypergeo_F as bs = hypergeo_F_aux as (1 # bs)"
  by (simp add: hypergeo_F_def hypergeo_F_aux_def fps_hypergeo_conv_fps_hypergeo_aux)

lemma hypergeo_F_0 [simp]: "hypergeo_F as bs 0 = 1"
  by (auto simp: hypergeo_F_def eval_fps_def)

lemma hypergeo_F_Nil_Nil [simp]: "hypergeo_F [] [] = exp"
  by (auto simp: fun_eq_iff hypergeo_F_def)

lemma hypergeo_F_singleton_Nil_real [simp]:
  assumes "|z| < (1::real)"
  shows   "hypergeo_F [a] [] z = (1 - z) powr (-a)"
proof -
  have "((λn. fps_nth (fps_hypergeo [a] [] 1) n * z ^ n) sums (1 - z)
powr (-a))"
    using gen_binomial_real[of "-z" "-a"] assms
    unfolding fps_hypergeo_nth by (simp_all add: gbinomial_pochhammer
power_minus' mult_ac)
  thus ?thesis
    by (simp add: sums_iff hypergeo_F_def eval_fps_def)
qed

lemma hypergeo_F_singleton_Niln_complex [simp]:
  assumes "norm (z :: complex) < 1"
  shows   "hypergeo_F [a] [] z = (1 - z) powr (-a)"
proof -
  have "((λn. fps_nth (fps_hypergeo [a] [] 1) n * z ^ n) sums (1 - z)
powr (-a))"
    using gen_binomial_complex[of "-z" "-a"] assms
    unfolding fps_hypergeo_nth by (simp_all add: gbinomial_pochhammer
power_minus' mult_ac)
  thus ?thesis
    by (simp add: sums_iff hypergeo_F_def eval_fps_def)
qed

lemma hypergeo_F_cancel:
  assumes "a ∉ ℤ≤0"
  shows   "hypergeo_F (a # as) (a # bs) = hypergeo_F as bs"
  using assms by (auto simp: hypergeo_F_def fun_eq_iff fps_hypergeo_cancel)
```

The convergence of ${}_pF_q(\mathbf{a};\mathbf{b};z)$ is uniform not only in z, but also in $\mathbf{a}$ and $\mathbf{b}$.

```
lemma uniform_limit_hypergeo_F_aux:
  fixes as bs :: "('a :: {topological_space} ⇒ 'b :: {banach, real_normed_field,
field_char_0}) list"
  assumes "compact X" and "continuous_on X g" and "list_all (continuous_on
X) (as @ bs)"
  assumes norm: "length as < length bs ∨ length as = length bs ∧ (∀x∈X.
```

```
norm (g x) < 1)"
  assumes "list_all (λb. ∀x∈X. b x ∉ ℤ≤0) bs"
  assumes "⋀x. A x = map (λa. a x) as" and "⋀x. B x = map (λb. b x)
bs"
  shows "uniform_limit X (λN x. ∑n<N. fps_nth (fps_hypergeo_aux (A x)
(B x)) n * (g x) ^ n)
           (λx. hypergeo_F_aux (A x) (B x) (g x)) sequentially"
proof -
  have "bounded (⋃a∈set (as@bs). a ‘ X)"
    by (intro bounded_UN ballI compact_imp_bounded compact_continuous_image)
       (use assms in ‹auto simp: list.pred_set›)
  then obtain C1 where C1: "C1 > 0" "⋀x a. x ∈ X ⟹ a ∈ set (as @
bs) ⟹ norm (a x) ≤ C1"
    using that unfolding bounded_pos by fast

  define f where "f = (λi x. (∏a←as. norm (a x + of_nat i)) / (∏b←bs.
norm (b x + of_nat i)))"
  define f' where "f' = (λi. (∏a←as. real i + C1) / (∏b←bs. real
i - C1))"

  have "bounded (g ‘ X)"
    by (intro compact_imp_bounded compact_continuous_image assms)
  define C3 where "C3 = Sup (insert (1/2) (norm ‘ g ‘ X))"

  have C3: "C3 > 0" "length as = length bs ⟹ C3 < 1" "⋀x. x ∈ X ⟹
norm (g x) ≤ C3"
  proof -
    have bdd: "bdd_above (norm ‘ g ‘ X)"
      unfolding bdd_above_norm by (intro compact_imp_bounded compact_continuous_image
assms)
    have "C3 ≥ 1 / 2"
      unfolding C3_def by (rule cSup_upper) (use bdd in auto)
    thus "C3 > 0"
      by simp

    show "norm (g x) ≤ C3" if "x ∈ X" for x
      unfolding C3_def by (rule cSup_upper) (use that bdd in auto)

    show "C3 < 1" if "length as = length bs"
    proof -
      have "C3 ∈ insert (1/2) (norm ‘ g ‘ X)"
        unfolding C3_def
      proof (rule closed_subset_contains_Sup)
        have "compact (norm ‘ g ‘ X)"
          by (intro compact_continuous_image assms continuous_intros)
        thus "closed (insert (1 / 2) (norm ‘ g ‘ X))"
          by (auto intro!: closed_insert dest: compact_imp_closed)
      qed (use bdd in auto)
      also have "... ⊆ {..<1}"
```

```
      using norm that by auto
    finally show "C3 < 1"
      by simp
  qed
qed

define C4 where "C4 = (if length as = length bs then (C3 + 1) / 2 else
C3 + 1)"
have C4: "C4 > 0" "C4 > C3" "length as = length bs ⟹ C4 < 1"
  using C3(1,2) by (auto simp: C4_def)

have "eventually (λi. real i > C1) at_top"
  by real_asymp
hence "eventually (λi. ∀x∈X. f i x ≤ f' i) at_top"
proof eventually_elim
  case i: (elim i)
  show ?case
  proof
    fix x :: 'a
    assume x: "x ∈ X"
    show "f i x ≤ f' i"
      unfolding f_def f'_def
    proof (intro frac_le prod_list_mono' prod_list_nonneg' prod_list_pos')
      fix a assume a: "a ∈ set as"
      have "norm (of_nat i + a x) ≤ real i + norm (a x)"
        by (rule norm_triangle_mono) auto
      also have "... ≤ real i + C1"
        using C1(2)[of x a] x a by auto
      finally show "norm (a x + of_nat i) ≤ real i + C1"
        by (simp add: add_ac)
    next
      fix b assume b: "b ∈ set bs"
      have "real i - C1 ≤ real i - norm (b x)"
        using C1(2)[of x b] x b by simp
      also have "... ≤ norm (of_nat i + b x)"
        using norm_diff_ineq[of "of_nat i" "b x"] by simp
      finally show "norm (b x + of_nat i) ≥ real i - C1"
        by (simp add: add_ac)
    qed (use i x C1 in auto)
  qed
qed

have "∀F x in sequentially. f' x < 1 / C4"
proof (cases "length as = length bs")
  case False
  with norm have "length as < length bs"
    by auto
  have "f' ∼[at_top] (λi. (∏a←as. real i) / (∏b←bs. real i))"
    unfolding f'_def by (intro asymp_equiv_intros) real_asymp+
```

```
    hence "f' ∼[at_top] (λi. real i ^ length as / real i ^ length bs)"
      by simp
    moreover have "(λi. real i ^ length as / real i ^ length bs) ⟶
0"
      using ‹length as < length bs› by real_asymp
    ultimately have "f' ⟶ 0"
      using tendsto_asymp_equiv_cong by blast
    moreover have "1 / C4 > 0"
      using ‹C4 > 0› by auto
    ultimately show "eventually (λi. f' i < 1 / C4) at_top"
      using order_tendstoD(2) by blast
  next
    case True
    have "f' ∼[at_top] (λi. (∏a←as. real i) / (∏b←bs. real i))"
      unfolding f'_def by (intro asymp_equiv_intros) real_asymp+
    hence "f' ∼[at_top] (λi. real i ^ length as / real i ^ length bs)"
      by simp
    also have "eventually (λi. real i ^ length as / real i ^ length bs
= 1) at_top"
      using eventually_gt_at_top[of 0] by eventually_elim (auto simp:
True)
    finally have "(f' ⟶ 1) at_top"
      by (rule asymp_equivD_const)
    moreover have "1 / C4 > 1"
      using C4 True by simp
    ultimately show "∀F x in sequentially. f' x < 1 / C4"
      by (rule order_tendstoD)
  qed
  hence "eventually (λi. ∀x∈X. f i x < 1 / C4) at_top"
    using ‹eventually (λi. ∀x∈X. f i x ≤ f' i) at_top› by eventually_elim
auto
  then obtain N where N: "⋀x i. i ≥ N ⟹ x ∈ X ⟹ f i x < 1 / C4"
    unfolding eventually_at_top_linorder by blast

  have "bounded ((λx. ∏i∈{..<N}. f i x) ` X)"
    unfolding f_def
    by (intro compact_imp_bounded compact_continuous_image continuous_intros)
       (use assms in ‹fastforce simp: list.pred_set prod_list_zero_iff
add_eq_0_iff2›)+
  then obtain C2 where C2: "C2 > 0" "∀x∈X. (∏i∈{..<N}. f i x) ≤ C2"
    unfolding bounded_pos by fastforce
  have f_nonneg [simp]: "f i x ≥ 0" for i x
    unfolding f_def by (intro divide_nonneg_nonneg prod_list_nonneg')
auto

  show ?thesis
    unfolding hypergeo_F_aux_def eval_fps_def
  proof (rule Weierstrass_m_test_ev)
    show "∀F n in sequentially.
```

```
            ∀x∈X. norm (fps_nth (fps_hypergeo_aux (A x) (B x)) n * g
x ^ n) ≤
                      C2 * C4 ^ N * (C3 / C4) ^ n"
      using eventually_ge_at_top[of N]
    proof eventually_elim
      case (elim n)
      show ?case
      proof
        fix x :: 'a
        assume x: "x ∈ X"
        have "norm (fps_nth (fps_hypergeo_aux (A x) (B x)) n * g x ^ n)
=
                norm (g x) ^ n * ((∏a←as. ∏i=0..<n. norm (a x + of_nat
i)) /
                  (∏b←bs. ∏i=0..<n. norm (b x + of_nat i)))"
          by (simp add: fps_nth_hypergeo_aux norm_mult norm_divide norm_prod_list
o_def
                        assms norm_power pochhammer_prod flip: prod_norm)
        also have "... = norm (g x) ^ n * (∏i=0..<n. f i x)"
          unfolding prod_list_conv_prod_nth' f_def
          by (subst (1 2) prod.swap) (simp_all add: prod_dividef)
        also have "... ≤ C3 ^ n * (∏i=0..<n. f i x)"
          using C3 x by (intro mult_right_mono prod_nonneg f_nonneg ballI
power_mono) auto
        also have "(∏i=0..<n. f i x) = (∏i∈{..<N}∪{N..<n}. f i x)"
          using elim by (intro prod.cong refl) auto
        also have "... = (∏i∈{..<N}. f i x) * (∏i∈{N..<n}. f i x)"
          by (subst prod.union_disjoint) auto
        also have "... ≤ C2 * (∏i∈{N..<n}. 1 / C4)"
          using x C2 less_imp_le[OF N]
          by (intro mult_mono C2 prod_nonneg prod_mono conjI) auto
        also have "(∏i∈{N..<n}. 1 / C4) = (1 / C4) ^ (n - N)"
          by simp
        also have "C3 ^ n * (C2 * (1 / C4) ^ (n - N)) =
                     C2 * C4 ^ N * (C3 / C4) ^ n"
          using ‹C4 > 0› elim by (simp add: field_simps power_diff)
        finally show "norm (fps_nth (fps_hypergeo_aux (A x) (B x)) n *
g x ^ n) ≤
                        C2 * C4 ^ N * (C3 / C4) ^ n"
          by - (use ‹C3 > 0› in ‹simp_all add: mult_left_mono›)
      qed
  qed
  next
    show "summable (λn. C2 * C4 ^ N * (C3 / C4) ^ n)"
      by (intro summable_mult) (use C3 C4 in ‹simp_all add: summable_geometric›)
  qed
qed

lemma sums_hypergeo_F_aux:
```

```
  fixes as bs :: "'a :: {banach, real_normed_field, field_char_0} list"
  assumes "length as < length bs ∨ length as = length bs ∧ norm z <
1"
  shows   "(λn. fps_nth (fps_hypergeo_aux as bs) n * z ^ n) sums hypergeo_F_aux
as bs z"
proof (cases "list_all (λb. b ∉ ℤ≤0) bs")
  case False
  then obtain N where N: "-of_nat N ∈ set bs"
    by (auto simp: list.pred_set elim: nonpos_Ints_cases')
  have "summable (λn. fps_nth (fps_hypergeo_aux as bs) n * z ^ n)"
  proof (rule summable_finite)
    show "finite {..N}"
      by auto
    show "fps_nth (fps_hypergeo_aux as bs) n * z ^ n = 0" if n: "n ∉
{..N}" for n
    proof -
      have "pochhammer (-of_nat N :: 'a) n = 0"
        using N n by (auto simp: pochhammer_of_nat_eq_0_iff)
      thus ?thesis
        using N by (auto simp: fps_nth_hypergeo_aux prod_list_zero_iff)
    qed
  qed
  thus ?thesis
    by (simp add: hypergeo_F_aux_def eval_fps_def sums_iff)
next
  case True
  have "uniform_limit {z} (λN x. ∑n<N. fps_nth (fps_hypergeo_aux as
bs) n * x ^ n)
          (λx. hypergeo_F_aux as bs x) sequentially"
    by (rule uniform_limit_hypergeo_F_aux[where as = "map (λa x. a) as"
and bs = "map (λb x. b) bs"])
       (use assms True in ‹auto simp: o_def list.pred_set›)
  from tendsto_uniform_limitI[OF this, of z] show ?thesis
    by (simp add: sums_def)
qed

lemma of_real_hypergeo_F_aux:
  assumes "length as < length bs ∨ length as = length bs ∧ norm z <
1"
  shows   "(of_real (hypergeo_F_aux as bs z) :: 'a :: {banach, real_normed_field,
field_char_0}) =
           hypergeo_F_aux (map of_real as) (map of_real bs) (of_real z)"
proof -
  have "(λn. fps_nth (fps_hypergeo_aux (map of_real as) (map of_real bs))
n * of_real z ^ n :: 'a) sums
         hypergeo_F_aux (map of_real as) (map of_real bs) (of_real z)"
    by (intro sums_hypergeo_F_aux) (use assms in ‹auto simp: list.pred_map
o_def of_real_in_nonpos_Ints_iff›)
  also have "(λn. fps_nth (fps_hypergeo_aux (map of_real as) (map of_real
```

```
bs)) n * of_real z ^ n :: 'a) =
            (λn. of_real (fps_nth (fps_hypergeo_aux as bs) n * z ^ n)
:: 'a)"
    by (auto simp: fps_nth_hypergeo_aux map_map o_def of_real_prod_list
pochhammer_of_real)
  finally have "(λn. of_real (fps_nth (fps_hypergeo_aux as bs) n * z ^
n) :: 'a) sums
                  hypergeo_F_aux (map of_real as) (map of_real bs) (of_real
z)" .
  moreover have "(λn. of_real (fps_nth (fps_hypergeo_aux as bs) n * z
^ n) :: 'a) sums
            of_real (hypergeo_F_aux as bs z)"
    by (intro sums_of_real sums_hypergeo_F_aux) (use assms in auto)
  ultimately show ?thesis
    by (simp add: sums_iff)
qed

lemma of_real_hypergeo_F:
  assumes "length as ≤ length bs ∨ length as = length bs + 1 ∧ norm
z < 1"
  shows   "(of_real (hypergeo_F as bs z) :: 'a :: {banach, real_normed_field,
field_char_0}) =
           hypergeo_F (map of_real as) (map of_real bs) (of_real z)"
  unfolding hypergeo_F_conv_hypergeo_F_aux by (subst of_real_hypergeo_F_aux)
(use assms in auto)
```

Due to the uniform convergence, ${}_pF_q$ is analytic and continuous in all its parameters.

```
lemma analytic_hypergeo_F_aux:
  assumes "f analytic_on X" and "list_all (λa. a analytic_on X) (as @
bs)"
  assumes norm: "length as < length bs ∨ length as = length bs ∧ (∀x∈X.
norm (f x) < 1)"
  assumes "list_all (λb. ∀x∈X. b x ∉ ℤ≤0) bs"
  assumes "⋀x. A x = map (λa. a x) as" and  "⋀x. B x = map (λb. b x)
bs"
  shows "(λx. hypergeo_F_aux (A x) (B x) (f x)) analytic_on X"
proof (subst analytic_on_analytic_at, safe)
  fix x assume x: "x ∈ X"
  have "∀a∈insert f (set (as @ bs)). ∃A. open A ∧ X ⊆ A ∧ a holomorphic_on
A"
    using assms(1,2) unfolding list.pred_set analytic_on_holomorphic by
fast
  then obtain Y where Y: "∀a∈insert f (set (as @ bs)). open (Y a) ∧
X ⊆ Y a ∧ a holomorphic_on Y a"
    by metis

  define X1 where "X1 = (⋂a∈insert f (set (as @ bs)). Y a)"
  have X1: "open X1" "X ⊆ X1"
```

```
    unfolding X1_def using Y by blast+
  have holo: "⋀a. a ∈ insert f (set (as @ bs)) ⟹ a holomorphic_on
X1"
    using Y unfolding X1_def by fast

  define X2 where "X2 = (if length as = length bs then X1 ∩ f -‘ ball
0 1 else X1)"
  have f_cont: "continuous_on A f" if "A ⊆ X1" for A
    by (intro holomorphic_on_imp_continuous_on[OF holomorphic_on_subset[of
_ X1]] holo that) auto
  have X2: "open X2" "X2 ⊆ X1" "X ⊆ X2" "⋀x. length as = length bs ⟹
x ∈ X2 ⟹ norm (f x) < 1"
    using X1 norm by (auto simp: X2_def intro!: continuous_open_preimage
f_cont)
  have holo: "a holomorphic_on X2" if "a ∈ insert f (set (as @ bs))" for
a
    using Y X2 holo[of a] holomorphic_on_subset[of a X1 X2] that by blast

  have "x ∈ X2 ∩ (⋂b∈set bs. b -‘ (-ℤ≤0))"
    using x X1 X2 assms(4) by (auto simp: list.pred_set)
  moreover {
    have "open (X2 ∩ (⋂b∈set bs. X2 ∩ b -‘ (-ℤ≤0)))"
      using X2 X1
      by (intro open_Int open_INT ballI continuous_open_preimage
              holomorphic_on_imp_continuous_on holomorphic_on_subset[OF
holo]) auto
    also have "... = X2 ∩ (⋂b∈set bs. b -‘ (-ℤ≤0))"
      by blast
    finally have "open (X2 ∩ (⋂b∈set bs. b -‘ (-ℤ≤0)))" .
  }
  ultimately obtain r where r: "r > 0" "cball x r ⊆ X2 ∩ (⋂b∈set bs.
b -‘ (-ℤ≤0))"
    using open_contains_cball_eq by meson

  show "(λx. hypergeo_F_aux (A x) (B x) (f x)) analytic_on {x}"
  proof (rule holomorphic_uniform_limit)
    show "uniform_limit (cball x r) (λN x. ∑n<N. fps_nth (fps_hypergeo_aux
(A x) (B x)) n * (f x) ^ n)
            (λx. hypergeo_F_aux (A x) (B x) (f x)) sequentially"
    proof (rule uniform_limit_hypergeo_F_aux[where as = as and bs = bs])
      show "continuous_on (cball x r) f" "list_all (continuous_on (cball
x r)) (as @ bs)"
        using holo r unfolding list.pred_set
        by (auto intro!: holomorphic_on_imp_continuous_on holomorphic_on_subset[OF
holo])
    next
      show "length as < length bs ∨ length as = length bs ∧ (∀x∈cball
x r. cmod (f x) < 1)"
      proof (cases "length as = length bs")
```

```
      case False
      thus ?thesis using norm
        by auto
    next
      case True
      have "∀x∈cball x r. cmod (f x) < 1"
      proof safe
        fix z assume "z ∈ cball x r"
        also have "... ⊆ X2"
          using r X2 by auto
        also have "... ⊆ f -‘ ball 0 1"
          using X2 True by auto
        finally show "norm (f z) < 1"
          by simp
      qed
      thus ?thesis using True by auto
    qed
  qed (use assms r X2 in ‹auto simp: list.pred_set›)
next
  show "∀F n in sequentially.
         continuous_on (cball x r)
          (λx. ∑n<n. fps_nth (fps_hypergeo_aux (A x) (B x)) n * f x
^ n) ∧
         (λx. ∑n<n. fps_nth (fps_hypergeo_aux (A x) (B x)) n * f x
^ n) holomorphic_on ball x r"
    unfolding fps_nth_hypergeo_aux assms map_map o_def using r
    by (intro always_eventually allI conjI holomorphic_on_imp_continuous_on
              holomorphic_intros holomorphic_on_subset[OF holo])
       (force simp: prod_list_zero_iff dest: pochhammer_eq_0_imp_nonpos_Int)+
next
  assume "(λx. hypergeo_F_aux (A x) (B x) (f x)) holomorphic_on ball
x r"
  thus "(λx. hypergeo_F_aux (A x) (B x) (f x)) analytic_on {x}"
    using ‹r > 0› analytic_at_ball by blast
qed auto
qed

lemma continuous_on_hypergeo_F_aux:
  fixes X :: "'a :: heine_borel set"
  fixes f :: "'a ⇒ 'b :: {banach, real_normed_field}"
  assumes "continuous_on X f" and "list_all (λa. continuous_on X a) (as
@ bs)"
  assumes "length as < length bs ∨ length as = length bs ∧ (∀x∈X. norm
(f x) < 1)"
  assumes "list_all (λb. ∀x∈X. b x ∉ ℤ≤0) bs"
  assumes "⋀x. A x = map (λa. a x) as" and  "⋀x. B x = map (λb. b x)
bs"
  shows   "continuous_on X (λx. hypergeo_F_aux (A x) (B x) (f x))"
proof -
```

```
  have "continuous (at x within X) (λx. hypergeo_F_aux (A x) (B x) (f
x))"
    if x: "x ∈ X" for x
  proof (rule continuous_within_sequentiallyI)
    fix h assume h: "h ⟶ x" "∀n. h n ∈ X"
    define X' where "X' = insert x (range h)"
    have "compact X'"
      unfolding X'_def by (rule compact_sequence_with_limit) fact
    have "X' ⊆ X"
      using x h by (auto simp: X'_def)

    have 1: "continuous_on X' f"
      by (rule continuous_on_subset[OF assms(1)]) (use ‹X' ⊆ X› in auto)
    have 2: "list_all (continuous_on X') (as @ bs)"
      unfolding list.pred_set
      by (intro ballI continuous_on_subset[of X _ X'])
         (use assms(2) h x in ‹auto simp: X'_def list.pred_set›)
    have "continuous_on X' (λx. hypergeo_F_aux (A x) (B x) (f x))"
    proof (rule uniform_limit_theorem)
      show "uniform_limit X' (λN x. ∑n<N. fps_nth (fps_hypergeo_aux
(A x) (B x)) n * (f x) ^ n)
              (λx. hypergeo_F_aux (A x) (B x) (f x)) sequentially"
      proof (rule uniform_limit_hypergeo_F_aux[where as = as and bs =
bs])
        show "compact X'"
          by fact
      qed (use assms 1 2 h x in ‹auto simp: list.pred_set X'_def›)
    next
      show "∀F n in sequentially. continuous_on X'
              (λx. ∑n<n. fps_hypergeo_aux (A x) (B x) $ n * f x ^ n)"
        unfolding fps_nth_hypergeo_aux assms map_map o_def using 1 2 assms(4)
h x
        by (intro always_eventually allI continuous_intros)
           (auto simp: list.pred_set prod_list_zero_iff X'_def dest!:
pochhammer_eq_0_imp_nonpos_Int)
    qed auto
    hence "((λx. hypergeo_F_aux (A x) (B x) (f x)) ∘ h)
             ⟶ hypergeo_F_aux (A x) (B x) (f x)"
      by (rule continuous_onD_sequentially) (use h in ‹auto simp: X'_def›)
    thus "(λn. hypergeo_F_aux (A (h n)) (B (h n)) (f (h n)))
             ⟶ hypergeo_F_aux (A x) (B x) (f x)"
      by (simp add: o_def)
  qed
  thus ?thesis
    by (auto simp: continuous_on_eq_continuous_within)
qed
```

We only evaluate the derivative in the variable x, not in the parameters, since there is no closed-form expression for that in general.

```
theorem has_field_derivative_hypergeo_F:
  fixes as bs
  defines "as' ≡ map (λn. n+1) as" and "bs' ≡ map (λn. n+1) bs"
  defines "C ≡ prod_list as / prod_list bs"
  assumes norm: "length as ≤ length bs ∨ length as = Suc (length bs)
∧ norm x < 1"
  assumes bs: "set bs ∩ ℤ≤0 = {}"
  shows   "(hypergeo_F as bs has_field_derivative
              (C * hypergeo_F as' bs' x)) (at x within A)"
proof -
  define F where "F = fps_hypergeo as bs 1"
  define R where "R = (if length as ≤ length bs then ∞ else 1 :: ereal)"
  have x: "norm x < R"
    using norm by (auto simp: R_def)

  have "(eval_fps F has_field_derivative eval_fps (fps_deriv F) x) (at
x within A)"
  proof (rule has_field_derivative_eval_fps)
    have "ereal (norm x) < R"
      by fact
    also have "... ≤ fps_conv_radius F"
      unfolding F_def fps_conv_radius_hypergeo using norm by (auto simp:
R_def one_ereal_def)
    finally show "ereal (norm x) < fps_conv_radius F" .
  qed
  also have "eval_fps F = hypergeo_F as bs"
    by (simp add: F_def hypergeo_F_def)
  also have "fps_deriv F = fps_const C * fps_hypergeo as' bs' 1"
    using fps_deriv_hypergeo1[OF bs, of as 1]
    by (auto simp: F_def as'_def bs'_def C_def field_simps)
  also have "eval_fps (fps_const C * fps_hypergeo as' bs' 1) x = C * hypergeo_F
as' bs' x"
  proof (subst eval_fps_mult)
    have "norm x < R"
      by fact
    also have "R ≤ fps_conv_radius (fps_hypergeo as' bs' 1)"
      by (subst fps_conv_radius_hypergeo)
         (use norm in ‹auto simp: as'_def bs'_def R_def one_ereal_def›)
    finally show "norm x < ..." .
  qed (auto simp: hypergeo_F_def)
  finally show ?thesis .
qed

lemma has_field_derivative_hypergeo_F' [derivative_intros]:
  fixes as bs
  defines "as' ≡ map (λn. n+1) as" and "bs' ≡ map (λn. n+1) bs"
  defines "C ≡ prod_list as / prod_list bs"
  assumes f: "(f has_field_derivative f') (at x within A)"
  assumes "length as ≤ length bs ∨ length as = Suc (length bs) ∧ norm
```

```
(f x) < 1"
  assumes "set bs ∩ ℤ≤0 = {}"
  shows   "((λx. hypergeo_F as bs (f x)) has_field_derivative
             (C * hypergeo_F as' bs' (f x) * f')) (at x within A)"
  unfolding as'_def bs'_def
  using DERIV_chain[OF has_field_derivative_hypergeo_F f, of as bs] assms

  by (simp add: o_def)

corollary deriv_hypergeo_F:
  fixes as bs
  defines "as' ≡ map (λn. n+1) as" and "bs' ≡ map (λn. n+1) bs"
  defines "C ≡ prod_list as / prod_list bs"
  assumes "length as ≤ length bs ∨ length as = Suc (length bs) ∧ norm
x < 1"
  assumes "set bs ∩ ℤ≤0 = {}"
  shows   "deriv (hypergeo_F as bs) x = C * hypergeo_F as' bs' x"
  by (rule DERIV_imp_deriv) (use assms in ‹auto intro!: derivative_eq_intros›)

corollary higher_deriv_hypergeo_F:
  fixes as bs m
  defines "as' ≡ (λm. map (λn. n + of_nat m) as)" and "bs' ≡ (λm. map
(λn. n + of_nat m) bs)"
  defines "C ≡ (λm. (∏a←as. pochhammer a m) / (∏b←bs. pochhammer
b m))"
  assumes "length as ≤ length bs ∨ length as = Suc (length bs) ∧ norm
x < 1"
  assumes bs: "set bs ∩ ℤ≤0 = {}"
  shows   "(deriv ^^ m) (hypergeo_F as bs) x = C m * hypergeo_F (as' m)
(bs' m) x"
  using assms(4)
proof (induction m arbitrary: x)
  case (Suc m)
  define A where "A = (if length as = Suc (length bs) then ball 0 1 else
UNIV :: 'a set)"
  have "open A"
    by (auto simp: A_def)
  have bs': "set (bs' m) ∩ ℤ≤0 = {}"
  proof -
    have "b + of_nat m ∉ ℤ≤0" if "b ∈ set bs" for b
    proof
      assume "b + of_nat m ∈ ℤ≤0"
      then obtain k where "b + of_nat m = of_int k" "k ≤ 0"
        by (elim nonpos_Ints_cases)
      hence "b = of_int (k - int m)"
        by (simp add: algebra_simps)
      moreover have "k - int m ≤ 0"
        using ‹k ≤ 0› by simp
      ultimately have "b ∈ ℤ≤0"
```

```
      using of_int_in_nonpos_Ints_iff by blast
    thus False
      using that bs by blast
  qed
  thus ?thesis
    unfolding bs'_def by auto
qed

have "(deriv ^^ Suc m) (hypergeo_F as bs) x = deriv ((deriv ^^ m) (hypergeo_F
as bs)) x"
  by simp
also have "... = deriv (λx. C m * hypergeo_F (as' m) (bs' m) x) x"
proof (rule deriv_cong_ev)
  have "x ∈ A"
    using Suc.prems by (auto simp: A_def)
  hence "eventually (λy. y ∈ A) (nhds x)"
    by (intro eventually_nhds_in_open ‹open A›)
  thus "∀F y in nhds x. (deriv ^^ m) (hypergeo_F as bs) y = C m * hypergeo_F
(as' m) (bs' m) y"
  proof eventually_elim
    case (elim y)
    show "(deriv ^^ m) (hypergeo_F as bs) y = C m * hypergeo_F (as'
m) (bs' m) y"
      by (rule Suc.IH) (use Suc.prems elim in ‹auto simp: A_def›)
  qed
qed auto
also have "... = C (Suc m) * hypergeo_F (as' (Suc m)) (bs' (Suc m))
x"
proof (rule DERIV_imp_deriv)
  define C' where "C' = (∏a←as. a + of_nat m) / (∏b←bs. b + of_nat
m)"
  have "((λx. C m * hypergeo_F (as' m) (bs' m) x) has_field_derivative
          C m * C' * hypergeo_F (as' (Suc m)) (bs' (Suc m)) x) (at x)"
    by (rule derivative_eq_intros refl)+
       (use Suc.prems bs' in ‹auto simp: as'_def bs'_def o_def C'_def
pochhammer_rec add_ac›)
  also have "C m * C' = C (Suc m)"
    unfolding C_def C'_def pochhammer_Suc prod_list_distrib by (simp
add: field_simps)
  finally show "((λx. C m * hypergeo_F (as' m) (bs' m) x) has_field_derivative
                  C (Suc m) * hypergeo_F (as' (Suc m)) (bs' (Suc m))
x) (at x)" .
qed
finally show ?case .
qed (auto simp: C_def as'_def bs'_def)


theorem uniform_limit_hypergeo_F:
  fixes as bs :: "('a :: {topological_space} ⇒ 'b :: {banach, real_normed_field,
```

```
field_char_0}) list"
  assumes "compact X" and "continuous_on X g" and "list_all (continuous_on
X) (as @ bs)"
  assumes "length as ≤ length bs ∨ length as = length bs + 1 ∧ (∀x∈X.
norm (g x) < 1)"
  assumes "list_all (λb. ∀x∈X. b x ∉ ℤ≤0) bs"
  assumes "⋀x. A x = map (λa. a x) as" and "⋀x. B x = map (λb. b x)
bs"
  shows "uniform_limit X (λN x. ∑n<N. fps_nth (fps_hypergeo (A x) (B
x) 1) n * (g x) ^ n)
           (λx. hypergeo_F (A x) (B x) (g x)) sequentially"
  using uniform_limit_hypergeo_F_aux[of X g as "(λ_. 1) # bs" A "λx. 1
# B x"] assms
  by (simp_all add: hypergeo_F_conv_hypergeo_F_aux fps_hypergeo_conv_fps_hypergeo_aux
less_Suc_eq_le)

corollary sums_hypergeo_F:
  fixes as bs :: "'a :: {banach, real_normed_field, field_char_0} list"
  assumes "list_all (λb. b ∉ ℤ≤0) bs" and "length as ≤ length bs ∨
length as = length bs + 1 ∧ norm z < 1"
  shows   "(λn. fps_nth (fps_hypergeo as bs 1) n * z ^ n) sums hypergeo_F
as bs z"
  using sums_hypergeo_F_aux[where as = as and bs = "1 # bs" and z =
z] assms
  by (simp_all add: hypergeo_F_conv_hypergeo_F_aux fps_hypergeo_conv_fps_hypergeo_aux
less_Suc_eq_le)

corollary sums_hypergeo_F':
  fixes as bs :: "'a :: {banach, real_normed_field, field_char_0} list"
  assumes "list_all (λb. b ∉ ℤ≤0) bs" and "length as ≤ length bs ∨
length as = length bs + 1 ∧ norm (c * z) < 1"
  shows   "(λn. fps_nth (fps_hypergeo as bs c) n * z ^ n) sums hypergeo_F
as bs (c * z)"
proof -
  have "(λn. fps_nth (fps_hypergeo as bs 1) n * (c * z) ^ n) sums hypergeo_F
as bs (c * z)"
    by (rule sums_hypergeo_F) (use assms in auto)
  also have "(λn. fps_nth (fps_hypergeo as bs 1) n * (c * z) ^ n) =
             (λn. fps_nth (fps_hypergeo as bs c) n * z ^ n)"
    by (simp add: fps_hypergeo_nth power_mult_distrib mult_ac)
  finally show ?thesis .
qed
```

The function is analytic in its variable and all of its parameters simultaneously (unsurprisingly).

```
corollary analytic_hypergeo_F:
  assumes "f analytic_on X" and "list_all (λa. a analytic_on X) (as @
bs)"
  assumes "length as ≤ length bs ∨ length as = length bs + 1 ∧ (∀x∈X.
```

```
norm (f x) < 1)"
  assumes "list_all (λb. ∀x∈X. b x ∉ ℤ≤0) bs"
  assumes "⋀x. A x = map (λa. a x) as" and "⋀x. B x = map (λb. b x)
bs"
  shows "(λx. hypergeo_F (A x) (B x) (f x)) analytic_on X"
proof -
  have "(λx. hypergeo_F_aux (A x) (1 # B x) (f x)) analytic_on X"
    by (rule analytic_hypergeo_F_aux[where as = as and bs = "(λ_. 1)
# bs"])
       (use assms in auto)
  thus ?thesis
    by (simp add: hypergeo_F_conv_hypergeo_F_aux fps_hypergeo_conv_fps_hypergeo_aux)
qed

corollary analytic_hypergeo_F_simple [analytic_intros]:
  assumes "f analytic_on X"
  assumes "length as ≤ length bs ∨ length as = length bs + 1 ∧ (∀x∈X.
norm (f x) < 1)"
  assumes "set bs ∩ ℤ≤0 = {}"
  shows "(λx. hypergeo_F as bs (f x)) analytic_on X"
  by (rule analytic_hypergeo_F[where as = "map (λa x. a) as" and bs
= "map (λb x. b) bs"])
     (use assms in ‹auto simp: o_def list.pred_set›)

corollary holomorphic_hypergeo_F_simple [holomorphic_intros]:
  assumes "f holomorphic_on X"
  assumes "length as ≤ length bs ∨ length as = length bs + 1 ∧ (∀x∈X.
norm (f x) < 1)"
  assumes "set bs ∩ ℤ≤0 = {}"
  shows "(λx. hypergeo_F as bs (f x)) holomorphic_on X"
proof -
  define A where "A = (if length as ≤ length bs then UNIV else ball 0
1 :: complex set)"
  have *: "hypergeo_F as bs holomorphic_on A"
    by (rule analytic_imp_holomorphic, rule analytic_hypergeo_F_simple)
       (use assms in ‹auto simp: A_def›)
  have "(hypergeo_F as bs ∘ f) holomorphic_on X"
    using assms(1) * by (rule holomorphic_on_compose_gen) (use assms(2)
in ‹auto simp: A_def›)
  thus ?thesis
    by (simp add: o_def)
qed
```

The continuity theorem looks a bit awkward. The natural way to express it would be as:

```
continuous_on {(as,bs,z). set bs ∩ ℤ≤0 = {} ∧ (length as ≤ length bs
∨ length as = length bs + 1 ∧ norm x < 1} (λ(as,bs,x). hypergeo_F as
bs x)
```

However, this is not possible since there is no topological space defined on

the list type, and creating one just for this would be a lot of work.

In practice, one will typically consider lists of fixed length and derive a corresponding version of the above sketched lemma using tuples instead of lists (see example for Kummer below).

```
corollary continuous_on_hypergeo_F:
  fixes X :: "'a :: heine_borel set"
  fixes f :: "'a ⇒ 'b :: {banach, real_normed_field}"
  assumes "continuous_on X f" and "list_all (λa. continuous_on X a) (as
@ bs)"
  assumes "length as ≤ length bs ∨ length as = length bs + 1 ∧ (∀x∈X.
norm (f x) < 1)"
  assumes "list_all (λb. ∀x∈X. b x ∉ ℤ≤0) bs"
  assumes "⋀x. A x = map (λa. a x) as" and  "⋀x. B x = map (λb. b x)
bs"
  shows   "continuous_on X (λx. hypergeo_F (A x) (B x) (f x))"
proof -
  have "continuous_on X (λx. hypergeo_F_aux (A x) (1 # B x) (f x))"
    by (rule continuous_on_hypergeo_F_aux[where as = as and bs = "(λ_.
1) # bs"])
       (use assms in auto)
  thus ?thesis
    by (simp add: hypergeo_F_conv_hypergeo_F_aux fps_hypergeo_conv_fps_hypergeo_aux)
qed

corollary continuous_on_hypergeo_F_simple [continuous_intros]:
  assumes "continuous_on X f"
  assumes "length as ≤ length bs ∨ length as = length bs + 1 ∧ (∀x∈X.
norm (f x) < 1)"
  assumes "set bs ∩ ℤ≤0 = {}"
  shows   "continuous_on X (λx. hypergeo_F as bs (f x))"
proof -
  define A where "A = (if length as ≤ length bs then UNIV else ball 0
1 :: 'b set)"
  have "continuous_on A (hypergeo_F as bs)"
    by (rule DERIV_continuous_on[OF has_field_derivative_hypergeo_F])
       (use assms in ‹auto simp: A_def›)
  thus ?thesis
    by (rule continuous_on_compose2) (use assms in ‹auto simp: A_def›)
qed

corollary tendsto_hypergeo_F_simple [tendsto_intros]:
  assumes "(f ⟶ z) F"
  assumes "length as ≤ length bs ∨ length as = length bs + 1 ∧ norm
z < 1"
  assumes "set bs ∩ ℤ≤0 = {}"
  shows   "((λx. hypergeo_F as bs (f x)) ⟶ hypergeo_F as bs z) F"
proof -
  define A where "A = (if length as ≤ length bs then UNIV else ball 0
```

```
1 :: 'b set)"
  have "continuous_on A (hypergeo_F as bs)"
    unfolding A_def by (intro continuous_intros) (use assms in auto)
  hence "isCont (hypergeo_F as bs) z"
    using assms(2) by (simp add: A_def continuous_on_eq_continuous_at
split: if_splits)
  thus ?thesis
    using assms(1) by (metis isCont_tendsto_compose)
qed

corollary continuous_hypergeo_F_simple [continuous_intros]:
  assumes "continuous (at z within A) f"
  assumes "length as ≤ length bs ∨ length as = length bs + 1 ∧ norm
(f z) < 1"
  assumes "set bs ∩ ℤ≤0 = {}"
  shows   "continuous (at z within A) (λx. hypergeo_F as bs (f x))"
proof -
  define A where "A = (if length as ≤ length bs then UNIV else ball 0
1 :: 'b set)"
  have "continuous_on A (hypergeo_F as bs)"
    unfolding A_def by (intro continuous_intros) (use assms in auto)
  hence "isCont (hypergeo_F as bs) (f z)"
    using assms(2) by (simp add: A_def continuous_on_eq_continuous_at
split: if_splits)
  thus ?thesis
    using assms(1) by (metis continuous_within_compose3)
qed
```

As an example, the proof that Kummer's hypergeometric function is continuous in all variables.

```
lemma continuous_on_hypergeo_F_kummer:
  fixes A :: "('a :: {real_normed_field, banach, heine_borel} × 'a × 'a)
set"
  assumes "A ⊆ UNIV × (-ℤ≤0) × UNIV"
  shows   "continuous_on A (λ(a,b,z). hypergeo_F [a] [b] z)"
  unfolding case_prod_unfold
  by (rule continuous_on_hypergeo_F[where as = "[λ(a,b,z). a]" and bs
= "[λ(a,b,z). b]"])
     (use assms in ‹auto simp: case_prod_unfold intro!: continuous_intros›)

lemma continuous_on_hypergeo_F_kummer':
  fixes f :: "'a :: topological_space ⇒ 'b :: {real_normed_field, banach,
heine_borel}"
  assumes "continuous_on A f" "continuous_on A g" "continuous_on A h"
  assumes "⋀x. x ∈ A ⟹ g x ∉ ℤ≤0"
  shows   "continuous_on A (λx. hypergeo_F [f x] [g x] (h x))"
proof -
  have "continuous_on A ((λ(a,b,z). hypergeo_F [a] [b] z) ∘ (λx. (f x,
g x, h x)))"
```

```
    by (rule continuous_on_compose[OF _ continuous_on_hypergeo_F_kummer])
       (use assms in ‹auto intro!: continuous_intros›)
  thus ?thesis
    by (simp add: o_def)
qed


lemma has_fps_expansion_hypergeo_F [fps_expansion_intros]:
  assumes "length as ≤ Suc (length bs)" "list_all (λb. b ∉ ℤ≤0) bs"
  shows   "(λx. hypergeo_F as bs (c * x)) has_fps_expansion fps_hypergeo
as bs c"
  unfolding has_fps_expansion_def
proof
  have "1 / ereal (norm c) > 0"
    by (simp add: one_ereal_def)
  thus "fps_conv_radius (fps_hypergeo as bs c) > 0"
    unfolding hypergeo_F_def by (subst fps_conv_radius_hypergeo) (use
assms in auto)
  have "eventually (λz::'a. z ∈ (if c = 0 then UNIV else ball 0 (1 /
norm c))) (nhds 0)"
    by (intro eventually_nhds_in_open) auto
  thus "∀F z in nhds 0. eval_fps (fps_hypergeo as bs c) z = hypergeo_F
as bs (c * z)"
  proof eventually_elim
    case (elim z)
    have "(λn. fps_nth (fps_hypergeo as bs c) n * z ^ n) sums hypergeo_F
as bs (c * z)"
      by (rule sums_hypergeo_F')
         (use assms elim in ‹auto split: if_splits simp: norm_mult field_simps›)
    thus ?case
      by (simp add: eval_fps_def sums_iff)
  qed
qed

lemma has_fps_expansion_hypergeo_F' [fps_expansion_intros]:
  assumes "length as ≤ Suc (length bs)" "list_all (λb. b ∉ ℤ≤0) bs"
  shows   "(λx. hypergeo_F as bs x) has_fps_expansion fps_hypergeo as
bs 1"
  using has_fps_expansion_hypergeo_F[OF assms, of 1] by simp
```

1.3 Representing trigonometric and hyperbolic functions

The functions sin, cos, sinh, cosh have simple representations in terms of ${}_0F_1$.

```
lemma pochhammer_three_halves:
  "pochhammer (3 / 2 :: 'a :: field_char_0) n =
     (2 * of_nat n + 1) * fact (2 * n) / (2 ^ (2 * n) * fact n)"
proof -
  have "pochhammer (1 / 2 + 1 :: 'a) n = 2 * pochhammer (1 / 2) (Suc n)"
```

```
    using pochhammer_rec[of "1/2 :: 'a" n] by (simp add: field_simps)
  also have "... = (2 * of_nat n + 1) * fact (2 * n) / (2 ^ (2 * n) *
fact n)"
    by (simp add: pochhammer_Suc pochhammer_one_half)
  finally show ?thesis by simp
qed

lemma cos_conv_hypergeo_F:
  fixes z :: "'a :: {real_normed_field, field_char_0, banach}"
  shows "cos z = hypergeo_F [] [1/2] (-z^2/4)"
proof -
  have "((λn. fps_nth (fps_hypergeo [] [1/2] 1) n * (-z^2/4) ^ n) sums
cos z)"
    using cos_series[of z]
    unfolding fps_hypergeo_nth power_mult prod_list.Cons list.map pochhammer_one_half
    by (simp add: power_minus' mult_ac power_divide scaleR_conv_of_real)
  thus ?thesis
    by (simp add: sums_iff hypergeo_F_def eval_fps_def)
qed

lemma sin_conv_hypergeo_F:
  fixes z :: "'a :: {real_normed_field, field_char_0, banach}"
  shows   "sin z = z * hypergeo_F [] [3/2] (-z^2/4)"
proof -
  have "((λn. z * (fps_nth (fps_hypergeo [] [3/2] 1) n * (-z^2/4) ^ n))
sums sin z)"
    using sin_series[of z]
    unfolding fps_hypergeo_nth power_mult prod_list.Cons list.map pochhammer_three_halves
              power_add power2_eq_square
    by (simp add: power_minus' mult_ac power_divide add_ac scaleR_conv_of_real)
  moreover have "(λn. z * (fps_nth (fps_hypergeo [] [3/2] 1) n * (-z^2/4)
^ n)) sums
                 (z * hypergeo_F [] [3/2] (-z^2/4))"
    by (intro sums_mult sums_hypergeo_F) auto
  ultimately show ?thesis
    using sums_unique2 by blast
qed

lemma cosh_conv_hypergeo_F:
  fixes z :: "'a :: {real_normed_field, field_char_0, banach}"
  shows "cosh z = hypergeo_F [] [1/2] (z^2/4)"
proof -
  have "((λn. fps_nth (fps_hypergeo [] [1/2] 1) n * (z^2/4) ^ n) sums cosh
z)"
    using cosh_series[of z]
    unfolding fps_hypergeo_nth power_mult prod_list.Cons list.map pochhammer_one_half
    by (simp add: mult_ac power_divide scaleR_conv_of_real)
  thus ?thesis
    by (simp add: sums_iff hypergeo_F_def eval_fps_def)
```

qed

lemma *sinh_conv_hypergeo_F:*
  fixes *z :: "'a :: {real_normed_field, field_char_0, banach}"*
  shows *"sinh z = z * hypergeo_F [] [3/2] (z^2/4)"*
proof -
  have *"((λn. z * (fps_nth (fps_hypergeo [] [3/2] 1) n * (z^2/4) ^ n))*
sums sinh z)"
    using *sinh_series[of z]*
    unfolding *fps_hypergeo_nth power_mult prod_list.Cons list.map pochhammer_three_halves*
          power_add power2_eq_square
    by *(simp add: power_minus' mult_ac power_divide add_ac scaleR_conv_of_real)*
  moreover have *"(λn. z * (fps_nth (fps_hypergeo [] [3/2] 1) n * (z^2/4)*
^ n)) sums
               *(z * hypergeo_F [] [3/2] (z^2/4))"*
    by *(intro sums_mult sums_hypergeo_F) auto*
  ultimately show *?thesis*
    using *sums_unique2* **by** *blast*
qed

Similarly, arctan and artanh have representations in terms of ${}_2F_1$.

lemma *arctan_conv_hypergeo_F_real:*
  assumes *"|z| < 1"*
  shows *"arctan z = z * hypergeo_F [1, 1/2] [3/2] (-z^2)"*
proof -
  have *"arctan z = (∑n. z * (fps_nth (fps_hypergeo [1, 1/2] [3/2] 1)*
*n * (-z^2) ^ n))"*
    using *arctan_series[of z] assms*
    unfolding *fps_hypergeo_nth power_mult prod_list.Cons list.map pochhammer_three_halves*
          pochhammer_one_half power_add power2_eq_square
    by *(simp add: power_minus' mult_ac power_divide add_ac scaleR_conv_of_real*
           power_mult_distrib flip: pochhammer_fact)
  moreover have *"(λn. z * (fps_nth (fps_hypergeo [1, 1/2] [3/2] 1) n*
** (-z^2) ^ n)) sums*
               *(z * hypergeo_F [1, 1/2] [3/2] (-z^2))"*
    by *(intro sums_mult sums_hypergeo_F) (use assms* **in** *‹auto simp: abs_square_less_1›)*
  ultimately show *?thesis*
    by *(simp add: sums_iff)*
qed

lemma *arctan_conv_hypergeo_F_complex:*
  assumes *"norm z < 1"*
  shows *"Arctan z = z * hypergeo_F [1, 1/2] [3/2] (-z^2)"*
proof -
  have *"(λn. z * (fps_nth (fps_hypergeo [1, 1/2] [3/2] 1) n * (-z^2) ^*
n)) sums Arctan z"
    using *Arctan_series(2)[of z] assms*
    unfolding *fps_hypergeo_nth power_mult prod_list.Cons list.map pochhammer_three_halves*
          pochhammer_one_half power_add power2_eq_square

```
    by (simp add: power_minus' mult_ac power_divide add_ac scaleR_conv_of_real
                  power_mult_distrib flip: pochhammer_fact)
  moreover have "(λn. z * (fps_nth (fps_hypergeo [1, 1/2] [3/2] 1) n
* (-z^2) ^ n)) sums
                 (z * hypergeo_F [1, 1/2] [3/2] (-z^2))"
    by (intro sums_mult sums_hypergeo_F) (use assms in ‹auto simp: norm_power
abs_square_less_1›)
  ultimately show ?thesis
    using sums_unique2 by blast
qed

lemma artanh_conv_hypergeo_F_complex:
  assumes "norm (z :: complex) < 1"
  shows   "artanh z = z * hypergeo_F [1, 1/2] [3/2] (z^2)"
  unfolding artanh_conv_Arctan using arctan_conv_hypergeo_F_complex[of
"i * z"] assms
  by (simp add: norm_mult algebra_simps)

lemma artanh_conv_hypergeo_F_real:
  assumes "|z::real| < 1"
  shows   "artanh z = z * hypergeo_F [1, 1/2] [3/2] (z^2)"
proof -
  have "complex_of_real (artanh z) = artanh (of_real z)"
    using assms by (simp add: artanh_def Ln_Reals_eq)
  also have "... = of_real z * hypergeo_F [1, 1/2] [3/2] (of_real (z^2))"
    by (subst artanh_conv_hypergeo_F_complex) (use assms in simp_all)
  also have "... = of_real z * of_real (hypergeo_F [1, 1/2] [3/2] (z^2))"
    by (subst of_real_hypergeo_F) (use assms in ‹simp_all add: norm_power
abs_square_less_1›)
  finally show ?thesis
    by (simp add: complex_eq_iff)
qed
```

1.4 Regularised version

The "normal" hypergeometric function is undefined if any of the parameters b_i are non-positive integers. This can be fixed by considering the following *regularised* version, which is well-defined for any combination of parameters $(a_i)_{1\leq i\leq m}$ and $(b_i)_{1\leq i\leq n}$:

```
definition reg_fps_hypergeo_aux :: "'a :: Gamma list ⇒ 'a list ⇒ 'a fps"
where
  "reg_fps_hypergeo_aux as bs =
     Abs_fps (λn. (∏a←as. pochhammer a n) * (∏b←bs. rGamma (b + of_nat
n)))"

definition reg_hypergeo_F :: "'a :: Gamma list ⇒ 'a list ⇒ 'a ⇒ 'a"
where
  "reg_hypergeo_F as bs = eval_fps (reg_fps_hypergeo_aux as (1 # bs))"
```

```
lemma reg_hypergeo_F_0: "reg_hypergeo_F as bs 0 = (∏ b←bs. rGamma b)"
  by (simp add: reg_hypergeo_F_def reg_fps_hypergeo_aux_def eval_fps_at_0)

lemma reg_fps_hypergeo_aux_conv_fps_hypergeo_aux:
  assumes "set bs ∩ ℤ_{≤0} = {}"
  shows   "reg_fps_hypergeo_aux as bs = fps_const (∏ b←bs. rGamma b)
* fps_hypergeo_aux as bs"
proof (rule fps_ext)
  fix n :: nat
  have "fps_nth (reg_fps_hypergeo_aux as bs) n =
          (∏ a←as. pochhammer a n) * (∏ b←bs. rGamma (b + of_nat n))"
    by (simp add: reg_fps_hypergeo_aux_def)
  also have "(∏ b←bs. rGamma (b + of_nat n)) = (∏ b←bs. rGamma b / pochhammer
b n)"
  proof (intro arg_cong[of _ _ prod_list] map_cong refl)
    fix b assume b: "b ∈ set bs"
    hence "b ∉ ℤ_{≤0}"
      using assms by blast
    thus "rGamma (b + of_nat n) = rGamma b / pochhammer b n"
      using pochhammer_rGamma[of b n] by (auto simp: divide_simps pochhammer_eq_0_iff)
  qed
  also have "(∏ a←as. pochhammer a n) * ... =
               (∏ b←bs. rGamma b) * ((∏ a←as. pochhammer a n) / (∏ b←bs.
pochhammer b n))"
    by (simp add: prod_list_divide)
  also have "... = fps_nth (fps_const (∏ b←bs. rGamma b) * fps_hypergeo_aux
as bs) n"
    by (simp add: fps_hypergeo_aux_def)
  finally show "fps_nth (reg_fps_hypergeo_aux as bs) n = ..." .
qed

lemma reg_hypergeo_F_conv_hypergeo_F:
  assumes "set bs ∩ ℤ_{≤0} = {}" "length as ≤ length bs ∨ length as =
length bs + 1 ∧ norm z < 1"
  shows   "reg_hypergeo_F as bs z = (∏ b←bs. rGamma b) * hypergeo_F
as bs z"
proof -
  define c where "c = (∏ b←bs. rGamma b)"
  have "reg_hypergeo_F as bs z = eval_fps (fps_const c * fps_hypergeo
as bs 1) z"
    unfolding reg_hypergeo_F_def c_def fps_hypergeo_conv_fps_hypergeo_aux
    by (subst reg_fps_hypergeo_aux_conv_fps_hypergeo_aux) (use assms(1)
in auto)
  also have "... = c * hypergeo_F as bs z"
  proof (subst eval_fps_mult)
    show "ereal (norm z) < fps_conv_radius (fps_hypergeo as bs 1)"
      using assms(2) by (subst fps_conv_radius_hypergeo) (auto simp: one_ereal_def)
  qed (auto simp: hypergeo_F_def)
```

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

lemma fps_deriv_reg_hypergeo_aux1:
  fixes as bs :: "'a :: Gamma list"
  defines "as' ≡ map (λa. a + 1) as"
  defines "bs' ≡ map (λb. b + 1) bs"
  shows "fps_deriv (reg_fps_hypergeo_aux as (1#bs)) =
           fps_const (prod_list as) * reg_fps_hypergeo_aux as' (1#bs')"
proof -
  have *: "rGamma (of_nat n + 1 :: 'a) = (of_nat n + 1) * rGamma (2 +
of_nat n)" for n
    using rGamma_plus1[of "of_nat n + 1 :: 'a"] by simp
  show ?thesis (is "?lhs = ?rhs")
  proof (rule fps_ext)
    fix n :: nat
    have "fps_nth ?lhs n =
            prod_list as * (∏a←as'. pochhammer a n) *
            ((of_nat n + 1) * rGamma (of_nat n + 2) * (∏b←bs'. rGamma
(b + of_nat n)))"
      by (auto simp: add_ac reg_fps_hypergeo_aux_def pochhammer_rec prod_list_distrib
as'_def bs'_def o_def)
    also have "(of_nat n + 1) * rGamma (of_nat n + 2) = rGamma (of_nat
n + 1 :: 'a)"
      using rGamma_plus1[of "of_nat n + 1 :: 'a"] by (simp add: add_ac)
    also have "... * (∏b←bs'. rGamma (b + of_nat n)) = (∏b←1#bs'.
rGamma (b + of_nat n))"
      by (simp add: add_ac)
    also have "prod_list as * (∏a←as'. pochhammer a n) * (∏b←1#bs'.
rGamma (b + of_nat n)) =
               fps_nth ?rhs n"
      by (simp add: reg_fps_hypergeo_aux_def)
    finally show "fps_nth ?lhs n = fps_nth ?rhs n" .
  qed
qed

lemma fps_conv_radius_reg_hypergeo_aux:
  fixes as bs :: "'a :: Gamma list"
  shows   "fps_conv_radius (reg_fps_hypergeo_aux as bs) =
             (if length as < length bs ∨ set as ∩ ℤ≤0 ≠ {} then ∞
              else if length as = length bs then 1
              else 0)" (is "_ = ?r")
proof (cases "set as ∩ ℤ≤0 ≠ {}")
  case True
  then obtain n where n: "-of_nat n ∈ set as"
    by (auto elim!: nonpos_Ints_cases')
  have "eventually (λn. fps_nth (reg_fps_hypergeo_aux as bs) n = 0) at_top"
    using eventually_gt_at_top[of n]
```

```
  proof eventually_elim
    case (elim k)
    have "pochhammer (-of_nat n :: 'a) k = 0"
      using elim by (auto simp: pochhammer_eq_0_iff)
    hence "(∏a←as. pochhammer a k) = 0"
      using n by (auto simp: prod_list_zero_iff)
    thus ?case
      by (simp add: reg_fps_hypergeo_aux_def)
  qed
  hence "fps_conv_radius (reg_fps_hypergeo_aux as bs) = fps_conv_radius
(0 :: 'a fps)"
    unfolding fps_conv_radius_def by (intro conv_radius_cong') auto
  also have "... = ∞"
    by simp
  finally show ?thesis
    using True by auto
next
  case *: False
  define c where "c = fps_nth (reg_fps_hypergeo_aux as bs)"

  have nz1: "(∏a←as. norm (pochhammer a n)) ≠ 0" for n
    using * pochhammer_eq_0_imp_nonpos_Int by (force simp: prod_list_zero_iff)

  have ev_b: "eventually (λn. ∀b∈set bs. b + of_nat n ∉ ℤ≤0) at_top"
  proof (intro eventually_ball_finite ballI)
    fix b assume b: "b ∈ set bs"
    have "eventually (λn. real n - norm b > 0) at_top"
      by real_asymp
    thus "eventually (λn. b + of_nat n ∉ ℤ≤0) at_top"
    proof eventually_elim
      case (elim n)
      show ?case
      proof
        assume "b + of_nat n ∈ ℤ≤0"
        then obtain k where k: "b + of_nat n = of_int k" "k ≤ 0"
          by (elim nonpos_Ints_cases)
        have b_eq: "b = of_int (k - int n)"
          unfolding of_int_diff k(1) [symmetric] by simp
        have "0 < real n - norm b"
          by fact
        also have "real n - norm b = of_int k"
          by (subst b_eq, subst norm_of_int) (use ‹k ≤ 0› in auto)
        also have "... ≤ 0"
          using k(2) by simp
        finally show False
          by simp
      qed
    qed
  qed
  qed auto
```

```
  have "eventually (λn. ereal (norm (c n) / norm (c (Suc n))) =
                        (∏b←bs. norm (b + of_nat n)) / (∏a←as. norm
(a + of_nat n))) at_top"
    using eventually_all_ge_at_top[OF ev_b]
  proof eventually_elim
    case (elim n)
    have "ereal (norm (c n) / norm (c (Suc n))) = ereal (norm
          ((∏b←bs. rGamma (b + of_nat n) / rGamma (b + of_nat n + 1))
/
          ((∏a←as. pochhammer a (Suc n) / pochhammer a n))))"
      by (simp add: c_def reg_fps_hypergeo_aux_def prod_list_divide norm_divide
norm_mult
                    add_ac divide_simps)
    also have "(∏a←as. pochhammer a (Suc n) / pochhammer a n) =
               (∏a←as. (a + of_nat n))"
      using nz1 by (intro arg_cong[of _ _ prod_list] map_cong)
                   (auto simp: pochhammer_Suc prod_list_zero_iff image_iff)
    also have "(∏b←bs. rGamma (b + of_nat n) / rGamma (b + of_nat n
+ 1)) =
               (∏b←bs. b + of_nat n)"
      by (intro arg_cong[of _ _ prod_list] map_cong refl, subst rGamma_plus1
[symmetric])
         (use spec[OF elim, of "n+1"] in ‹auto simp: rGamma_eq_zero_iff
add_ac›)
    finally show ?case
      by (simp add: norm_divide norm_prod_list o_def)
  qed

  also have "(λn. (∏b←bs. norm (b + of_nat n)) / (∏a←as. norm (a +
of_nat n))) ⟶ ?r"
  proof -
    let ?m = "int (length bs) - int (length as)"
    have [asymp_equiv_intros]: "(λx. norm (b + of_nat x)) ∼[sequentially]
real" for b :: 'a
    proof -
      have "(λx. norm (b + of_real x)) ∘ real ∼[sequentially] (λx. x)
∘ real"
        by (rule asymp_equiv_compose norm_plus_of_real_asymp_equiv filterlim_real_sequentia
      thus ?thesis
        by (simp add: o_def)
    qed
    let ?lhs = "(λn. (∏b←bs. norm (b + of_nat n)) / (∏a←as. norm (a
+ of_nat n)))"
    have "?lhs ∼[at_top] (λn. (∏b←bs. real n) / (∏a←as. real n))"
      by (intro asymp_equiv_intros norm_plus_of_real_asymp_equiv)
    also have "... ∼[at_top] (λn. real n powi ?m)"
    proof (rule asymp_equiv_refl_ev)
      have "eventually (λn::nat. n > 0) at_top"
```

```
      by (rule eventually_gt_at_top)
    thus "eventually (λn. (∏b←bs. real n) / (∏a←as. real n) = real
n powi ?m) at_top"
      by eventually_elim (auto simp: power_int_diff power_int_add)
  qed
  finally have "(λn. ereal (?lhs n)) ⟶ ?r ⟷ (λn. ereal (real
n powi ?m)) ⟶ ?r"
    by (rule tendsto_ereal_asymp_equiv_transfer)
  also have "(λn. ereal (real n powi ?m)) ⟶ ?r"
  proof (cases "?m ≤ 0")
    case True
    from True have "(λn. 1 / real n ^ (nat (-?m))) ⟶ (if ?m =
0 then 1 else 0)"
      by (cases "?m = 0") (real_asymp simp: field_simps)+
    also have "(λn. 1 / real n ^ (nat (-?m))) = (λn. real n powi ?m)"
      using ‹?m ≤ 0› unfolding power_int_def by (auto simp: fun_eq_iff
field_simps)
    finally have "(λn. ereal (real n powi ?m)) ⟶ ereal (if ?m =
0 then 1 else 0)"
      by (intro tendsto_intros)
    also have "ereal (if ?m = 0 then 1 else 0) = ?r"
      using * True by (auto simp: one_ereal_def)
    finally show ?thesis .
  next
    case False
    hence "filterlim (λn. real n ^ (nat ?m)) at_top at_top"
      by real_asymp
    also have "(λn. real n ^ nat (?m)) = (λn. real n powi ?m)"
      using False unfolding power_int_def by (auto simp: fun_eq_iff
field_simps)
    finally have "(λn. ereal (real n powi ?m)) ⟶ ∞"
      by (simp add: tendsto_PInfty_eq_at_top)
    also have "∞ = ?r"
      using * False by (auto simp: one_ereal_def)
    finally show ?thesis .
  qed
  finally show ?thesis .
qed

finally have lim: "(λn. ereal (norm (c n) / norm (c (Suc n)))) ⟶
?r" .

have nz: "∀F n in sequentially. c n ≠ 0"
  using ev_b
proof eventually_elim
  case (elim n)
  thus ?case using nz1[of n]
    by (auto simp: c_def reg_fps_hypergeo_aux_def prod_list_zero_iff
rGamma_eq_zero_iff)
```

```
  qed
  show ?thesis using conv_radius_ratio_limit_ereal[OF nz lim]
    by (simp add: fps_conv_radius_def c_def)
qed

lemma uniform_limit_reg_hypergeo_F_aux:
  fixes as bs :: "('a :: {topological_space} ⇒ 'b :: Gamma) list"
  assumes "compact X" and "continuous_on X g" and "list_all (continuous_on
X) (as @ bs)"
  assumes norm: "length as < length bs ∨ length as = length bs ∧ (∀x∈X.
norm (g x) < 1)"
  assumes "⋀x. x ∈ X ⟹ A x = map (λa. a x) as" and "⋀x. x ∈ X ⟹
B x = map (λb. b x) bs"
  shows "uniform_limit X (λN x. ∑n<N. fps_nth (reg_fps_hypergeo_aux
(A x) (B x)) n * (g x) ^ n)
          (λx. eval_fps (reg_fps_hypergeo_aux (A x) (B x)) (g x)) sequentially"
proof -
  have "bounded (⋃a∈set (as@bs). a ` X)"
    by (intro bounded_UN ballI compact_imp_bounded compact_continuous_image)
       (use assms in ‹auto simp: list.pred_set›)
  then obtain C1 where C1: "C1 > 0" "⋀x a. x ∈ X ⟹ a ∈ set (as @
bs) ⟹ norm (a x) ≤ C1"
    using that unfolding bounded_pos by fast

  define f where "f = (λi x. (∏a←as. norm (a x + of_nat i)) / (∏b←bs.
norm (b x + of_nat i)))"
  define f' where "f' = (λi. (∏a←as. real i + C1) / (∏b←bs. real
i - C1))"
  have f_nonneg [simp]: "f i x ≥ 0" for i x
    unfolding f_def by (intro divide_nonneg_nonneg prod_list_nonneg')
auto

  have "bounded (g ` X)"
    by (intro compact_imp_bounded compact_continuous_image assms)
  define C3 where "C3 = Sup (insert (1/2) (norm ` g ` X))"

  have C3: "C3 > 0" "length as = length bs ⟹ C3 < 1" "⋀x. x ∈ X ⟹
norm (g x) ≤ C3"
  proof -
    have bdd: "bdd_above (norm ` g ` X)"
      unfolding bdd_above_norm by (intro compact_imp_bounded compact_continuous_image
assms)
    have "C3 ≥ 1 / 2"
      unfolding C3_def by (rule cSup_upper) (use bdd in auto)
    thus "C3 > 0"
      by simp

    show "norm (g x) ≤ C3" if "x ∈ X" for x
      unfolding C3_def by (rule cSup_upper) (use that bdd in auto)
```

```
    show "C3 < 1" if "length as = length bs"
    proof -
      have "C3 ∈ insert (1/2) (norm ‘ g ‘ X)"
        unfolding C3_def
      proof (rule closed_subset_contains_Sup)
        have "compact (norm ‘ g ‘ X)"
          by (intro compact_continuous_image assms continuous_intros)
        thus "closed (insert (1 / 2) (norm ‘ g ‘ X))"
          by (auto intro!: closed_insert dest: compact_imp_closed)
      qed (use bdd in auto)
      also have "... ⊆ {..<1}"
        using norm that by auto
      finally show "C3 < 1"
        by simp
    qed
  qed

  define C4 where "C4 = (if length as = length bs then (C3 + 1) / 2 else
C3 + 1)"
  have C4: "C4 > 0" "C4 > C3" "length as = length bs ⟹ C4 < 1"
    using C3(1,2) by (auto simp: C4_def)

  have "eventually (λi. real i > C1) at_top"
    by real_asymp
  hence "eventually (λi. ∀x∈X. f i x ≤ f' i) at_top"
  proof eventually_elim
    case i: (elim i)
    show ?case
    proof
      fix x :: 'a
      assume x: "x ∈ X"
      show "f i x ≤ f' i"
        unfolding f_def f'_def
      proof (intro frac_le prod_list_mono' prod_list_nonneg' prod_list_pos')
        fix a assume a: "a ∈ set as"
        have "norm (of_nat i + a x) ≤ real i + norm (a x)"
          by (rule norm_triangle_mono) auto
        also have "... ≤ real i + C1"
          using C1(2)[of x a] x a by auto
        finally show "norm (a x + of_nat i) ≤ real i + C1"
          by (simp add: add_ac)
      next
        fix b assume b: "b ∈ set bs"
        have "real i - C1 ≤ real i - norm (b x)"
          using C1(2)[of x b] x b by simp
        also have "... ≤ norm (of_nat i + b x)"
          using norm_diff_ineq[of "of_nat i" "b x"] by simp
        finally show "norm (b x + of_nat i) ≥ real i - C1"
```

```
        by (simp add: add_ac)
      qed (use i x C1 in auto)
    qed
  qed

  have "∀⇩F x in sequentially. f' x < 1 / C4"
  proof (cases "length as = length bs")
    case False
    with norm have "length as < length bs"
      by auto
    have "f' ∼[at_top] (λi. (∏a←as. real i) / (∏b←bs. real i))"
      unfolding f'_def by (intro asymp_equiv_intros) real_asymp+
    hence "f' ∼[at_top] (λi. real i ^ length as / real i ^ length bs)"
      by simp
    moreover have "(λi. real i ^ length as / real i ^ length bs) ⟶
0"
      using ‹length as < length bs› by real_asymp
    ultimately have "f' ⟶ 0"
      using tendsto_asymp_equiv_cong by blast
    moreover have "1 / C4 > 0"
      using ‹C4 > 0› by auto
    ultimately show "eventually (λi. f' i < 1 / C4) at_top"
      using order_tendstoD(2) by blast
  next
    case True
    have "f' ∼[at_top] (λi. (∏a←as. real i) / (∏b←bs. real i))"
      unfolding f'_def by (intro asymp_equiv_intros) real_asymp+
    hence "f' ∼[at_top] (λi. real i ^ length as / real i ^ length bs)"
      by simp
    also have "eventually (λi. real i ^ length as / real i ^ length bs
= 1) at_top"
      using eventually_gt_at_top[of 0] by eventually_elim (auto simp:
True)
    finally have "(f' ⟶ 1) at_top"
      by (rule asymp_equivD_const)
    moreover have "1 / C4 > 1"
      using C4 True by simp
    ultimately show "∀⇩F x in sequentially. f' x < 1 / C4"
      by (rule order_tendstoD)
  qed
  hence "eventually (λi. ∀x∈X. f i x < 1 / C4) at_top"
    using ‹eventually (λi. ∀x∈X. f i x ≤ f' i) at_top› by eventually_elim
auto
  moreover have "eventually (λn. ∀b∈set bs. ∀x∈X. b x + of_nat n ≠
0) at_top"
  proof (intro eventually_ball_finite[of "set bs"] ballI)
    fix b assume b: "b ∈ set bs"
    have "compact (b ‘ X)"
      using ‹compact X› ‹list_all (continuous_on X) (as @ bs)› b
```

```
    by (intro compact_continuous_image) (auto simp: list.pred_set)
  hence "bounded (b ‘ X)"
    by (rule compact_imp_bounded)
  then obtain B where B: "B > 0" "⋀x. x ∈ X ⟹ norm (b x) ≤ B"
    unfolding bounded_pos by blast
  have "eventually (λn. real n - B > 0) at_top"
    by real_asymp
  thus "eventually (λn. ∀x∈X. b x + of_nat n ≠ 0) at_top"
  proof eventually_elim
    case (elim n)
    show "∀x∈X. b x + of_nat n ≠ 0"
    proof
      fix x assume x: "x ∈ X"
      have "0 < norm (of_nat n :: 'b) - norm (b x)"
        using elim B(2)[OF x] by simp
      also have "... ≤ norm (b x + of_nat n)"
        by norm
      finally show "b x + of_nat n ≠ 0"
        by auto
    qed
  qed
qed auto

ultimately have "eventually (λi. ∀x∈X. f i x < 1 / C4 ∧ (∀b∈set bs.
b x + of_nat i ≠ 0)) at_top"
  by eventually_elim auto
then obtain N where N: "⋀x i. i ≥ N ⟹ x ∈ X ⟹ f i x < 1 / C4"
                       "⋀i x b. i ≥ N ⟹ b ∈ set bs ⟹ x ∈ X ⟹
b x + of_nat i ≠ 0"
  unfolding eventually_at_top_linorder by metis

obtain C2 where C2: "C2 > 0"
  "norm ((∏i<N. ∏a←as. a x + of_nat i) * (∏b←bs. rGamma (b x +
of_nat N))) ≤ C2"
  if "x ∈ X" for x
proof -
  have "compact ((λx. (∏i<N. ∏a←as. a x + of_nat i) * (∏b←bs.
rGamma (b x + of_nat N))) ‘ X)"
    using ‹compact X› ‹list_all (continuous_on X) (as @ bs)›
    by (intro compact_continuous_image continuous_intros) (auto simp:
list.pred_set)
  hence "bounded ((λx. (∏i<N. ∏a←as. a x + of_nat i) * (∏b←bs.
rGamma (b x + of_nat N))) ‘ X)"
    by (rule compact_imp_bounded)
  thus ?thesis
    unfolding bounded_pos using that by blast
qed

show ?thesis
```

```
    unfolding hypergeo_F_aux_def eval_fps_def
  proof (rule Weierstrass_m_test_ev)
    show "∀⇩F n in sequentially.
            ∀x∈X. norm (fps_nth (reg_fps_hypergeo_aux (A x) (B x)) n
* g x ^ n) ≤
                    C2 * C4 ^ N * (C3 / C4) ^ n"
      using eventually_ge_at_top[of N]
    proof eventually_elim
      case (elim n)
      show ?case
      proof
        fix x :: 'a
        assume x: "x ∈ X"
        have "norm (fps_nth (reg_fps_hypergeo_aux (A x) (B x)) n * g x
^ n) =
                norm (g x) ^ n * ((∏a←as. ∏i=0..<n. norm (a x + of_nat
i)) *
                  (∏b←bs. norm (rGamma (b x + of_nat n))))" using x
          by (simp add: norm_mult norm_divide norm_prod_list o_def assms
norm_power
                        pochhammer_prod reg_fps_hypergeo_aux_def flip:
prod_norm)
        also have "(∏b←bs. norm (rGamma (b x + of_nat n))) =
                   (∏b←bs. norm (rGamma (b x + of_nat N) / pochhammer
(b x + of_nat N) (n - N)))"
        proof (intro arg_cong[of _ _ prod_list] map_cong, goal_cases)
          case (2 b)
          have "pochhammer (b x + of_nat N) (n - N) ≠ 0"
            unfolding pochhammer_eq_0_iff using N(2)[OF _ 2 x] elim
            by (metis ab_left_minus add.inverse_inverse add.left_commute
le_add2 of_nat_add)
          hence "rGamma (b x + of_nat n) = rGamma (b x + of_nat N) / pochhammer
(b x + of_nat N) (n - N)"
            using elim by (subst (2) pochhammer_rGamma[of _ "n - N"])
auto
          thus ?case
            by simp
        qed auto
        also have "... = norm (∏b←bs. rGamma (b x + of_nat N)) /
                         (∏b←bs. norm (pochhammer (b x + of_nat N) (n
- N)))"
          unfolding norm_prod_list norm_divide prod_list_divide map_map
o_def ..
        also have "(∏a←as. ∏i=0..<n. norm (a x + of_nat i)) =
                   (∏i=0..<n. ∏a←as. norm (a x + of_nat i))"
          unfolding prod_list_conv_prod_nth' by (subst prod.swap) simp
        also have "... = (∏i∈{..<N}∪{N..<n}. ∏a←as. norm (a x + of_nat
i))"
          by (rule prod.cong) (use elim in auto)
```

```
    also have "... = (∏i<N. ∏a←as. norm (a x + of_nat i)) *
                     (∏i=N..<n. ∏a←as. norm (a x + of_nat i))"
      by (subst prod.union_disjoint) auto
    also have "(∏b←bs. norm (pochhammer (b x + of_nat N) (n - N)))
=
                 (∏b←bs. ∏i=0..<n-N. norm (b x + of_nat (N+i)))"
      by (simp add: pochhammer_prod add_ac flip: prod_norm)
    also have "... = (∏i=0..<n-N. ∏b←bs. norm (b x + of_nat (N+i)))"
      unfolding prod_list_conv_prod_nth' by (subst prod.swap) simp
    also have "... = (∏i=N..<n. ∏b←bs. norm (b x + of_nat i))"
      by (rule prod.reindex_bij_witness[of _ "λi. i - N" "λi. i +
N"]) (auto simp: add_ac)
    finally have "norm (fps_nth (reg_fps_hypergeo_aux (A x) (B x))
n * g x ^ n) =
                    norm (g x) ^ n * (∏i=N..<n. f i x) *
                    norm ((∏i<N. ∏a←as. a x + of_nat i) * (∏b←bs.
rGamma (b x + of_nat N)))"
      by (simp add: f_def prod_dividef norm_prod_list o_def norm_mult
flip: prod_norm)

    also have "... ≤ C3 ^ n * (∏i=N..<n. 1 / C4) * C2"
      by (intro mult_mono C3 less_imp_le[OF N(1)] C2 conjI power_mono
prod_mono
                  mult_nonneg_nonneg zero_le_power prod_nonneg)
         (use elim x ‹C3 > 0› ‹C4 > 0› in auto)
    also have "... = C2 * C4 ^ N * (C3 / C4) ^ n"
      using ‹C4 > 0› elim by (simp add: field_simps power_diff)
    finally show "norm (fps_nth (reg_fps_hypergeo_aux (A x) (B x))
n * g x ^ n) ≤
                    C2 * C4 ^ N * (C3 / C4) ^ n" .
  qed
 qed
 next
  show "summable (λn. C2 * C4 ^ N * (C3 / C4) ^ n)"
   by (intro summable_mult) (use C3 C4 in ‹simp_all add: summable_geometric›)
 qed
qed

lemma sums_reg_hypergeo_F_aux:
 fixes as bs :: "'a :: Gamma list"
 assumes "length as < length bs ∨ length as = length bs ∧ norm z <
1"
 shows   "(λn. fps_nth (reg_fps_hypergeo_aux as bs) n * z ^ n) sums

             eval_fps (reg_fps_hypergeo_aux as bs) z"
proof -
 have "uniform_limit {z} (λN x. ∑n<N. fps_nth (reg_fps_hypergeo_aux
as bs) n * x ^ n)
        (λx. eval_fps (reg_fps_hypergeo_aux as bs) x) sequentially"
```

```
    by (rule uniform_limit_reg_hypergeo_F_aux[where as = "map (λa x.
a) as" and bs = "map (λb x. b) bs"])
       (use assms in ‹auto simp: o_def list.pred_set›)
  from tendsto_uniform_limitI[OF this, of z] show ?thesis
    by (simp add: sums_def)
qed

lemma complex_of_real_reg_hypergeo_F_aux:
  assumes "length as < length bs ∨ length as = length bs ∧ norm z <
1"
  shows   "(of_real (eval_fps (reg_fps_hypergeo_aux as bs) z) :: complex)
=
           eval_fps (reg_fps_hypergeo_aux (map of_real as) (map of_real
bs)) (of_real z)"
proof -
  have "(λn. fps_nth (reg_fps_hypergeo_aux (map of_real as) (map of_real
bs)) n * of_real z ^ n :: complex) sums
          eval_fps (reg_fps_hypergeo_aux (map of_real as) (map of_real
bs)) (of_real z)"
    by (intro sums_reg_hypergeo_F_aux)
       (use assms in ‹auto simp: list.pred_map o_def of_real_in_nonpos_Ints_iff›)
  also have "(λn. fps_nth (reg_fps_hypergeo_aux (map of_real as) (map
of_real bs)) n * of_real z ^ n) =
             (λn. of_real (fps_nth (reg_fps_hypergeo_aux as bs) n * z
^ n) :: complex)"
    by (auto simp: reg_fps_hypergeo_aux_def o_def of_real_prod_list pochhammer_of_real

             simp flip: rGamma_complex_of_real)
  finally have "(λn. of_real (fps_nth (reg_fps_hypergeo_aux as bs) n *
z ^ n) :: complex) sums
                  eval_fps (reg_fps_hypergeo_aux (map of_real as) (map
of_real bs)) (of_real z)" .
  moreover have "(λn. of_real (fps_nth (reg_fps_hypergeo_aux as bs) n
* z ^ n) :: complex) sums
             of_real (eval_fps (reg_fps_hypergeo_aux as bs) z)"
    by (intro sums_of_real sums_reg_hypergeo_F_aux) (use assms in auto)
  ultimately show ?thesis
    by (simp add: sums_iff)
qed

lemma complex_of_real_reg_hypergeo_F:
  assumes "length as ≤ length bs ∨ length as = length bs + 1 ∧ norm
z < 1"
  shows   "(complex_of_real (reg_hypergeo_F as bs z) :: complex) =
           reg_hypergeo_F (map of_real as) (map of_real bs) (of_real z)"
  unfolding reg_hypergeo_F_def
  using complex_of_real_reg_hypergeo_F_aux[of as "1 # bs" z] assms by
auto
```

```
lemma analytic_reg_hypergeo_F_aux:
  assumes "f analytic_on X" and "list_all (λa. a analytic_on X) (as @
bs)"
  assumes norm: "length as < length bs ∨ length as = length bs ∧ (∀x∈X.
norm (f x) < 1)"
  assumes "⋀x. A x = map (λa. a x) as" and  "⋀x. B x = map (λb. b x)
bs"
  shows "(λx. eval_fps (reg_fps_hypergeo_aux (A x) (B x)) (f x)) analytic_on
X"
proof (subst analytic_on_analytic_at, safe)
  fix x assume x: "x ∈ X"
  have "∀a∈insert f (set (as @ bs)). ∃A. open A ∧ X ⊆ A ∧ a holomorphic_on
A"
    using assms(1,2) unfolding list.pred_set analytic_on_holomorphic by
fast
  then obtain Y where Y: "∀a∈insert f (set (as @ bs)). open (Y a) ∧
X ⊆ Y a ∧ a holomorphic_on Y a"
    by metis

  define X1 where "X1 = (⋂a∈insert f (set (as @ bs)). Y a)"
  have X1: "open X1" "X ⊆ X1"
    unfolding X1_def using Y by blast+
  have holo: "⋀a. a ∈ insert f (set (as @ bs)) ⟹ a holomorphic_on
X1"
    using Y unfolding X1_def by fast

  define X2 where "X2 = (if length as = length bs then X1 ∩ f -‘ ball
0 1 else X1)"
  have f_cont: "continuous_on A f" if "A ⊆ X1" for A
    by (intro holomorphic_on_imp_continuous_on[OF holomorphic_on_subset[of
_ X1]] holo that) auto
  have X2: "open X2" "X2 ⊆ X1" "X ⊆ X2" "⋀x. length as = length bs ⟹
x ∈ X2 ⟹ norm (f x) < 1"
    using X1 norm by (auto simp: X2_def intro!: continuous_open_preimage
f_cont)
  have holo: "a holomorphic_on X2" if "a ∈ insert f (set (as @ bs))" for
a
    using Y X2 holo[of a] holomorphic_on_subset[of a X1 X2] that by blast

  have "x ∈ X2"
    using x X2 by auto
  then obtain r where r: "r > 0" "cball x r ⊆ X2"
    using ‹open X2› open_contains_cball_eq by meson

  show "(λx. eval_fps (reg_fps_hypergeo_aux (A x) (B x)) (f x)) analytic_on
{x}"
  proof (rule holomorphic_uniform_limit)
    show "uniform_limit (cball x r) (λN x. ∑n<N. fps_nth (reg_fps_hypergeo_aux
```

```
(A x) (B x)) n * (f x) ^ n)
            (λx. eval_fps (reg_fps_hypergeo_aux (A x) (B x)) (f x)) sequentially"
    proof (rule uniform_limit_reg_hypergeo_F_aux[where as = as and bs
= bs])
      show "continuous_on (cball x r) f" "list_all (continuous_on (cball
x r)) (as @ bs)"
        using holo r unfolding list.pred_set
        by (auto intro!: holomorphic_on_imp_continuous_on holomorphic_on_subset[OF
holo])
    next
      show "length as < length bs ∨ length as = length bs ∧ (∀x∈cball
x r. cmod (f x) < 1)"
      proof (cases "length as = length bs")
        case False
        thus ?thesis using norm
          by auto
      next
        case True
        have "∀x∈cball x r. cmod (f x) < 1"
        proof safe
          fix z assume "z ∈ cball x r"
          also have "... ⊆ X2"
            using r X2 by auto
          also have "... ⊆ f -‘ ball 0 1"
            using X2 True by auto
          finally show "norm (f z) < 1"
            by simp
        qed
        thus ?thesis using True by auto
      qed
    qed (use assms r X2 in ‹auto simp: list.pred_set›)
  next
    show "∀F n in sequentially.
           continuous_on (cball x r)
            (λx. ∑n<n. fps_nth (reg_fps_hypergeo_aux (A x) (B x)) n *
f x ^ n) ∧
           (λx. ∑n<n. fps_nth (reg_fps_hypergeo_aux (A x) (B x)) n *
f x ^ n) holomorphic_on ball x r"
      unfolding reg_fps_hypergeo_aux_def fps_nth_Abs_fps assms map_map
o_def using r
      by (intro always_eventually allI conjI holomorphic_on_imp_continuous_on
                holomorphic_intros holomorphic_on_subset[OF holo]) auto
  next
    assume "(λx. eval_fps (reg_fps_hypergeo_aux (A x) (B x)) (f x)) holomorphic_on
ball x r"
    thus "(λx. eval_fps (reg_fps_hypergeo_aux (A x) (B x)) (f x)) analytic_on
{x}"
      using ‹r > 0› analytic_at_ball by blast
  qed auto
```

qed

lemma *analytic_reg_hypergeo_F:*
assumes *"f analytic_on X"* **and** *"list_all (λa. a analytic_on X) (as @ bs)"*
assumes *norm: "length as ≤ length bs ∨ length as = length bs + 1 ∧ (∀x∈X. norm (f x) < 1)"*
assumes *"⋀x. A x = map (λa. a x) as"* **and** *"⋀x. B x = map (λb. b x) bs"*
shows *"(λx. reg_hypergeo_F (A x) (B x) (f x)) analytic_on X"*
unfolding *reg_hypergeo_F_def*
by *(rule analytic_reg_hypergeo_F_aux[***where** *as = as* **and** *bs = "(λ_. 1) # bs"])*
(use assms **in** *auto)*

corollary *analytic_reg_hypergeo_F_simple [analytic_intros]:*
assumes *"f analytic_on X"*
assumes *"length as ≤ length bs ∨ length as = length bs + 1 ∧ (∀x∈X. norm (f x) < 1)"*
shows *"(λx. reg_hypergeo_F as bs (f x)) analytic_on X"*
by *(rule analytic_reg_hypergeo_F[***where** *as = "map (λa x. a) as"* **and** *bs = "map (λb x. b) bs"])*
(use assms **in** *‹auto simp: o_def list.pred_set›)*

corollary *holomorphic_reg_hypergeo_F_simple [holomorphic_intros]:*
assumes *"f holomorphic_on X"*
assumes *"length as ≤ length bs ∨ length as = length bs + 1 ∧ (∀x∈X. norm (f x) < 1)"*
shows *"(λx. reg_hypergeo_F as bs (f x)) holomorphic_on X"*
proof -
define *A* **where** *"A = (if length as ≤ length bs then UNIV else ball 0 1 :: complex set)"*
have *∗: "reg_hypergeo_F as bs holomorphic_on A"*
by *(rule analytic_imp_holomorphic, rule analytic_reg_hypergeo_F_simple)*
(use assms **in** *‹auto simp: A_def›)*
have *"(reg_hypergeo_F as bs ∘ f) holomorphic_on X"*
using *assms(1) ∗* **by** *(rule holomorphic_on_compose_gen) (use assms(2)* **in** *‹auto simp: A_def›)*
thus *?thesis*
by *(simp add: o_def)*
qed

lemma *continuous_on_reg_hypergeo_F_aux:*
fixes *X :: "'a :: heine_borel set"*
fixes *f :: "'a ⇒ 'b :: Gamma"*
assumes *"continuous_on X f"* **and** *"list_all (λa. continuous_on X a) (as @ bs)"*
assumes *"length as < length bs ∨ length as = length bs ∧ (∀x∈X. norm (f x) < 1)"*

```
  assumes AB: "⋀x. x ∈ X ⟹ A x = map (λa. a x) as" "⋀x. x ∈ X ⟹
B x = map (λb. b x) bs"
  shows   "continuous_on X (λx. eval_fps (reg_fps_hypergeo_aux (A x)
(B x)) (f x))"
proof -
  have "continuous (at x within X) (λx. eval_fps (reg_fps_hypergeo_aux
(A x) (B x)) (f x))"
    if x: "x ∈ X" for x
  proof (rule continuous_within_sequentiallyI)
    fix h assume h: "h ⟶ x" "∀n. h n ∈ X"
    define X' where "X' = insert x (range h)"
    have "compact X'"
      unfolding X'_def by (rule compact_sequence_with_limit) fact
    have "X' ⊆ X"
      using x h by (auto simp: X'_def)

    have 1: "continuous_on X' f"
      by (rule continuous_on_subset[OF assms(1)]) (use ‹X' ⊆ X› in auto)
    have 2: "list_all (continuous_on X') (as @ bs)"
      unfolding list.pred_set
      by (intro ballI continuous_on_subset[of X _ X'])
         (use assms(2) h x in ‹auto simp: X'_def list.pred_set›)
    have "continuous_on X' (λx. eval_fps (reg_fps_hypergeo_aux (A x) (B
x)) (f x))"
    proof (rule uniform_limit_theorem)
      show "uniform_limit X' (λN x. ∑n<N. fps_nth (reg_fps_hypergeo_aux
(A x) (B x)) n * (f x) ^ n)
              (λx. eval_fps (reg_fps_hypergeo_aux (A x) (B x)) (f x))
sequentially"
      proof (rule uniform_limit_reg_hypergeo_F_aux[where as = as and
bs = bs])
        show "compact X'"
          by fact
      qed (use assms 1 2 h x in ‹auto simp: list.pred_set X'_def›)
    next
      have cont_A: "continuous_on X' (λx. ∏a←A x. pochhammer a n)"
for n
      proof -
        have "continuous_on X' (λx. ∏a←as. pochhammer (a x) n)"
          using 2 by (auto intro!: continuous_intros simp: list.pred_set)
        also have "?this ⟷ continuous_on X' (λx. ∏a←A x. pochhammer
a n)"
          by (intro continuous_on_cong) (use ‹X' ⊆ X› in ‹auto simp:
AB subset_iff o_def›)
        finally show ?thesis .
      qed

      have cont_B: "continuous_on X' (λx. ∏b←B x. rGamma (b + of_nat
n))" for n
```

```
    proof -
      have "continuous_on X' (λx. ∏b←bs. rGamma (b x + of_nat n))"
        using 2 by (auto intro!: continuous_intros simp: list.pred_set)
      also have "?this ⟷ continuous_on X' (λx. ∏b←B x. rGamma (b
+ of_nat n))"
        by (intro continuous_on_cong) (use ‹X' ⊆ X› in ‹auto simp:
AB subset_iff o_def›)
      finally show ?thesis .
    qed

    show "∀F n in sequentially. continuous_on X'
          (λx. ∑n<n. fps_nth (reg_fps_hypergeo_aux (A x) (B x)) n
* f x ^ n)"
      unfolding reg_fps_hypergeo_aux_def fps_nth_Abs_fps assms map_map
o_def
      using 1 h x cont_A cont_B
      by (intro always_eventually allI continuous_intros)
         (auto simp: list.pred_set prod_list_zero_iff X'_def dest!:
pochhammer_eq_0_imp_nonpos_Int)
  qed auto
  hence "((λx. eval_fps (reg_fps_hypergeo_aux (A x) (B x)) (f x)) ∘
h)
           ⟶ eval_fps (reg_fps_hypergeo_aux (A x) (B x)) (f x)"
    by (rule continuous_onD_sequentially) (use h in ‹auto simp: X'_def›)
  thus "(λn. eval_fps (reg_fps_hypergeo_aux (A (h n)) (B (h n))) (f
(h n)))
           ⟶ eval_fps (reg_fps_hypergeo_aux (A x) (B x)) (f x)"
    by (simp add: o_def)
qed
thus ?thesis
  by (auto simp: continuous_on_eq_continuous_within)
qed

corollary continuous_on_reg_hypergeo_F:
  fixes X :: "'a :: heine_borel set"
  fixes f :: "'a ⇒ 'b :: Gamma"
  assumes "continuous_on X f" and "list_all (λa. continuous_on X a) (as
@ bs)"
  assumes "length as ≤ length bs ∨ length as = length bs + 1 ∧ (∀x∈X.
norm (f x) < 1)"
  assumes "⋀x. A x = map (λa. a x) as" and "⋀x. B x = map (λb. b x)
bs"
  shows   "continuous_on X (λx. reg_hypergeo_F (A x) (B x) (f x))"
  unfolding reg_hypergeo_F_def
  by (rule continuous_on_reg_hypergeo_F_aux[where as = as and bs = "(λ_.
1) # bs"])
     (use assms in auto)

theorem has_field_derivative_reg_hypergeo_F:
```

```
  fixes as bs :: "'a :: Gamma list"
  defines "as' ≡ map (λn. n+1) as" and "bs' ≡ map (λn. n+1) bs"
  assumes norm: "length as ≤ length bs ∨ length as = Suc (length bs)
∧ norm x < 1"
  shows   "(reg_hypergeo_F as bs has_field_derivative
              (prod_list as * reg_hypergeo_F as' bs' x)) (at x within
A)"
proof -
  define F where "F = reg_fps_hypergeo_aux as (1 # bs)"
  define F' where "F' = reg_fps_hypergeo_aux as' (1 # bs')"
  define R where "R = (if length as ≤ length bs then ∞ else 1 :: ereal)"
  have x: "norm x < R"
    using norm by (auto simp: R_def)

  have "(eval_fps F has_field_derivative eval_fps (fps_deriv F) x) (at
x within A)"
  proof (rule has_field_derivative_eval_fps)
    have "ereal (norm x) < R"
      by fact
    also have "... ≤ fps_conv_radius F"
      unfolding F_def fps_conv_radius_reg_hypergeo_aux using norm
      by (auto simp: R_def one_ereal_def)
    finally show "ereal (norm x) < fps_conv_radius F" .
  qed
  also have "eval_fps F = reg_hypergeo_F as bs"
    by (simp add: F_def reg_hypergeo_F_def)
  also have "fps_deriv F = fps_const (prod_list as) * F'"
    unfolding F_def F'_def as'_def bs'_def by (subst fps_deriv_reg_hypergeo_aux1)
simp_all
  also have "eval_fps ... x = prod_list as * reg_hypergeo_F as' bs' x"
  proof (subst eval_fps_mult)
    have "norm x < R"
      by fact
    also have "R ≤ fps_conv_radius F'"
      unfolding F'_def
      by (subst fps_conv_radius_reg_hypergeo_aux)
         (use norm in ‹auto simp: as'_def bs'_def R_def one_ereal_def›)
    finally show "norm x < ..." .
  qed (auto simp: reg_hypergeo_F_def F'_def)
  finally show ?thesis .
qed

lemma has_field_derivative_reg_hypergeo_F' [derivative_intros]:
  fixes as bs :: "'a :: Gamma list"
  defines "as' ≡ map (λn. n+1) as" and "bs' ≡ map (λn. n+1) bs"
  assumes f: "(f has_field_derivative f') (at x within A)"
  assumes "length as ≤ length bs ∨ length as = Suc (length bs) ∧ norm
(f x) < 1"
  shows   "((λx. reg_hypergeo_F as bs (f x)) has_field_derivative
```

```
            (prod_list as * reg_hypergeo_F as' bs' (f x) * f')) (at
x within A)"
  unfolding as'_def bs'_def
  using DERIV_chain[OF has_field_derivative_reg_hypergeo_F f, of as bs]
assms
  by (simp add: o_def)

corollary continuous_on_reg_hypergeo_F_simple [continuous_intros]:
  assumes "continuous_on X f"
  assumes "length as ≤ length bs ∨ length as = length bs + 1 ∧ (∀x∈X.
norm (f x) < 1)"
  shows   "continuous_on X (λx. reg_hypergeo_F as bs (f x))"
proof -
  define A where "A = (if length as ≤ length bs then UNIV else ball 0
1 :: 'b set)"
  have "continuous_on A (reg_hypergeo_F as bs)"
    by (rule DERIV_continuous_on[OF has_field_derivative_reg_hypergeo_F])
       (use assms in ‹auto simp: A_def›)
  thus ?thesis
    by (rule continuous_on_compose2) (use assms in ‹auto simp: A_def›)
qed

corollary tendsto_reg_hypergeo_F_simple [tendsto_intros]:
  assumes "(f ⟶ z) F"
  assumes "length as ≤ length bs ∨ length as = length bs + 1 ∧ norm
z < 1"
  shows   "((λx. reg_hypergeo_F as bs (f x)) ⟶ reg_hypergeo_F as bs
z) F"
proof -
  define A where "A = (if length as ≤ length bs then UNIV else ball 0
1 :: 'b set)"
  have "continuous_on A (reg_hypergeo_F as bs)"
    unfolding A_def by (intro continuous_intros) (use assms in auto)
  hence "isCont (reg_hypergeo_F as bs) z"
    using assms(2) by (simp add: A_def continuous_on_eq_continuous_at
split: if_splits)
  thus ?thesis
    using assms(1) by (metis isCont_tendsto_compose)
qed

corollary continuous_reg_hypergeo_F_simple [continuous_intros]:
  assumes "continuous (at z within A) f"
  assumes "length as ≤ length bs ∨ length as = length bs + 1 ∧ norm
(f z) < 1"
  shows   "continuous (at z within A) (λx. reg_hypergeo_F as bs (f x))"
proof -
  define A where "A = (if length as ≤ length bs then UNIV else ball 0
1 :: 'b set)"
  have "continuous_on A (reg_hypergeo_F as bs)"
```

```
    unfolding A_def by (intro continuous_intros) (use assms in auto)
  hence "isCont (reg_hypergeo_F as bs) (f z)"
    using assms(2) by (simp add: A_def continuous_on_eq_continuous_at
split: if_splits)
  thus ?thesis
    using assms(1) by (metis continuous_within_compose3)
qed

corollary deriv_reg_hypergeo_F:
  fixes as bs :: "'a :: Gamma list"
  defines "as' ≡ map (λn. n+1) as" and "bs' ≡ map (λn. n+1) bs"
  assumes "length as ≤ length bs ∨ length as = Suc (length bs) ∧ norm
x < 1"
  shows   "deriv (reg_hypergeo_F as bs) x = prod_list as * reg_hypergeo_F
as' bs' x"
  by (rule DERIV_imp_deriv) (use assms in ‹auto intro!: derivative_eq_intros›)

corollary higher_deriv_reg_hypergeo_F:
  fixes as bs :: "'a :: Gamma list"
  defines "as' ≡ (λm. map (λn. n + of_nat m) as)" and "bs' ≡ (λm. map
(λn. n + of_nat m) bs)"
  defines "C ≡ (λm. (∏a←as. pochhammer a m))"
  assumes "length as ≤ length bs ∨ length as = Suc (length bs) ∧ norm
x < 1"
  shows   "(deriv ^^ m) (reg_hypergeo_F as bs) x = C m * reg_hypergeo_F
(as' m) (bs' m) x"
  using assms(4)
proof (induction m arbitrary: x)
  case (Suc m)
  define A where "A = (if length as = Suc (length bs) then ball 0 1 else
UNIV :: 'a set)"
  have "open A"
    by (auto simp: A_def)

  have "(deriv ^^ Suc m) (reg_hypergeo_F as bs) x = deriv ((deriv ^^ m)
(reg_hypergeo_F as bs)) x"
    by simp
  also have "... = deriv (λx. C m * reg_hypergeo_F (as' m) (bs' m) x)
x"
  proof (rule deriv_cong_ev)
    have "x ∈ A"
      using Suc.prems by (auto simp: A_def)
    hence "eventually (λy. y ∈ A) (nhds x)"
      by (intro eventually_nhds_in_open ‹open A›)
    thus "∀F y in nhds x. (deriv ^^ m) (reg_hypergeo_F as bs) y =
                          C m * reg_hypergeo_F (as' m) (bs' m) y"
    proof eventually_elim
      case (elim y)
      show "(deriv ^^ m) (reg_hypergeo_F as bs) y = C m * reg_hypergeo_F
```

```
(as' m) (bs' m) y"
        by (rule Suc.IH) (use Suc.prems elim in ‹auto simp: A_def›)
    qed
  qed auto
  also have "... = C (Suc m) * reg_hypergeo_F (as' (Suc m)) (bs' (Suc
m)) x"
  proof (rule DERIV_imp_deriv)
    define C' where "C' = (∏a←as. a + of_nat m)"
    have "((λx. C m * reg_hypergeo_F (as' m) (bs' m) x) has_field_derivative
            C m * C' * reg_hypergeo_F (as' (Suc m)) (bs' (Suc m)) x) (at
x)"
      by (rule derivative_eq_intros refl)+
         (use Suc.prems in ‹auto simp: as'_def bs'_def o_def C'_def pochhammer_rec
add_ac mult_ac›)
    also have "C m * C' = C (Suc m)"
      unfolding C_def C'_def pochhammer_Suc prod_list_distrib by (simp
add: field_simps)
    finally show "((λx. C m * reg_hypergeo_F (as' m) (bs' m) x) has_field_derivative
                    C (Suc m) * reg_hypergeo_F (as' (Suc m)) (bs' (Suc
m)) x) (at x)" .
  qed
  finally show ?case .
qed (auto simp: C_def as'_def bs'_def)

theorem uniform_limit_reg_hypergeo_F:
  fixes as bs :: "('a :: {topological_space} ⇒ 'b :: Gamma) list"
  assumes "compact X" and "continuous_on X g" and "list_all (continuous_on
X) (as @ bs)"
  assumes "length as ≤ length bs ∨ length as = length bs + 1 ∧ (∀x∈X.
norm (g x) < 1)"
  assumes "⋀x. A x = map (λa. a x) as" and "⋀x. B x = map (λb. b x)
bs"
  shows "uniform_limit X
         (λN x. ∑n<N. (∏a←as. pochhammer (a x) n) * (∏b←bs. rGamma
(b x + of_nat n)) *
                          (g x) ^ n / fact n)
           (λx. reg_hypergeo_F (A x) (B x) (g x)) sequentially"
proof -
  have *: "rGamma (1 + of_nat n :: 'b) = 1 / fact n" for n
    using Gamma_fact[of n, where ?'a = 'b] by (simp add: rGamma_inverse_Gamma
field_simps)
  show ?thesis
    using uniform_limit_reg_hypergeo_F_aux[of X g as "(λ_. 1) # bs" A
"λx. 1 # B x"] assms *
    by (simp_all add: reg_hypergeo_F_def reg_fps_hypergeo_aux_def o_def

         fps_hypergeo_conv_fps_hypergeo_aux less_Suc_eq_le)
qed
```

```
corollary sums_reg_hypergeo_F:
  fixes as bs :: "'a :: Gamma list"
  assumes "length as ≤ length bs ∨ length as = length bs + 1 ∧ norm
z < 1"
  shows   "(λn. (∏a←as. pochhammer a n) * (∏b←bs. rGamma (b + of_nat
n)) * z ^ n / fact n)
             sums reg_hypergeo_F as bs z"
proof -
  have "uniform_limit {z}
         (λN x. ∑n<N. (∏a←as. pochhammer a n) * (∏b←bs. rGamma
(b + of_nat n)) * x ^ n / fact n)
         (λx. reg_hypergeo_F as bs x) sequentially"
    using uniform_limit_reg_hypergeo_F[of
            "{z}" "λz. z" "map (λa x. a) as" "map (λb x. b) bs" "λx.
as" "λx. bs"] assms
    by (simp add: o_def list.pred_set)
  from tendsto_uniform_limitI[OF this, of z] show ?thesis
    by (simp add: sums_def)
qed

lemma has_fps_expansion_reg_hypergeo_F [fps_expansion_intros]:
  assumes "length as ≤ Suc (length bs)" "list_all (λb. b ∉ ℤ≤0) bs"
  shows   "(λx. reg_hypergeo_F as bs x) has_fps_expansion reg_fps_hypergeo_aux
as (1 # bs)"
  unfolding has_fps_expansion_def
proof
  show "fps_conv_radius (reg_fps_hypergeo_aux as (1 # bs)) > 0"
    by (subst fps_conv_radius_reg_hypergeo_aux) (use assms in auto)
  show "∀F z in nhds 0. eval_fps (reg_fps_hypergeo_aux as (1 # bs)) z
= reg_hypergeo_F as bs z"
    unfolding reg_hypergeo_F_def by simp
qed
```

1.5 Gauß's contiguous relations for ${}_2F_1$

Two hypergeometric series are called *contiguous* if one can obtain one from the other by adding or subtracting 1 from exactly one of the parameters. There are a number of identities that relate various contiguous hypergeometric series to one another. In this section, we will derive the classic ones for ${}_2F_1$ given by Gauß.

```
context
  fixes F :: "'a :: field_char_0 ⇒ 'a ⇒ 'a ⇒ 'a fps"
  fixes F' :: "'a :: field_char_0 ⇒ 'a ⇒ 'a ⇒ 'a fls"
  defines "F ≡ (λa b c. fps_hypergeo [a, b] [c] 1)"
  defines "F' ≡ (λa b c. fps_to_fls (fps_hypergeo [a, b] [c] 1))"
begin

lemma fps_gauss_commute: "F a b c = F b a c"
```

unfolding *F_def* **by** *(rule fps_hypergeo_cong) auto*

lemma *fls_gauss_commute: "F' a b c = F' b a c"*
unfolding *F'_def* **by** *(rule arg_cong[of _ _ fps_to_fls], rule fps_hypergeo_cong) auto*

lemma *gauss_contiguous1:*
assumes *c: "c* $\notin \mathbb{Z}_{\leq 0}$*"*
shows *"fps_X * fps_deriv (F a b c) = fps_X * fps_const (a * b / c) * F (a+1) (b+1) (c+1)"*
using *fps_deriv_hypergeo1[of "[c]" "[a,b]" 1] c* **by** *(simp add: F_def field_simps)*

lemma *gauss_contiguous2:*
assumes *c: "c* $\notin \mathbb{Z}_{\leq 0}$*"*
shows *"fps_X * fps_deriv (F a b c) = fps_const a * (F (a+1) b c - F a b c)"*
using *fps_deriv_hypergeo2[of "[c]" a "[b]" 1] c* **by** *(simp add: F_def algebra_simps)*

lemma *gauss_contiguous3:*
assumes *c: "c* $\notin \mathbb{Z}_{\leq 0}$*"*
shows *"fps_X * fps_deriv (F a b c) = fps_const b * (F a (b+1) c - F a b c)"*
using *gauss_contiguous2[of c b a] c* **by** *(simp add: fps_gauss_commute)*

lemma *gauss_contiguous4:*
assumes *c: "c* $\notin \mathbb{Z}_{\leq 0}$*" "c* $\neq$ *1"*
shows *"fps_X * fps_deriv (F a b c) = fps_const (c - 1) * (F a b (c-1) - F a b c)"*
using *fps_deriv_hypergeo3[of "[]" c "[a,b]" 1] c assms* **by** *(simp add: F_def algebra_simps)*

lemma *gauss_contiguous1':*
assumes *c: "c* $\notin \mathbb{Z}_{\leq 0}$*"*
shows *"fls_deriv (F' a b c) = fls_const (a * b / c) * F' (a+1) (b+1) (c+1)"*
using *gauss_contiguous1[of c a b] c*
by *(simp add: F'_def F_def fls_deriv_fps_to_fls fls_times_fps_to_fls)*

lemma *gauss_contiguous2':*
assumes *c: "c* $\notin \mathbb{Z}_{\leq 0}$*"*
shows *"fls_deriv (F' a b c) = fls_const a * (F' (a+1) b c - F' a b c) / fls_X"*
using *arg_cong[OF gauss_contiguous2[of c a b], of fps_to_fls] c*
by *(simp add: F'_def F_def fls_deriv_fps_to_fls fls_times_fps_to_fls divide_simps mult_ac*
del: fls_divide_X)

```
lemma gauss_contiguous3':
  assumes c: "c ∉ ℤ≤0"
  shows   "fls_deriv (F' a b c) = fls_const b * (F' a (b+1) c - F' a b
c) / fls_X"
  using gauss_contiguous2'[of c b a] c by (simp add: fls_gauss_commute)

lemma gauss_contiguous4':
  assumes c: "c ∉ ℤ≤0" "c ≠ 1"
  shows   "fls_deriv (F' a b c) = fls_const (c - 1) * (F' a b (c-1) -
F' a b c) / fls_X"
  using arg_cong[OF gauss_contiguous4[of c a b], of fps_to_fls] assms
  by (simp add: F'_def F_def fls_deriv_fps_to_fls fls_times_fps_to_fls
divide_simps mult_ac
          del: fls_divide_X)

lemma gauss_contiguous5_strong:
  assumes c: "c ∉ ℤ≤0"
  shows   "fps_X * (1 - fps_X) * fps_deriv (F a b c) =
             (fps_const (c - a) * F (a-1) b c +
             (fps_const (a - c) + fps_const b * fps_X) * F a b c)"
    (is "?lhs = ?rhs")
proof (rule fps_ext)
  fix n :: nat
  consider "n = 0" | "n = 1" | "n ≥ 2"
    by force
  thus "fps_nth ?lhs n = fps_nth ?rhs n"
  proof cases
    assume [simp]: "n = 0"
    show ?thesis unfolding ring_distribs by (simp add: F_def)
  next
    assume [simp]: "n = 1"
    from assms have "c ≠ 0"
      by auto
    thus ?thesis unfolding ring_distribs
      by (simp add: F_def fps_hypergeo_nth field_simps)
  next
    assume n: "n ≥ 2"
    define m where "m = n - 1"
    have n_conv_m: "n = Suc m"
      using n by (simp add: m_def)
    have nz2: "pochhammer c (Suc m) ≠ 0"
      using assms by (auto simp: pochhammer_eq_0_iff)
    hence nz1: "pochhammer c m ≠ 0" and nz3: "c + of_nat m ≠ (0 :: 'a)"
      by (simp_all add: pochhammer_Suc)
    define C where "C = (pochhammer a m * pochhammer b m) / (pochhammer
c m * fact m)"

    have "fps_nth ?rhs n =
          (c - a) * pochhammer b (Suc m) / (pochhammer c (Suc m) * fact
```

```
n) *
            (pochhammer (a - 1) (Suc m) - pochhammer a (Suc m)) + b *
C"
      unfolding ring_distribs mult.assoc C_def using nz1 nz2
      apply (simp add: F_def fps_hypergeo_nth n_conv_m divide_simps del:
of_nat_Suc of_nat_add)
      apply (simp add: algebra_simps del: of_nat_Suc of_nat_add)
      done
    also have "pochhammer (a - 1) (Suc m) - pochhammer a (Suc m) = -of_nat
n * pochhammer a m"
      by (simp add: pochhammer_rec[of "a - 1"] pochhammer_Suc[of a] ring_distribs
n_conv_m)
    also have "pochhammer c (Suc m) = (c + of_nat m) * pochhammer c m"
      by (simp add: pochhammer_Suc)
    also have "pochhammer b (Suc m) = (b + of_nat m) * pochhammer b m"
      by (simp add: pochhammer_Suc)
    also have "(c - a) * ((b + of_nat m) * pochhammer b m) / (((c + of_nat
m) * pochhammer c m) * fact n) *
                  (-of_nat n * pochhammer a m) =
               -(c - a) * (b + of_nat m) / (c + of_nat m) * C"
      unfolding C_def
      apply (simp del: of_nat_Suc of_nat_add add: divide_simps n_conv_m)
      apply (simp add: algebra_simps)?
      done
    also have "- (c - a) * (b + of_nat m) / (c + of_nat m) * C + b * C
=
               ((a - c) * (b + of_nat m) / (c + of_nat m) + b) * C"
      using nz3 by (simp add: field_simps)
    finally have 1: "fps_nth ?rhs n =
        ((a - c) * (b + of_nat m) / (c + of_nat m) + b) * C" .

    have "fps_nth ?lhs n =
            pochhammer a n * pochhammer b n / (pochhammer c n * fact m)
- of_nat m * C"
      unfolding ring_distribs C_def mult.assoc using n
      by (simp add: fps_hypergeo_nth Suc_diff_Suc n_conv_m F_def del:
of_nat_Suc)
    also have "pochhammer a n * pochhammer b n / (pochhammer c n * fact
m) =
                  (a + of_nat m) * (b + of_nat m) / (c + of_nat m) * C"
      unfolding C_def
      by (simp add: divide_simps n_conv_m pochhammer_Suc del: of_nat_Suc)
    also have "... - of_nat m * C = ((a + of_nat m) * (b + of_nat m) /
(c + of_nat m) - of_nat m) * C"
      by (simp add: algebra_simps)
    also have "(a + of_nat m) * (b + of_nat m) / (c + of_nat m) - of_nat
m =
               (a - c) * (b + of_nat m) / (c + of_nat m) + b"
      using nz3 by (simp add: field_simps)
```

```
    finally have 2: "fps_nth ?lhs n =
                       ((a - c) * (b + of_nat m) / (c + of_nat m) + b)
* C" .

    from 1 and 2 show "fps_nth ?lhs n = fps_nth ?rhs n"
      by simp
  qed
qed

lemma gauss_contiguous5:
  assumes c: "c ∉ ℤ≤0"
  shows   "fps_X * fps_deriv (F a b c) =
             (fps_const (c - a) * F (a-1) b c +
             (fps_const (a - c) + fps_const b * fps_X) * F a b c) / (1
- fps_X)"
  by (subst gauss_contiguous5_strong [OF c, symmetric]) simp_all

lemma gauss_contiguous6:
  assumes "c ∉ ℤ≤0"
  shows   "fps_X * fps_deriv (F a b c) =
             (fps_const (c - b) * F a (b-1) c +
             (fps_const (b - c) + fps_const a * fps_X) * F a b c) / (1
- fps_X)"
  using gauss_contiguous5[of c b a] assms by (simp add: fps_gauss_commute)

lemma gauss_contiguous5':
  assumes c: "c ∉ ℤ≤0"
  shows  "fls_deriv (F' a b c) =
            (fls_const (c - a) * F' (a-1) b c + (fls_const (a - c) + fls_const
b * fls_X) * F' a b c) /
            (fls_X * (1 - fls_X))"
  using arg_cong[OF gauss_contiguous5[OF c, of a b], of fps_to_fls]
  by (simp add: F'_def F_def fls_deriv_fps_to_fls fls_times_fps_to_fls
divide_simps mult_ac
           flip: fls_divide_fps_to_fls del: fls_divide_X)

lemma gauss_contiguous6':
  assumes c: "c ∉ ℤ≤0"
  shows "fls_deriv (F' a b c) =
           (fls_const (c - b) * F' a (b-1) c +
           (fls_const (b - c) + fls_const a * fls_X) * F' a b c) / (fls_X
* (1 - fls_X))"
  using gauss_contiguous5'[of c b a] c by (simp add: fls_gauss_commute)

lemma gauss_contiguous7_strong:
  assumes c: "c ∉ ℤ≤0"
  shows "fps_X * (1 - fps_X) * fps_deriv (F a b c) = fps_const (1/c) *
fps_X *
           (fps_const ((c-a)*(c-b)) * F a b (c+1) + fps_const (c*(a+b-c))
```

```
* F a b c)" (is "?lhs = ?rhs")
proof (rule fps_ext)
  fix n :: nat
  consider "n = 0" | "n = 1" | "n ≥ 2"
    by force
  thus "fps_nth ?lhs n = fps_nth ?rhs n"
  proof cases
    assume [simp]: "n = 0"
    show ?thesis unfolding ring_distribs by (simp add: F_def)
  next
    assume [simp]: "n = 1"
    from assms have "c ≠ 0"
      by auto
    thus ?thesis unfolding ring_distribs
      by (simp add: F_def fps_hypergeo_nth field_simps)
  next
    assume n: "n ≥ 2"
    define m where "m = n - 1"
    from assms have [simp]: "c ≠ 0"
      by auto
    have n_conv_m: "n = Suc m"
      using n by (simp add: m_def)
    have nz2: "pochhammer c (Suc m) ≠ 0"
      using assms by (auto simp: pochhammer_eq_0_iff)
    hence nz1: "pochhammer c m ≠ 0" and nz3: "c + of_nat m ≠ (0 :: 'a)"
      by (simp_all add: pochhammer_Suc)
    define C where "C = (pochhammer a m * pochhammer b m) / (pochhammer
c m * fact m)"

    have "fps_nth ?rhs n =
            (c * (c - a - b) + a * b) *
              ((pochhammer a m * pochhammer b m) / (pochhammer c (Suc
m) * fact m)) +
            (a + b - c) * C"
      unfolding ring_distribs mult.assoc C_def using nz1 nz2
      apply (simp add: F_def fps_hypergeo_nth n_conv_m pochhammer_rec
divide_simps del: of_nat_Suc of_nat_add)
      apply (simp add: algebra_simps del: of_nat_Suc of_nat_add)
      done
    also have "(pochhammer a m * pochhammer b m) / (pochhammer c (Suc
m) * fact m) =
               (1 / (c + of_nat m)) * C"
      by (simp add: pochhammer_Suc C_def)
    also have "(c * (c - a - b) + a * b) * (1 / (c + of_nat m) * C) +
(a + b - c) * C =
               ((c * (c - a - b) + a * b) / (c + of_nat m) + a + b - c)
* C"
      by Groebner_Basis.algebra
    also have "(c * (c - a - b) + a * b) / (c + of_nat m) + a + b - c
```

```
=
                (a - c) * (b + of_nat m) / (c + of_nat m) + b"
      using nz3 by (simp add: field_simps)
    finally have 1: "fps_nth ?rhs n = ((a - c) * (b + of_nat m) / (c +
of_nat m) + b) * C" .

    have "fps_nth ?lhs n =
           pochhammer a n * pochhammer b n / (pochhammer c n * fact m)
- of_nat m * C"
      unfolding ring_distribs C_def mult.assoc using n
      by (simp add: fps_hypergeo_nth Suc_diff_Suc n_conv_m F_def del:
of_nat_Suc)
    also have "pochhammer a n * pochhammer b n / (pochhammer c n * fact
m) =
                (a + of_nat m) * (b + of_nat m) / (c + of_nat m) * C"
      unfolding C_def
      by (simp add: divide_simps n_conv_m pochhammer_Suc del: of_nat_Suc)
    also have "... - of_nat m * C = ((a + of_nat m) * (b + of_nat m) /
(c + of_nat m) - of_nat m) * C"
      by (simp add: algebra_simps)
    also have "(a + of_nat m) * (b + of_nat m) / (c + of_nat m) - of_nat
m =
              (a - c) * (b + of_nat m) / (c + of_nat m) + b"
      using nz3 by (simp add: field_simps)
    finally have 2: "fps_nth ?lhs n =
                     ((a - c) * (b + of_nat m) / (c + of_nat m) + b)
* C" .

    from 1 and 2 show "fps_nth ?lhs n = fps_nth ?rhs n"
      by simp
  qed
qed

lemma gauss_contiguous7:
  assumes c: "c ∉ ℤ≤0"
  shows "fps_X * fps_deriv (F a b c) = fps_const (1/c) * fps_X *
          (fps_const ((c-a)*(c-b)) * F a b (c+1) + fps_const (c*(a+b-c))
* F a b c) / (1 - fps_X)"
  by (subst gauss_contiguous7_strong [OF assms, symmetric]) simp_all

lemma gauss_contiguous7':
  assumes c: "c ∉ ℤ≤0"
  shows "fls_deriv (F' a b c) = (fls_const ((c-a)*(c-b)) * F' a b (c+1)
+
                                fls_const (c*(a+b-c)) * F' a b c) / (fls_const
c * (1 - fls_X))"
proof -
  have "c ≠ 0"
    using c by auto
```

```
  thus ?thesis
    using arg_cong[OF gauss_contiguous7[OF c, of a b], of fps_to_fls]
    by (simp add: F'_def F_def fls_deriv_fps_to_fls fls_times_fps_to_fls
divide_simps mult_ac
            flip: fls_divide_fps_to_fls del: fls_divide_X)
qed

lemma fps_deriv_gauss_hypergeo:
  assumes c: "c ∉ ℤ≤0"
  shows "fps_deriv (F a b c) = fps_const (a * b / c) * F (a+1) (b+1) (c+1)"
  unfolding F_def by (subst fps_deriv_hypergeo1) (auto simp: c field_simps)

lemma higher_fps_deriv_gauss_hypergeo:
  assumes c: "c ∉ ℤ≤0"
  shows "(fps_deriv ^^ n) (F a b c) =
          fps_const (pochhammer a n * pochhammer b n / pochhammer c n)
*
          F (a + of_nat n) (b + of_nat n) (c + of_nat n)"
  using assms
proof (induction n arbitrary: a b c)
  case (Suc n)
  have nz: "c + of_nat n ∉ ℤ≤0"
    using Suc.prems
    by (metis diff_minus_eq_add eq_diff_eq minus_of_nat_in_nonpos_Ints
nonpos_Ints_add)
  have "(fps_deriv ^^ Suc n) (F a b c) = fps_deriv ((fps_deriv ^^ n) (F
a b c))"
    by simp
  also have "... = fps_const (pochhammer a n * pochhammer b n * inverse
(pochhammer c n)) *
                   fps_deriv (F (a + of_nat n) (b + of_nat n) (c + of_nat
n))"
    using Suc.prems by (subst Suc.IH) (auto simp: field_simps)
  also have "... = fps_const (pochhammer a (Suc n) * pochhammer b (Suc
n) * inverse (pochhammer c (Suc n))) *
                   F (a + of_nat (Suc n)) (b + of_nat (Suc n)) (c + of_nat
(Suc n))"
    by (subst fps_deriv_gauss_hypergeo)
       (use nz in ‹auto simp: pochhammer_Suc add_ac divide_inverse simp
flip: fps_const_mult›)
  finally show ?case
    by (simp add: field_simps)
qed auto



end
```

1.6 Bessel-type hypergeometric functions

We briefly look at the regularised hypergeometric funcrion ${}_0F_1$, which is closely related to Bessel functions of the first kind (it is also known as the Bessel–Clifford function).

```
definition fps_bessel :: "'a :: Gamma ⇒ 'a fps" where
  "fps_bessel a = reg_fps_hypergeo_aux [] [1, a+1]"

lemma fps_conv_radius_bessel [simp]: "fps_conv_radius (fps_bessel a)
= ∞"
  unfolding fps_bessel_def by (subst fps_conv_radius_reg_hypergeo_aux)
auto

lemma fps_deriv_bessel [simp]: "fps_deriv (fps_bessel a) = fps_bessel
(a+1)"
  by (simp add: fps_bessel_def fps_deriv_reg_hypergeo_aux1)
```

The contiguous identity for these functions has the following nice form:

```
lemma fps_bessel_contiguous:
  "fps_bessel (a-1) = fps_const a * fps_bessel a + fps_X * fps_bessel
(a+1)"
proof (rule fps_ext)
  fix n :: nat
  define F1 where "F1 = fps_X * fps_bessel (a+1)"
  define F2 where "F2 = fps_const a * fps_bessel a"
  define F3 where "F3 = fps_bessel (a-1)"

  have F1: "fps_nth F1 n = of_nat n * rGamma (1 + a + of_nat n) / fact
n"
  proof (cases n)
    case (Suc m)
    have "fps_nth F1 n =
            rGamma (1 + a + of_nat n) / Gamma (1 + of_nat m)"
      by (simp add: F1_def Suc fps_bessel_def reg_fps_hypergeo_aux_def
rGamma_inverse_Gamma field_simps)
    also have "Gamma (1 + of_nat m) = (fact m :: 'a)"
      by (rule Gamma_fact)
    also have "fact m = fact n / (of_nat n :: 'a)"
      by (simp add: Suc del: of_nat_Suc)
    finally show ?thesis
      by (simp add: Suc)
  qed (auto simp: F1_def)

  have F2: "fps_nth F2 n = a * rGamma (a + 1 + of_nat n) / fact n"
    using Gamma_fact[of n, where ?'a = 'a]
    by (auto simp: F2_def fps_bessel_def reg_fps_hypergeo_aux_def rGamma_inverse_Gamma
field_simps)

  have "fps_nth F3 n = rGamma (a + of_nat n) / fact n"
```

```
    using Gamma_fact[of n, where ?'a = 'a]
    by (auto simp: F3_def fps_bessel_def reg_fps_hypergeo_aux_def rGamma_inverse_Gamma
field_simps)
  also have "rGamma (a + of_nat n) = (a + of_nat n) * rGamma (a + of_nat
n + 1) "
    using rGamma_plus1[of "a + of_nat n"] by simp
  finally have "fps_nth F3 n = (a + of_nat n) * rGamma (a + of_nat n +
1) / fact n" .
  also have "... = fps_nth F1 n + fps_nth F2 n"
    unfolding F1 F2 by (simp add: field_simps)
  finally show "fps_nth F3 n = fps_nth (F2 + F1) n"
    by simp
qed
```

Together with the derivative identity, we find that ${}_0F_1(;a;z)$ is a solution of the ordinary differential equation $F = (a+1)F' + XF'' = 0$.

```
lemma fps_bessel_ODE:
  fixes a
  defines "D ≡ fps_deriv"
  defines "F ≡ fps_bessel a"
  shows   "F = fps_const (a+1) * D F + fps_X * (D ^^ 2) F"
  using fps_bessel_contiguous[of "a+1"]
  unfolding D_def F_def numeral_2_eq_2 funpow.simps o_def id_def fps_deriv_bessel
by simp
```

1.7 Kummer's confluent hypergeometric function

We will now look at Kummer's confluent hypergeometric function ${}_1F_1(a;b;z)$.

```
definition fps_kummer :: "'a ⇒ 'a ⇒ 'a ⇒ 'a :: field_char_0 fps"
  where "fps_kummer a b c = fps_hypergeo [a :: 'a] [b] c"

lemma fps_kummer_0_left [simp]: "fps_kummer 0 b c = 1"
  by (simp add: fps_kummer_def)
```

Kummer's series converges everywhere, so Kummer's function is entire.

```
lemma fps_conv_radius_kummer [simp]:
  fixes a b c :: "'a :: {banach, real_normed_field}"
  shows "fps_conv_radius (fps_kummer a b c) = ∞"
  unfolding fps_kummer_def by (subst fps_conv_radius_hypergeo) auto

lemma fps_kummer_same [simp]:
  assumes "a ∉ ℤ≤0"
  shows "fps_kummer a a c = fps_exp c"
  using assms by (simp add: fps_kummer_def fps_hypergeo_cancel)

lemma hypergeo_F_kummer_same [simp]:
  assumes "a ∉ ℤ≤0"
  shows   "hypergeo_F [a] [a] z = exp z"
```

```
proof -
  have "hypergeo_F [a] [a] z = eval_fps (fps_kummer a a 1) z"
    by (simp add: fps_kummer_def hypergeo_F_def)
  also have "fps_kummer a a 1 = fps_exp 1"
    using assms by simp
  also have "eval_fps ... z = exp z"
    by simp
  finally show ?thesis .
qed
```

This function satisfies the identity ${}_1F_1(a;b;z) = e^z {}_1F_1(b-a;b;-z)$, also known as the *Kummer transform*.

```
theorem fps_kummer_transform:
  assumes "b ∉ ℤ_≤0"
  shows "fps_kummer a b c = fps_exp c * fps_kummer (b - a) b (-c)"
proof (rule fps_ext)
  fix n :: nat
  have "fps_nth (fps_exp c * fps_kummer (b - a) b (-c)) n =
          (∑ i=0..n. (-1) ^ i * (c ^ i * c ^ (n - i)) * pochhammer (b
- a) i /
                       (pochhammer b i * fact i * fact (n - i)))"
    by (subst mult.commute, subst fps_mult_nth)
       (simp add: fps_hypergeo_nth fps_kummer_def field_simps power_minus')
  also have "... = c ^ n * (∑ i=0..n. (-1) ^ i * pochhammer (b - a) i
/ (pochhammer b i * fact i * fact (n - i)))"
    by (subst power_add [symmetric]) (simp add: sum_distrib_left sum_distrib_right
mult_ac)
  also have "(∑ i=0..n. (-1) ^ i * pochhammer (b - a) i / (pochhammer
b i * fact i * fact (n - i))) =
             (∑ i=0..n. of_nat (n choose i) * (-1) ^ i * pochhammer (b
- a) i / (pochhammer b i)) / fact n"
    unfolding sum_divide_distrib by (intro sum.cong) (auto simp: binomial_fact)
  also from assms have "∀ i∈{0..<n}. b ≠ - of_nat i"
    by blast
  hence "(∑ i=0..n. of_nat (n choose i) * (- 1) ^ i * pochhammer (b -
a) i / pochhammer b i) = pochhammer a n / pochhammer b n"
    using Vandermonde_pochhammer[of n b "b - a"]
    by (simp add: binomial_gbinomial gbinomial_pochhammer mult_ac)
  finally show "fps_nth (fps_kummer a b c) n = fps_nth (fps_exp c * fps_kummer
(b - a) b (-c)) n"
    by (simp add: fps_kummer_def fps_hypergeo_nth)
qed

lemma fps_kummer_transform':
  assumes "b ∉ ℤ_≤0"
  shows   "fps_exp (-c) * fps_kummer a b c = fps_kummer (b - a) b (-c)"
proof -
  have "fps_exp (-c) * fps_kummer a b c = fps_exp (-c) * (fps_exp c *
fps_kummer (b - a) b (-c))"
```

```
    by (subst fps_kummer_transform [OF assms]) (rule refl)
  also have "... = fps_kummer (b - a) b (-c)"
    by (subst mult.assoc [symmetric], subst fps_exp_add_mult [symmetric])
auto
  finally show ?thesis .
qed

lemma fps_kummer_minus:
  assumes "b ∉ ℤ_{≤0}"
  shows   "fps_kummer a b (-c) = fps_exp (-c) * fps_kummer (b - a) b c"
  using assms by (subst fps_kummer_transform) auto

lemma hypergeo_F_kummer_transform:
  fixes a b :: "'a :: {banach, real_normed_field}"
  assumes "b ∉ ℤ_{≤0}"
  shows   "hypergeo_F [a] [b] z = exp z * hypergeo_F [b - a] [b] (-z)"
proof -
  have "eval_fps (fps_exp (-1) * fps_kummer a b 1) z = exp (-z) * eval_fps
(fps_kummer a b 1) z"
    by (subst eval_fps_mult) auto
  also have "fps_exp (-1) * fps_kummer a b 1 = fps_kummer (b - a) b (-1)"
    using assms by (subst fps_kummer_transform) (simp_all flip: mult.assoc
fps_exp_add_mult)
  also have "... = fps_compose (fps_kummer (b-a) b 1) (fps_const (-1)
* fps_X)"
    by (simp add: fps_kummer_def fps_hypergeo_conv_fps_hypergeo_aux)
  also have "eval_fps ... z = hypergeo_F [b-a] [b] (-z)"
    unfolding hypergeo_F_def by (simp add: eval_fps_compose_linear fps_kummer_def)
  finally show ?thesis
    by (simp add: hypergeo_F_def fps_kummer_def exp_minus field_simps)
qed

lemma fps_kummer_1_2_aux: "fps_X * fps_kummer 1 2 1 = fps_exp 1 - (1
:: 'a :: field_char_0 fps)"
proof (rule fps_ext)
  fix n :: nat
  show "fps_nth (fps_X * fps_kummer 1 2 1) n = fps_nth (fps_exp 1 - (1
:: 'a fps)) n"
  proof (cases n)
    case (Suc m)
    have "fps_nth (fps_X * fps_kummer 1 2 1) n = 1 / pochhammer 2 m"
      by (simp add: Suc fps_kummer_def fps_hypergeo_nth flip: pochhammer_fact)
    also have "pochhammer (2::'a) m = fact n"
      using pochhammer_rec[of "1::'a" m] by (simp add: Suc pochhammer_fact)
    also have "1 / ... = fps_nth (fps_exp 1 - (1 :: 'a fps)) n"
      by (simp del: fact_Suc add: Suc)
    finally show ?thesis .
  qed auto
qed
```

```
lemma fps_kummer_1_2: "fps_kummer 1 2 1 = (fps_exp 1 - (1 :: 'a :: field_char_0
fps)) / fps_X"
  by (subst fps_kummer_1_2_aux [symmetric])  (metis fps_X_neq_zero nonzero_mult_div_cancel_

lemma hypergeo_F_1_2:
  assumes "x ≠ 0"
  shows   "hypergeo_F [1] [2] x = (exp x - 1) / x"
proof -
  have "eval_fps (fps_X * fps_kummer 1 2 1) x = x * hypergeo_F [1] [2]
x"
    by (subst eval_fps_mult) (auto simp: fps_kummer_def hypergeo_F_def
fps_conv_radius_hypergeo)
  also have "fps_X * fps_kummer 1 2 1 = fps_exp 1 - (1 :: 'a fps)"
    by (rule fps_kummer_1_2_aux)
  also have "eval_fps ... x = exp x - 1"
    by (subst eval_fps_diff) auto
  finally show ?thesis
    using assms by (simp add: field_simps)
qed
```

We derive some simple contiguous relations.

```
lemma fps_kummer_contiguous1:
  assumes "b ∉ ℤ≤0"
  shows   "fps_deriv (fps_kummer a b c) =
             fps_const (a * c / b) * fps_kummer (a+1) (b+1) c"
  using fps_deriv_hypergeo1[of "[b]" "[a]" c] assms
  by (auto simp: fps_kummer_def field_simps simp flip: fps_const_mult)

lemma fps_kummer_contiguous1':
  assumes "b ∉ ℤ≤0"
  shows   "fps_X * fps_deriv (fps_kummer a b c) =
             fps_const (a * c / b) * fps_X * fps_kummer (a+1) (b+1) c"
  by (subst fps_kummer_contiguous1[OF assms]) (simp_all add: mult_ac)

lemma fps_kummer_contiguous2:
  assumes "b ∉ ℤ≤0"
  shows   "fps_X * fps_deriv (fps_kummer a b c) =
             fps_const a * (fps_kummer (a+1) b c - fps_kummer a b c)"
  using fps_deriv_hypergeo2[of "[b]" a "[]" c] assms
  by (auto simp: fps_kummer_def field_simps simp flip: fps_const_mult)

lemma fps_kummer_contiguous3:
  assumes "b ∉ ℤ≤0" "b ≠ 1"
  shows   "fps_X * fps_deriv (fps_kummer a b c) =
             fps_const (b-1) * (fps_kummer a (b-1) c - fps_kummer a b
c)"
  using fps_deriv_hypergeo3[of "[]" b "[a]" c] assms
  by (auto simp: fps_kummer_def field_simps simp flip: fps_const_mult)
```

```
lemma fps_kummer_contiguous4:
  assumes "b ∉ ℤ≤0"
  shows   "fps_const b * (fps_kummer (a+1) b c - fps_kummer a b c) =
             fps_const c * fps_X * fps_kummer (a+1) (b+1) c" (is "?lhs
= ?rhs")
proof (cases "a = 0")
  case [simp]: True
  from assms have [simp]: "b ≠ 0"
    by auto
  show ?thesis
  proof (rule fps_ext)
    show "fps_nth ?lhs n = fps_nth ?rhs n" for n
    proof (cases n)
      case (Suc m)
      have "pochhammer 2 m = (fact (Suc m) :: 'a)"
        unfolding pochhammer_fact by (simp add: pochhammer_rec)
      thus ?thesis
        by (simp add: Suc mult.assoc fps_kummer_def fps_hypergeo_nth pochhammer_rec
                 flip: pochhammer_fact pochhammer_of_nat del: of_nat_Suc)
    qed (auto simp: fps_kummer_def)
  qed
next
  case [simp]: False
  have [simp]: "b ≠ 0"
    using assms by auto
  have "fps_const a * (fps_const b * (fps_kummer (a+1) b c - fps_kummer
a b c)) =
        fps_const b * (fps_const a * (fps_kummer (a+1) b c - fps_kummer
a b c))"
    by (simp only: mult_ac)
  also have "... = fps_const b * (fps_X * fps_deriv (fps_kummer a b c))"
    by (subst fps_kummer_contiguous2 [symmetric]) (use assms in auto)
  also have "... = fps_const a * (fps_const c * fps_X * fps_kummer (a
+ 1) (b + 1) c)"
    by (subst fps_kummer_contiguous1') (use assms in auto)
  finally show ?thesis
    by (subst (asm) mult_cancel_left) auto
qed

lemma fps_kummer_contiguous5:
  assumes "b ∉ ℤ≤0" "b ≠ 1"
  shows   "fps_const (b*(b-1)) * (fps_kummer a (b - 1) c - fps_kummer
a b c) =
             fps_const (a * c) * fps_X * fps_kummer (a+1) (b+1) c"
proof -
  have [simp]: "b ≠ 0"
    using assms by auto
  have "fps_const (b*(b-1)) * (fps_kummer a (b - 1) c - fps_kummer a b
```

```
c) =
        fps_const b * (fps_const (b-1) * (fps_kummer a (b - 1) c - fps_kummer
a b c))"
    by (simp only: mult_ac flip: fps_const_mult)
  also have "... = fps_const b * (fps_X * fps_deriv (fps_kummer a b c))"
    by (subst fps_kummer_contiguous3 [symmetric]) (use assms in auto)
  also have "... = fps_const b * fps_const (inverse b) *
                   fps_X * fps_const a * fps_const c * fps_kummer (a
+ 1) (b + 1) c"
    by (subst fps_kummer_contiguous1) (use assms in ‹auto simp flip: fps_const_mult
fps_const_divide›)
  also have "fps_const b * fps_const (inverse b) = 1"
    by simp
  finally show ?thesis
    by simp
qed
```

Kummer's function is a solution of the ODE $af(z)-(b-z)f'(z)-zf''(z) = 0$.

```
theorem fps_kummer_ODE:
  fixes a b :: "'a :: field_char_0"
  defines "W ≡ fps_kummer a b 1"
  assumes b: "b ∉ ℤ≤0"
  shows   "fps_const a * W - (fps_const b - fps_X) * fps_deriv W - fps_X
* (fps_deriv ^^ 2) W = 0"
proof -
  have [simp]: "b ≠ 0"
    using b by auto
  have b': "b + 1 ≠ 0"
    using b by (auto simp: add_eq_0_iff2)
  have b'': "1 + b ∉ ℤ≤0"
    using b by (metis add.commute plus_one_in_nonpos_Ints_imp)

  define F where "F = (λa b. fps_to_fls (fps_kummer a b 1) :: 'a fls)"
  have F_def: "F a b = fps_to_fls (fps_kummer a b 1)" for a b
    by (simp add: F_def)

  have F1: "fls_deriv (F a b) = fls_const (a / b) * F (a+1) (b+1)"
    unfolding F_def fls_deriv_fps_to_fls using b
    by (subst fps_kummer_contiguous1) (auto simp: fls_times_fps_to_fls)
  have F2: "(fls_deriv ^^ 2) (F a b) = fls_const (a / (b*(b+1))) * (fls_const
(a+1) * F (a+2) (b+2))"
    unfolding numeral_2_eq_2 funpow.simps o_def id_def F1 fls_deriv_mult
    unfolding F_def fls_deriv_fps_to_fls using b''
    by (subst fps_kummer_contiguous1)
       (auto simp: fls_times_fps_to_fls mult_ac add_ac fls_deriv_divide
             simp flip: fls_const_divide_const fls_const_mult_const)
  have F3: "F (a+1) (b+1) = fls_const b / fls_X * (F (a+1) b - F a b)"
if "b ∉ ℤ≤0" for a b
  proof -
```

```
    have "fps_to_fls (fps_const b * (fps_kummer (a + 1) b 1 - fps_kummer
a b 1)) =
          fps_to_fls (fps_X * fps_kummer (a + 1) (b + 1) 1)"
      using fps_kummer_contiguous4[of b a 1] that by simp
    thus ?thesis
      unfolding fls_times_fps_to_fls F_def
      by (auto simp: field_simps fls_X_conv_shift_1 fls_X_intpow_times_conv_shift)
  qed
  have F4: "fls_const a * F (a+1) (b+2) = fls_const (b*(b+1)) / fls_X
* (F a b - F a (b+1))" for a
  proof -
    have "fps_to_fls (fps_const (b*(b+1)) * (fps_kummer a b 1 - fps_kummer
a (b+1) 1)) =
          fps_to_fls (fps_const a * fps_X * fps_kummer (a + 1) (b + 2)
1)"
      using fps_kummer_contiguous5[of "b+1" a 1] b''
      by (intro arg_cong[of _ _ fps_to_fls]) (simp add: mult_ac add_ac)
    also have "fps_to_fls (fps_const (b*(b+1)) * (fps_kummer a b 1 - fps_kummer
a (b+1) 1)) =
               fls_const (b*(b+1)) * (F a b - F a (b+1))"
      by (simp add: F_def fls_times_fps_to_fls)
    also have "fps_to_fls (fps_const a * fps_X * fps_kummer (a + 1) (b
+ 2) 1) =
               fls_const a * fls_X * F (a+1) (b+2)"
      by (simp add: F_def fls_times_fps_to_fls)
    finally show ?thesis
      by (simp add: divide_simps fls_X_conv_shift_1)
  qed

  have F5: "fls_const (a + 1) * F (a+2) (b+2) =
            fls_const (b * (b + 1)) / fls_X * (F (a + 1) b - fls_const
b / fls_X * (F (a + 1) b - F a b))"
  proof -
    have "fls_const (a + 1) * F (a+1+1) (b+2) =
            fls_const (b * (b + 1)) / fls_X * (F (a + 1) b - fls_const
b / fls_X * (F (a + 1) b - F a b))"
      by (subst F4, subst F3) (use b in auto)
    thus ?thesis
      by (simp add: add_ac)
  qed

  have nz: "1 + fls_const b ≠ 0"
    using ‹b + 1 ≠ 0› by (auto simp: fls_eq_iff add_ac)
  have "fps_to_fls (fps_const a * W - fps_X * (fps_deriv ^^ 2) W - (fps_const
b - fps_X) * fps_deriv W) =
          fls_const a * F a b - fls_X * (fls_deriv ^^ 2) (F a b) - (fls_const
b - fls_X) * fls_deriv (F a b)"
    by (simp add: F_def fls_times_fps_to_fls eval_nat_numeral fls_deriv_fps_to_fls
W_def)
```

```
  also have "... = fps_to_fls 0"
    unfolding F1 F2 F5 F3[OF b] using ‹b + 1 ≠ 0› nz
    apply (simp add: divide_simps fls_X_conv_shift_1 add_ac
                flip: fls_const_divide_const fls_const_mult_const fls_plus_const)
    apply (simp add: field_simps fls_shifted_times_simps del: fls_const_mult_const)?
    done
  finally have "fps_const a * W - fps_X * (fps_deriv ^^ 2) W - (fps_const
b - fps_X) * fps_deriv W = 0"
    by (subst (asm) fps_to_fls_eq_iff)
  thus ?thesis
    by Groebner_Basis.algebra
qed
```

As an application, we show that the error function can be expression terms of the hypergeometric function ${}_1F_1(\frac{1}{2}; \frac{3}{2}; -z^2)$.

The error function is the unique function such that $\operatorname{erf}(0) = 0$ and $\operatorname{erf}'(z) = \frac{2}{\pi}\exp(-z^2)$. Or, in other words:

$$\operatorname{erf}(z) = \frac{2}{\pi}\int_0^x \exp(-t^2)\,\mathrm{d}t$$

This function is already available in the AFP and it is defined there using its Maclaurin series, so it is easy to prove the identity we want just by comparing coefficients.

```
theorem erf_conv_hypergeo_F:
  fixes z :: "'a :: {banach, real_normed_field}"
  shows "erf z = of_real (2 / sqrt pi) * z * hypergeo_F [1/2] [3/2] (-(z
^ 2))"
proof -
  have "(λn. (2 / of_real (sqrt pi)) *R
                (z * (fps_nth (fps_hypergeo [1 / 2] [3 / 2] 1) n * (-(z^2))
^ n)))
          sums ((2 / of_real (sqrt pi)) *R (z * hypergeo_F [1 / 2] [3
/ 2] (-(z^2))))"
    by (intro sums_scaleR_right sums_mult sums_hypergeo_F) auto
  also have "(λn. (2 / of_real (sqrt pi)) *R
                    (z * (fps_nth (fps_hypergeo [1 / 2] [3 / 2] 1) n *
(-(z^2)) ^ n))) =
             (λn. erf_coeffs (2 * n + 1) *R z ^ (2 * n + 1))" (is "?lhs
= ?rhs")
  proof
    fix n :: nat
    have nz: "pochhammer (1/2) n ≠ (0::'a)"
      using pochhammer_eq_0_imp_nonpos_Int[of "1/2::'a" n] by (auto dest!:
nonpos_Ints_Int)
    have "?lhs n = 2 * z * (- z^2) ^ n / of_real (sqrt pi) / fact n *
                    (pochhammer (1/2) n / pochhammer (3/2) n)"
      by (simp add: fps_hypergeo_nth scaleR_conv_of_real pochhammer_prod
mult_ac)
```

```
    also have "pochhammer (3/2 :: 'a) n = 2 * pochhammer (1 / 2) (Suc
n)"
      using pochhammer_rec[of "1/2 :: 'a" n] by (simp add: mult_ac)
    also have "pochhammer (1/2) n / ... = 1 / (1 + 2 * of_nat n)"
      using nz by (simp add: pochhammer_Suc)
    also have "2 * z * (- z^2) ^ n / of_real (sqrt pi) / fact n * (1 /
(1 + 2 * of_nat n)) = ?rhs n"
      by (simp add: erf_coeffs_def scaleR_conv_of_real mult_ac power_minus'
flip: power_mult)
    finally show "?lhs n = ?rhs n" .
  qed
  finally have "((λn. erf_coeffs n *R z ^ n) ∘ (λn. 2 * n + 1)) sums
    ((2 / of_real (sqrt pi)) *R (z * hypergeo_F [1 / 2] [3 / 2] (- z^2)))"
    by (simp add: o_def)
  also have "?this ⟷ (λn. erf_coeffs n *R z ^ n) sums
               ((2 / of_real (sqrt pi)) *R (z * hypergeo_F [1 / 2] [3
/ 2] (- z^2)))"
    unfolding o_def
    by (rule sums_mono_reindex)
       (auto intro!: strict_monoI simp: erf_coeffs_def elim!: oddE)
  finally show ?thesis
    using erf_converges[of z]
    by (simp add: sums_iff scaleR_conv_of_real algebra_simps)
qed

end
```

References

[1] D. Duverney. *An Introduction to Hypergeometric Functions*. Springer, 2024.