The Euler–MacLaurin summation formula

Manuel Eberl

September 9, 2026

Abstract

The Euler–MacLaurin formula relates the value of a discrete sum $\sum_{i=a}^{b} f(i)$ to that of the integral $\int_a^b f(x)\,\mathrm{d}x$ in terms of the derivatives of f at a and b and a remainder term. Since the remainder term is often very small as b grows, this can be used to compute asymptotic expansions for sums.

This entry contains a proof of this formula for functions from the reals to an arbitrary Banach space. Two variants of the formula are given: the standard textbook version and a variant outlined in *Concrete Mathematics* [3] that is more useful for deriving asymptotic estimates.

As example applications, we use that formula to derive the full asymptotic expansion of the harmonic numbers and the sum of inverse squares.

Contents

1 The Euler–MacLaurin summation formula **2**
- 1.1 Auxiliary facts . . . 2
- 1.2 The remainder terms . . . 3
- 1.3 The conventional version of the Euler–MacLaurin formula . . 22
- 1.4 The "Concrete Mathematics" version of the Euler–MacLaurin formula . . . 24
- 1.5 Bounds on the remainder term . . . 30
- 1.6 Application to harmonic numbers . . . 34
- 1.7 Application to sums of inverse squares . . . 36

2 Connection of Euler–MacLaurin summation to Landau symbols **37**
- 2.1 O-bound for the remainder term . . . 38
- 2.2 Asymptotic expansion of the harmonic numbers . . . 38
- 2.3 Asymptotic expansion of the sum of inverse squares . . . 40

1 The Euler–MacLaurin summation formula

theory *Euler-MacLaurin*
imports
HOL−Complex-Analysis.Complex-Analysis
Bernoulli.Periodic-Bernpoly
Bernoulli.Bernoulli-FPS
begin

1.1 Auxiliary facts

lemma *pbernpoly-of-int* [*simp*]: *pbernpoly n* (*of-int a*) = *bernoulli n*
by (*simp add*: *pbernpoly-def*)

lemma *continuous-on-bernpoly'* [*continuous-intros*]:
assumes *continuous-on A f*
shows *continuous-on A* (λ*x*. *bernpoly n* (*f x*) :: *'a* :: *real-normed-algebra-1*)
using *continuous-on-compose2*[*OF continuous-on-bernpoly assms*, *of UNIV n*] **by** *auto*

lemma *sum-atLeastAtMost-int-last*:
assumes *a* < (*b* :: *int*)
shows *sum f* {*a*..*b*} = *sum f* {*a*..<*b*} + *f b*
proof −
from *assms* **have** {*a*..*b*} = *insert b* {*a*..<*b*} **by** *auto*
also have *sum f* ... = *sum f* {*a*..<*b*} + *f b*
by (*subst sum.insert*) (*auto simp*: *add-ac*)
finally show *?thesis* **.**
qed

lemma *sum-atLeastAtMost-int-head*:
assumes *a* < (*b* :: *int*)
shows *sum f* {*a*..*b*} = *f a* + *sum f* {*a*<..*b*}
proof −
from *assms* **have** {*a*..*b*} = *insert a* {*a*<..*b*} **by** *auto*
also have *sum f* ... = *f a* + *sum f* {*a*<..*b*}
by (*subst sum.insert*) *auto*
finally show *?thesis* **.**
qed

lemma *not-in-nonpos-Reals-imp-add-nonzero*:
assumes *z* ∉ $\mathbb{R}_{\leq 0}$ *x* ≥ *0*
shows *z* + *of-real x* ≠ *0*
using *assms* **by** (*auto simp*: *add-eq-0-iff2*)

lemma *negligible-atLeastAtMostI*: *b* ≤ *a* ⟹ *negligible* {*a*..(*b*::*real*)}
by (*cases b* < *a*) *auto*

lemma *integrable-on-negligible*:

negligible A $\Longrightarrow$ (*f* :: *'n* :: *euclidean-space* $\Rightarrow$ *'a* :: *banach*) *integrable-on A*
by (*subst integrable-spike-set-eq*[*of* - {}]) (*simp-all add*: *integrable-on-empty*)

lemma *Union-atLeastAtMost-real-of-int*:
assumes *a* < *b*
shows ($\bigcup$ *n*∈{*a*..<*b*}. {*real-of-int n*..*real-of-int* (*n* + *1*)}) = {*real-of-int a*..*real-of-int b*}
proof (*intro equalityI subsetI*)
fix *x* **assume** *x*: *x* ∈ {*real-of-int a*..*real-of-int b*}
thus *x* ∈ ($\bigcup$ *n*∈{*a*..<*b*}. {*real-of-int n*..*real-of-int* (*n* + *1*)})
proof (*cases x* = *real-of-int b*)
case *True*
with *assms* **show** *?thesis* **by** (*auto intro*!: *bexI*[*of* - *b* − *1*])
next
case *False*
with *x* **have** *x*: *x* ≥ *real-of-int a x* < *real-of-int b* **by** *simp-all*
hence *x* ≥ *of-int* ⌊*x*⌋ *x* ≤ *of-int* ⌊*x*⌋ + *1* **by** *linarith*+
moreover from *x* **have** ⌊*x*⌋ ≥ *a* ⌊*x*⌋ < *b* **by** *linarith*+
ultimately have ∃ *n*∈{*a*..<*b*}. *x* ∈ {*of-int n*..*of-int* (*n* + *1*)}
by (*intro bexI*[*of* - ⌊*x*⌋]) *simp-all*
thus *?thesis* **by** *blast*
qed
qed *auto*

1.2 The remainder terms

The following describes the remainder term in the classical version of the Euler–MacLaurin formula.

definition *EM-remainder'* :: *nat* ⇒ (*real* ⇒ *'a* :: *banach*) ⇒ *real* ⇒ *real* ⇒ *'a*
where
EM-remainder' n f a b = ((−*1*) ^ *Suc n* / *fact n*) ∗$_R$ *integral* {*a*..*b*} (λ*t*. *pbernpoly n t* ∗$_R$ *f t*)

Next, we define the remainder term that occurs when one lets the right bound of summation in the Euler–MacLaurin formula tend to infinity.

definition *EM-remainder-converges* :: *nat* ⇒ (*real* ⇒ *'a* :: *banach*) ⇒ *int* ⇒ *bool*
where
EM-remainder-converges n f a ⟷ (∃ *L*. ((λ*x*. *EM-remainder' n f a* (*of-int x*)) ⟶ *L*) *at-top*)

definition *EM-remainder* :: *nat* ⇒ (*real* ⇒ *'a* :: *banach*) ⇒ *int* ⇒ *'a* **where**
EM-remainder n f a =
(*if EM-remainder-converges n f a then*
Lim at-top (λ*x*. *EM-remainder' n f a* (*of-int x*)) *else 0*)

The following lemmas are fairly obvious – but tedious to prove – properties of the remainder terms.

lemma *EM-remainder-eqI*:

```
  fixes L
  assumes ((λx. EM-remainder′ n f b (of-int x)) ⟶ L) at-top
  shows   EM-remainder n f b = L
  using assms by (auto simp: EM-remainder-def EM-remainder-converges-def intro!: tendsto-Lim)

lemma integrable-EM-remainder′-int:
  fixes a b :: int and f :: real ⇒ 'a :: banach
  assumes continuous-on {of-int a..of-int b} f
  shows   (λt. pbernpoly n t *R f t) integrable-on {a..b}
proof −
  have [continuous-intros]: continuous-on A f if A ⊆ {of-int a..of-int b} for A
    using continuous-on-subset[OF assms that] .
  consider a > b | a = b | a < b n = 1 | a < b n ≠ 1
    by (cases a b rule: linorder-cases) auto
  thus ?thesis
  proof cases
    assume a < b and n ≠ 1
    thus ?thesis by (intro integrable-continuous-real continuous-intros) auto
  next
    assume ab: a < b and [simp]: n = 1
    let ?A = (λn. {real-of-int n..real-of-int (n+1)}) ` {a..<b}
    show ?thesis
    proof (rule integrable-combine-division; (intro ballI)?)
      show ?A division-of {of-int a..of-int b}
       using Union-atLeastAtMost-real-of-int[OF ab] by (simp add: division-of-def)
    next
      fix I assume I ∈ ?A
      then obtain i where i: i ∈ {a..<b} I = {of-int i..of-int (i + 1)} by auto
      show (λt. pbernpoly n t *R f t) integrable-on I
      proof (rule integrable-spike)
        show (λt. (t − of-int i − 1/2) *R f t) integrable-on I
          using i by (auto intro!: integrable-continuous-real continuous-intros)
      next
        fix x assume x ∈ I − {of-int (i + 1)}
        with i have of-int i ≤ x x < of-int i + 1 by simp-all
        hence floor x = i by linarith
        thus pbernpoly n x *R f x = (x − of-int i − 1 / 2) *R f x
          by (simp add: pbernpoly-def bernpoly-def frac-def)
      qed simp-all
    qed
  qed (simp-all add: integrable-on-negligible)
qed

lemma integrable-EM-remainder′:
  fixes a b :: real and f :: real ⇒ 'a :: banach
  assumes continuous-on {a..b} f
  shows   (λt. pbernpoly n t *R f t) integrable-on {a..b}
proof (cases ⌈a⌉ ≤ ⌊b⌋)
```

```
  case True
  define a' b' where a' = ⌈a⌉ and b' = ⌊b⌋
  from True have *: a' ≤ b' a' ≥ a b' ≤ b by (auto simp: a'-def b'-def)
  from * have A: (λt. pbernpoly n t *R f t) integrable-on ({a'..b'})
    by (intro integrable-EM-remainder'-int continuous-on-subset[OF assms]) auto
  have B: (λt. pbernpoly n t *R f t) integrable-on ({a..a'})
  proof (rule integrable-spike)
    show pbernpoly n x *R f x = bernpoly n (x − of-int (floor a)) *R f x
        if x ∈ {a..real-of-int a'} − {real-of-int a'}for x
    proof −
      have x ≥ a x <real-of-int a' using that by auto
      with True have floor x = floor a unfolding a'-def
        using ceiling-diff-floor-le-1[of a] by (intro floor-unique; linarith)
      thus ?thesis by (simp add: pbernpoly-def frac-def)
    qed
  qed (insert *, auto intro!: continuous-intros integrable-continuous-real
       continuous-on-subset[OF assms])
  have C: (λt. pbernpoly n t *R f t) integrable-on ({b'..b})
  proof (rule integrable-spike)
    show pbernpoly n x *R f x = bernpoly n (x − of-int b') *R f x
        if x ∈ {real-of-int b'..b} − {real-of-int b'} for x
    proof −
      have x ≤ b x > real-of-int b' using that by auto
     with True have floor x = b' unfolding b'-def by (intro floor-unique; linarith)

      thus ?thesis by (simp add: pbernpoly-def frac-def)
    qed
  qed (insert *, auto intro!: continuous-intros integrable-continuous-real
       continuous-on-subset[OF assms])
  have (λt. pbernpoly n t *R f t) integrable-on ({a..a'} ∪ {a'..b'} ∪ {b'..b}) using
* A B C
    by (intro integrable-Un; (subst ivl-disj-un)?)
       (auto simp: ivl-disj-un max-def min-def)
  also have {a..a'} ∪ {a'..b'} ∪ {b'..b} = {a..b} using * by auto
  finally show ?thesis .
next
  assume *: ¬ceiling a ≤ floor b
  show ?thesis
  proof (rule integrable-spike)
    show (λt. bernpoly n (t − floor a) *R f t) integrable-on {a..b} using *
      by (auto intro!: integrable-continuous-real continuous-intros assms)
  next
    show pbernpoly n x *R f x = bernpoly n (x − floor a) *R f x
        if x ∈ {a..b} − {} for x
    proof −
      from * have **: b < floor a + 1
        unfolding ceiling-altdef by (auto split: if-splits simp: le-floor-iff)
      from that have x: x ≥ a x ≤ b by simp-all
      with * ** have floor x = floor a by linarith
```

```
    thus ?thesis by (simp add: pbernpoly-def frac-def)
   qed
  qed simp-all
qed

lemma EM-remainder'-bounded-linear:
  assumes bounded-linear h
  assumes continuous-on {a..b} f
  shows   EM-remainder' n (λx. h (f x)) a b = h (EM-remainder' n f a b)
proof −
  have integral {a..b} (λt. pbernpoly n t *R h (f t)) =
         integral {a..b} (λt. h (pbernpoly n t *R f t)) using assms
    by (simp add: linear-simps)
  also have ... = h (integral {a..b} (λt. pbernpoly n t *R f t))
    by (subst integral-linear [OF - assms(1), symmetric])
      (auto intro!: integrable-EM-remainder' assms(2) simp: o-def)
  finally show ?thesis using assms(1)
    by (simp add: EM-remainder'-def linear-simps)
qed

lemma EM-remainder-converges-of-real:
  assumes EM-remainder-converges n f a continuous-on {of-int a..} f
  shows   EM-remainder-converges n (λx. of-real (f x)) a
proof −
  from assms obtain L
   where L: ((λb. EM-remainder' n f (real-of-int a) (real-of-int b)) ⟶ L) at-top
    by (auto simp: EM-remainder-converges-def)
  have ((λb. EM-remainder' n (λx. of-real (f x)) (of-int a) (of-int b)) ⟶ of-real
L) at-top
  proof (rule Lim-transform-eventually)
    show eventually (λb. of-real (EM-remainder' n f (of-int a) (of-int b)) =
          EM-remainder' n (λx. of-real (f x)) (of-int a) (of-int b)) at-top
      using eventually-ge-at-top[of a]
      by eventually-elim
       (intro EM-remainder'-bounded-linear [OF bounded-linear-of-real, symmetric]
             continuous-on-subset[OF assms(2)], auto)
  qed (intro tendsto-intros L)
  thus ?thesis unfolding EM-remainder-converges-def ..
qed

lemma EM-remainder-converges-of-real-iff:
  fixes f :: real ⇒ real
  assumes continuous-on {of-int a..} f
  shows   EM-remainder-converges n (λx. of-real (f x) ::
        'a :: {banach, real-normed-algebra-1, real-inner}) a ⟷ EM-remainder-converges
n f a
proof
  assume EM-remainder-converges n (λx. of-real (f x) :: 'a) a
  then obtain L :: 'a
```

where *L*: ((λ*b*. *EM-remainder′ n* (λ*x*. *of-real* (*f x*)) (*of-int a*) (*of-int b*)) ⟶ *L*) *at-top*
by (*auto simp*: *EM-remainder-converges-def*)
have ((λ*b*. *EM-remainder′ n f* (*of-int a*) (*of-int b*)) ⟶ *L* · *1*) *at-top*
proof (*rule Lim-transform-eventually*)
show *eventually* (λ*b*. *EM-remainder′ n* (λ*x*. *of-real* (*f x*) :: *′a*) (*of-int a*) (*of-int b*) · *1* =
EM-remainder′ n f (*of-int a*) (*of-int b*)) *at-top* **using** *eventually-ge-at-top*[*of a*]
by *eventually-elim*
(*subst EM-remainder′-bounded-linear* [*OF bounded-linear-of-real*],
auto intro!: *continuous-on-subset*[*OF assms*])
qed (*intro tendsto-intros L*)
thus *EM-remainder-converges n f a* **unfolding** *EM-remainder-converges-def* **..**
qed (*intro EM-remainder-converges-of-real assms*)

lemma *EM-remainder-of-real*:
assumes *continuous-on* {*a*..} *f*
shows *EM-remainder n* (λ*x*. *of-real* (*f x*) ::
′a :: {*banach*, *real-normed-algebra-1*, *real-inner*}) *a* =
of-real (*EM-remainder n f a*)
proof −
have *eq*: *EM-remainder′ n* (λ*x*. *of-real* (*f x*) :: *′a*) (*real-of-int a*) =
(λ*x*::*int*. *of-real* (*EM-remainder′ n f a x*))
by (*intro ext EM-remainder′-bounded-linear*[*OF bounded-linear-of-real*]
continuous-on-subset[*OF assms*]) *auto*
show *?thesis*
proof (*cases EM-remainder-converges n f a*)
case *False*
with *EM-remainder-converges-of-real-iff*[*OF assms*, *of n*] **show** *?thesis*
by (*auto simp*: *EM-remainder-def*)
next
case *True*
then obtain *L* **where** *L*: ((λ*x*. *EM-remainder′ n f a* (*real-of-int x*)) ⟶ *L*) *at-top*
by (*auto simp*: *EM-remainder-converges-def*)
have *L′*: ((λ*x*. *EM-remainder′ n* (λ*x*. *of-real* (*f x*) :: *′a*) *a*
(*real-of-int x*)) ⟶ *of-real L*) *at-top* **unfolding** *eq* **by** (*intro tendsto-of-real L*)
from *L L′ tendsto-Lim*[*OF - L*] *tendsto-Lim*[*OF - L′*] **show** *?thesis*
by (*auto simp*: *EM-remainder-def EM-remainder-converges-def*)
qed
qed

lemma *EM-remainder′-cong*:
assumes ⋀*x*. *x* ∈ {*a*..*b*} ⟹ *f x* = *g x n* = *n′ a* = *a′ b* = *b′*
shows *EM-remainder′ n f a b* = *EM-remainder′ n′ g a′ b′*
proof −
have *integral* {*a*..*b*} (λ*t*. *pbernpoly n t* ∗*R f t*) = *integral* {*a′*..*b′*} (λ*t*. *pbernpoly*

$n'\ t *_R g\ t$)
  unfolding *assms* **using** *assms* **by** (*intro integral-cong*) *auto*
 with *assms* **show** *?thesis* **by** (*simp add*: *EM-remainder'-def*)
qed

lemma *EM-remainder-converges-cong*:
 assumes $\bigwedge x.\ x \geq$ *of-int a* $\Longrightarrow f\ x = g\ x\ n = n'\ a = a'$
 shows *EM-remainder-converges n f a* = *EM-remainder-converges n' g a'*
 unfolding *EM-remainder-converges-def*
 by (*subst EM-remainder'-cong*[*OF - refl refl refl*, *of - - f g*]) (*use assms* **in** *auto*)

lemma *EM-remainder-cong*:
 assumes $\bigwedge x.\ x \geq$ *of-int a* $\Longrightarrow f\ x = g\ x\ n = n'\ a = a'$
 shows *EM-remainder n f a* = *EM-remainder n' g a'*
proof −
 have $*$: *EM-remainder-converges n f a* = *EM-remainder-converges n' g a'*
  using *assms* **by** (*intro EM-remainder-converges-cong*) *auto*
 show *?thesis* **unfolding** *EM-remainder-def*
  by (*subst EM-remainder'-cong*[*OF - refl refl refl*, *of - - f g*]) (*use assms* $*$ **in** *auto*)
qed

lemma *EM-remainder-converges-cnj*:
 assumes *continuous-on* {*a*..} *f* **and** *EM-remainder-converges n f a*
 shows *EM-remainder-converges n* (λ*x*. *cnj* (*f x*)) *a*
proof −
 interpret *bounded-linear cnj* **by** (*rule bounded-linear-cnj*)
 obtain *L* **where** *L*: ((λ*x*. *EM-remainder' n f* (*real-of-int a*) (*real-of-int x*)) ⟶ *L*) *at-top*
  using *assms* **unfolding** *EM-remainder-converges-def* **by** *blast*
 note *tendsto-cnj* [*OF this*]
 also have (λ*x*. *cnj* (*EM-remainder' n f* (*real-of-int a*) (*real-of-int x*))) =
          (λ*x*. *EM-remainder' n* (λ*x*. *cnj* (*f x*)) (*real-of-int a*) (*real-of-int x*))
  by (*subst EM-remainder'-bounded-linear* [*OF bounded-linear-cnj*])
    (*rule continuous-on-subset* [*OF assms*(*1*)], *auto*)
 finally have *L'*: (*...* ⟶ *cnj L*) *at-top* **.**
 thus *EM-remainder-converges n* (λ*x*. *cnj* (*f x*)) *a*
  by (*auto simp*: *EM-remainder-converges-def*)
qed

lemma *EM-remainder-converges-cnj-iff*:
 assumes *continuous-on* {*of-int a*..} *f*
 shows *EM-remainder-converges n* (λ*x*. *cnj* (*f x*)) *a* ⟷ *EM-remainder-converges n f a*
proof
 assume *EM-remainder-converges n* (λ*x*. *cnj* (*f x*)) *a*
 hence *EM-remainder-converges n* (λ*x*. *cnj* (*cnj* (*f x*))) *a*
  by (*rule EM-remainder-converges-cnj* [*rotated*]) (*auto intro*: *continuous-intros assms*)

thus *EM-remainder-converges n f a* **by** *simp*
qed (*intro EM-remainder-converges-cnj assms*)

lemma *EM-remainder-cnj*:
assumes *continuous-on* {*a*..} *f*
shows *EM-remainder n* (λ*x*. *cnj* (*f x*)) *a* = *cnj* (*EM-remainder n f a*)
proof (*cases EM-remainder-converges n f a*)
case *False*
hence ¬*EM-remainder-converges n* (λ*x*. *cnj* (*f x*)) *a*
by (*subst EM-remainder-converges-cnj-iff* [*OF assms*])
with *False* **show** *?thesis* **by** (*simp add*: *EM-remainder-def*)
next
case *True*
then obtain *L* **where** *L*: ((λ*x*. *EM-remainder′ n f* (*real-of-int a*) (*real-of-int x*))
⟶ *L*) *at-top*
unfolding *EM-remainder-converges-def* **by** *blast*
note *tendsto-cnj* [*OF this*]
also have (λ*x*. *cnj* (*EM-remainder′ n f* (*real-of-int a*) (*real-of-int x*))) =
(λ*x*. *EM-remainder′ n* (λ*x*. *cnj* (*f x*)) (*real-of-int a*) (*real-of-int x*))
by (*subst EM-remainder′-bounded-linear* [*OF bounded-linear-cnj*])
(*rule continuous-on-subset* [*OF assms*(*1*)], *auto*)
finally have *L′*: (*...* ⟶ *cnj L*) *at-top* **.**
moreover from *assms* **and** *L* **have** *EM-remainder n f a* = *L*
by (*intro EM-remainder-eqI*)
ultimately show *EM-remainder n* (λ*x*. *cnj* (*f x*)) *a* = *cnj* (*EM-remainder n f*
a)
using *L′* **by** (*intro EM-remainder-eqI*) *simp-all*
qed

lemma *EM-remainder′-combine*:
fixes *f* :: *real* ⇒ *′a* :: *banach*
assumes [*continuous-intros*]: *continuous-on* {*a*..*c*} *f*
assumes *a* ≤ *b b* ≤ *c*
shows *EM-remainder′ n f a b* + *EM-remainder′ n f b c* = *EM-remainder′ n f*
a c
proof −
have *integral* {*a*..*b*} (λ*t*. *pbernpoly n t* ∗*R f t*) + *integral* {*b*..*c*} (λ*t*. *pbernpoly n*
t ∗*R f t*) =
integral {*a*..*c*} (λ*t*. *pbernpoly n t* ∗*R f t*)
by (*intro Henstock-Kurzweil-Integration.integral-combine assms integrable-EM-remainder′*)
from *this* [*symmetric*] **show** *?thesis* **by** (*simp add*: *EM-remainder′-def algebra-simps*)
qed

lemma *uniformly-convergent-EM-remainder′*:
fixes *f* :: *′a* ⇒ *real* ⇒ *′b* :: {*banach*,*real-normed-algebra*}
assumes *deriv*: ⋀*y*. *a* ≤ *y* ⟹ (*G has-real-derivative g y*) (*at y within* {*a*..})
assumes *integrable*: ⋀*a′ b y*. *y* ∈ *A* ⟹ *a* ≤ *a′* ⟹ *a′* ≤ *b* ⟹
(λ*t*. *pbernpoly n t* ∗*R f y t*) *integrable-on* {*a′*..*b*}

```
  assumes conv: convergent (λy. G (real y))
  assumes bound: eventually (λx. ∀y∈A. norm (f y x) ≤ g x) at-top
  shows   uniformly-convergent-on A (λb s. EM-remainder' n (f s) a b)
proof −
  interpret bounded-linear λx::'b. ((− 1) ^ Suc n / fact n) *R x
    by (rule bounded-linear-scaleR-right)

  from bounded-pbernpoly obtain C where C: ⋀x. norm (pbernpoly n x) ≤ C by
auto
  from C[of 0] have [simp]: C ≥ 0 by simp

  show ?thesis unfolding EM-remainder'-def
  proof (intro uniformly-convergent-on uniformly-convergent-improper-integral')
    fix x assume x ≥ a
    thus ((λx. C * G x) has-real-derivative C * g x) (at x within {a..})
      by (intro DERIV-cmult deriv)
  next
    fix y a' b assume y ∈ A a ≤ a' a' ≤ b
    thus (λt. pbernpoly n t *R f y t) integrable-on {a'..b}
      by (rule integrable)
  next
    from conv obtain L where (λy. G (real y)) ⟶ L
      by (auto simp: convergent-def)
    from tendsto-mult[OF tendsto-const[of C] this]
      show convergent (λy. C * G (real y))
        by (auto simp: convergent-def)
  next
    show ∀F x in at-top. ∀y∈A. norm (pbernpoly n x *R f y x) ≤ C * g x
      using C unfolding norm-scaleR
      by (intro eventually-mono[OF bound] ballI mult-mono) auto
  qed
qed

lemma uniform-limit-EM-remainder:
  fixes f :: 'a ⇒ real ⇒ 'b :: {banach,real-normed-algebra}
  assumes deriv: ⋀y. a ≤ y ⟹ (G has-real-derivative g y) (at y within {a..})
  assumes integrable: ⋀a' b y. y ∈ A ⟹  a ≤ a' ⟹ a' ≤ b ⟹
                       (λt. pbernpoly n t *R f y t) integrable-on {a'..b}
  assumes conv: convergent (λy. G (real y))
  assumes bound: eventually (λx. ∀y∈A. norm (f y x) ≤ g x) at-top
  shows   uniform-limit A (λb s. EM-remainder' n (f s) a b)
          (λs. EM-remainder n (f s) a) sequentially
proof −
  have *: uniformly-convergent-on A (λb s. EM-remainder' n (f s) a b)
    by (rule uniformly-convergent-EM-remainder'[OF assms])
  also have ?this ⟷ ?thesis
    unfolding uniformly-convergent-uniform-limit-iff
  proof (intro uniform-limit-cong refl always-eventually allI ballI)
    fix s assume s ∈ A
```

```
    with ∗ have ∗∗: convergent (λb. EM-remainder′ n (f s) a b)
      by (rule uniformly-convergent-imp-convergent)
    show lim (λb. EM-remainder′ n (f s) a b) = EM-remainder n (f s) a
    proof (rule sym, rule EM-remainder-eqI)
      have ((λx. EM-remainder′ n (f s) (real-of-int a) (real x)) ⟶
              lim (λx. EM-remainder′ n (f s) (real-of-int a) (real x))) at-top
        (is (- ⟶ ?L) -) using ∗∗ unfolding convergent-LIMSEQ-iff by blast
      hence ((λx. EM-remainder′ n (f s) (real-of-int a) (real (nat x))) ⟶ ?L)
at-top
        by (rule filterlim-compose) (fact filterlim-nat-sequentially)
      thus ((λx. EM-remainder′ n (f s) (real-of-int a) (real-of-int x)) ⟶ ?L)
at-top
        by (rule Lim-transform-eventually)
          (auto intro: eventually-mono[OF eventually-ge-at-top[of 0]])
    qed
  qed
  finally show ... .
qed

lemma tendsto-EM-remainder:
  fixes f :: real ⇒ 'b :: {banach,real-normed-algebra}
  assumes deriv: ⋀y. a ≤ y ⟹ (G has-real-derivative g y) (at y within {a..})
  assumes integrable: ⋀a′ b . a ≤ a′ ⟹ a′ ≤ b ⟹
                      (λt. pbernpoly n t ∗R f t) integrable-on {a′..b}
  assumes conv: convergent (λy. G (real y))
  assumes bound: eventually (λx. norm (f x) ≤ g x) at-top
  shows   filterlim (λb. EM-remainder′ n f a b)
          (nhds (EM-remainder n f a)) sequentially
proof −
  have uniform-limit {()} (λb s. EM-remainder′ n f a b)
          (λs. EM-remainder n f a) sequentially
    using assms by (intro uniform-limit-EM-remainder[where G = G and g =
g]) auto
  moreover have () ∈ {()} by simp
  ultimately show ?thesis by (rule tendsto-uniform-limitI)
qed

lemma EM-remainder-0 [simp]: EM-remainder n (λx. 0) a = 0
  by (rule EM-remainder-eqI) (simp add: EM-remainder′-def)

lemma holomorphic-EM-remainder′:
  assumes deriv:
    ⋀z t. z ∈ U ⟹ t ∈ {a..x} ⟹
      ((λz. f z t) has-field-derivative f′ z t) (at z within U)
  assumes int: ⋀b c z e. a ≤ b ⟹ c ≤ x ⟹ z ∈ U ⟹
                (λt. of-real (bernpoly n (t − e)) ∗ f z t) integrable-on {b..c}
  assumes cont: continuous-on (U × {a..x}) (λ(z, t). f′ z t)
  assumes convex U
  shows (λs. EM-remainder′ n (f s) a x) holomorphic-on U
```

unfolding *EM-remainder'-def scaleR-conv-of-real*
proof (*intro holomorphic-intros*)
have *holo*: (λ*z*. *integral* (*cbox b c*) (λ*t*. *of-real* (*bernpoly n* (*t* − *e*)) ∗ *f z t*)) *holomorphic-on U*
if *b* ≥ *a c* ≤ *x* **for** *b c e* :: *real*
proof (*rule leibniz-rule-holomorphic*)
fix *z t* **assume** *z* ∈ *U t* ∈ *cbox b c*
thus ((λ*z*. *complex-of-real* (*bernpoly n* (*t* − *e*)) ∗ *f z t*) *has-field-derivative*
complex-of-real (*bernpoly n* (*t* − *e*)) ∗ *f' z t*) (*at z within U*)
using *that* **by** (*intro DERIV-cmult deriv*) *auto*
next
fix *z* **assume** *z* ∈ *U*
thus (λ*t*. *complex-of-real* (*bernpoly n* (*t* − *e*)) ∗ *f z t*) *integrable-on cbox b c*
using *that int*[*of b c z*] **by** *auto*
next
have *continuous-on* (*U* × {*b*..*c*}) (λ(*z*, *t*). *f' z t*)
using *cont* **by** (*rule continuous-on-subset*) (*insert that*, *auto*)
thus *continuous-on* (*U* × *cbox b c*) (λ(*z*, *t*).
complex-of-real (*bernpoly n* (*t* − *e*)) ∗ *f' z t*)
by (*auto simp*: *case-prod-unfold intro*!: *continuous-intros*)
qed *fact*+

consider *a* > *x* | *a* ≤ *x floor x* ≤ *a* | *a* ≤ *x floor x* > *a* **by** *force*
hence (λ*z*. *integral* (*cbox a x*) (λ*t*. *of-real* (*pbernpoly n t*) ∗ *f z t*)) *holomorphic-on U*
(**is** *?f a x holomorphic-on* -)
proof *cases*
case *2*
have (λ*z*. *integral* (*cbox a x*) (λ*t*. *of-real* (*bernpoly n* (*t* − *of-int* ⌊*x*⌋)) ∗ *f z t*))
holomorphic-on U
by (*intro holo*) *auto*
also have (λ*z*. *integral* (*cbox a x*) (λ*t*. *of-real* (*bernpoly n* (*t* − *of-int* ⌊*x*⌋)) ∗ *f z t*)) = *?f a x*
proof (*intro ext integral-cong*, *goal-cases*)
case (*1 z t*)
hence *t* ≥ *a t* ≤ *x* **by** *auto*
hence *floor t* = *floor x* **using** *2* **by** *linarith*
thus *?case* **by** (*simp add*: *pbernpoly-def frac-def*)
qed
finally show *?thesis* **.**
next
case *3*
define *N* :: *int set* **where** *N* = {⌈*a*⌉..<⌊*x*⌋}
define *A* **where** *A* = *insert* {*a*..*of-int* ⌈*a*⌉} (*insert* {*of-int* ⌊*x*⌋..*x*}
((λ*n*. {*of-int n*..*of-int n* + *1*}) ' *N*))
{
fix *X* **assume** *X* ∈ *A*
then consider *X* = {*a*..*of-int* ⌈*a*⌉} | *X* = {*of-int* ⌊*x*⌋..*x*} |
n **where** *X* = {*of-int n*..*of-int n* + *1*} *n* ∈ *N* **by** (*auto simp*: *A-def*)

```
  } note A-cases = this

  have division: A division-of {a..x}
  proof (rule division-ofI)
    show finite A by (auto simp: A-def N-def)
    fix K assume K: K ∈ A
    from 3 have of-int ⌈a⌉ ≤ x
      using ceiling-le[of a floor x] by linarith
    moreover from 3 have of-int ⌊x⌋ ≥ a by linarith
      ultimately show K ⊆ {a..x} using K 3 by (auto simp: A-def N-def)
linarith+
    from K show K ≠ {} and ∃ a b. K = cbox a b by (auto simp: A-def)
  next
    fix K1 K2 assume K: K1 ∈ A K2 ∈ A K1 ≠ K2
     have F1: interior {a..⌈a⌉} ∩ interior {⌊x⌋..x} = {} using 3 ceiling-le[of a
floor x]
      by (auto simp: min-def max-def)
    hence F2: interior {⌊x⌋..x} ∩ interior {a..⌈a⌉} = {} by simp
    have F3: interior {a..⌈a⌉} ∩ interior {of-int n..of-int n+1} = {}
          interior {⌊x⌋..x} ∩ interior {of-int n..of-int n+1} = {}
          interior {of-int n..of-int n+1} ∩ interior {a..⌈a⌉} = {}
         interior {of-int n..of-int n+1} ∩ interior {⌊x⌋..x} = {}if n ∈ N for n
      using 3 ceiling-le[of a floor x] that by (auto simp: min-def max-def N-def)
      have F4: interior {real-of-int n..of-int n+1} ∩ interior {of-int m..of-int
m+1} = {}
      if {real-of-int n..of-int n+1} ≠ {of-int m..of-int m+1} for m n
    proof −
      from that have n ≠ m by auto
      thus ?thesis by simp
    qed
    from F1 F2 F3 F4 K show interior K1 ∩ interior K2 = {}
      by (elim A-cases) (simp-all only: not-False-eq-True)
  next
    show ⋃ A = {a..x}
    proof (cases ⌈a⌉ = ⌊x⌋)
      case True
    thus ?thesis using 3 by (auto simp: A-def N-def intro: order.trans) linarith+
    next
      case False
      with 3 have ∗: ⌈a⌉ < ⌊x⌋ by linarith
     have ⋃ A = {a..of-int ⌈a⌉} ∪ (⋃ n∈N. {of-int n..of-int (n + 1)}) ∪ {of-int
⌊x⌋..x}
        by (simp add: A-def Un-ac)
     also have (⋃ n∈N. {of-int n..of-int (n + 1)}) = {of-int ⌈a⌉..real-of-int ⌊x⌋}
        using ∗ unfolding N-def by (intro Union-atLeastAtMost-real-of-int)
      also have {a..of-int ⌈a⌉} ∪ ... = {a..real-of-int ⌊x⌋}
        using 3 ∗ by (intro ivl-disj-un) auto
      also have ... ∪ {of-int ⌊x⌋..x} = {a..x}
        using 3 ∗ by (intro ivl-disj-un) auto
```

```
      finally show ?thesis .
    qed
  qed


  have (λz. ∑X∈A. integral X (λt. of-real (bernpoly n (t − ⌊Inf X⌋)) ∗ f z t))
       holomorphic-on U
  proof (intro holomorphic-on-sum holo, goal-cases)
    case (1 X)
   from 1 and division have subset: X ⊆ {a..x} by (auto simp: division-of-def)
    from 1 obtain b c where [simp]: X = cbox b c b ≤ c by (auto simp: A-def)
    from subset have b ≥ a c ≤ x by auto
    hence (λx. integral (cbox b c) (λt. of-real (bernpoly n (t − ⌊Inf {b..c}⌋)) ∗ f
x t))
          holomorphic-on U by (intro holo) auto
    thus ?case by simp
  qed
  also have ?this ⟷ (λz. integral {a..x} (λt. of-real (pbernpoly n t) ∗ f z t))
                    holomorphic-on U
  proof (intro holomorphic-cong refl, goal-cases)
    case (1 z)
    have ((λt. of-real (pbernpoly n t) ∗ f z t) has-integral
          (∑X∈A. integral X (λt. of-real (bernpoly n (t − ⌊Inf X⌋)) ∗ f z t)))
{a..x}
      using division
    proof (rule has-integral-combine-division)
      fix X assume X: X ∈ A
      then obtain b c where X': X = {b..c} b ≤ c by (elim A-cases) auto
      from X and division have X ⊆ {a..x} by (auto simp: division-of-def)
      with X' have bc: b ≥ a c ≤ x by auto
      have ((λt. of-real (bernpoly n (t − of-int ⌊Inf X⌋)) ∗ f z t) has-integral
            integral X (λt. of-real (bernpoly n (t − of-int ⌊Inf X⌋)) ∗ f z t)) X
        unfolding X' using ‹z ∈ U› bc by (intro integrable-integral int)
      also have ?this ⟷ ((λt. of-real (pbernpoly n t) ∗ f z t) has-integral
            integral X (λt. of-real (bernpoly n (t − of-int ⌊Inf X⌋)) ∗ f z t)) X
      proof (rule has-integral-spike-eq[of {Sup X}], goal-cases)
        case (2 t)
        note t = this
        from ‹X ∈ A› have ⌊t⌋ = ⌊Inf X⌋
        proof (cases rule: A-cases [consumes 1])
          case 1
          with t show ?thesis
              by (intro floor-unique) (auto simp: ceiling-altdef split: if-splits,
(linarith+)?)
        next
          case 2
          with t show ?thesis
              by (intro floor-unique) (auto simp: ceiling-altdef split: if-splits,
(linarith+)?)
```

```
        next
          case 3
          with t show ?thesis
            by (intro floor-unique) (auto simp: ceiling-altdef N-def split: if-splits)
        qed
        thus ?case by (simp add: pbernpoly-def frac-def)
      qed auto
      finally show ... .
    qed
    thus ?case by (simp add: has-integral-iff)
  qed
  finally show ?thesis by simp
 qed auto
 thus (λz. integral {a..x} (λt. of-real (pbernpoly n t) * f z t)) holomorphic-on U
  by simp
qed

lemma
 assumes deriv: ⋀y. a ≤ y ⟹ (G has-real-derivative g y) (at y within {a..})
 assumes deriv':
  ⋀z t x. z ∈ U ⟹ x ≥ a ⟹ t ∈ {a..x} ⟹
    ((λz. f z t) has-field-derivative f' z t) (at z within U)
 assumes cont: continuous-on (U × {of-int a..}) (λ(z, t). f' z t)
 assumes int: ⋀b c z e. a ≤ b ⟹ z ∈ U ⟹
          (λt. of-real (bernpoly n (t − e)) * f z t) integrable-on {b..c}
 assumes int': ⋀a' b y. y ∈ U ⟹ a ≤ a' ⟹ a' ≤ b ⟹
                  (λt. pbernpoly n t *R f y t) integrable-on {a'..b}
 assumes conv: convergent (λy. G (real y))
 assumes bound: eventually (λx. ∀ y∈U. norm (f y x) ≤ g x) at-top
 assumes open U
  shows analytic-EM-remainder: (λs::complex. EM-remainder n (f s) a) analytic-on U
    and holomorphic-EM-remainder: (λs::complex. EM-remainder n (f s) a) holomorphic-on U
proof −
 show (λs::complex. EM-remainder n (f s) a) analytic-on U
 unfolding analytic-on-def
 proof
  fix z assume z ∈ U
  from ‹z ∈ U› and ‹open U› obtain ε where ε: ε > 0 ball z ε ⊆ U
   by (auto simp: open-contains-ball)
  have (λs. EM-remainder n (f s) a) holomorphic-on ball z ε
  proof (rule holomorphic-uniform-sequence)
   fix x :: nat
   show (λs. EM-remainder' n (f s) a x) holomorphic-on ball z ε
   proof (rule holomorphic-EM-remainder', goal-cases)
    fix s t assume s ∈ ball z ε t ∈ {real-of-int a..real x}
    thus ((λz. f z t) has-field-derivative f' s t) (at s within ball z ε)
     using ε by (intro DERIV-subset[OF deriv'[of - x]]) auto
```

```
    next
      case (2 b c s e)
      with ε have s ∈ U by blast
      with 2 show ?case using ε int[of b s e c] by (cases a ≤ x) auto
    next
      from cont show continuous-on (ball z ε × {real-of-int a..real x}) (λ(z, t).
f' z t)
        by (rule continuous-on-subset) (insert ε, auto)
    qed (auto)
  next
    fix s assume s: s ∈ ball z ε
    have open (ball z ε) by simp
    with s obtain δ where δ: δ > 0 cball s δ ⊆ ball z ε
      unfolding open-contains-cball by blast
    moreover have bound': eventually (λx. ∀ y∈cball s δ. norm (f y x) ≤ g x)
at-top
      by (intro eventually-mono [OF bound]) (insert δ ε, auto)
    have uniform-limit (cball s δ) (λx s. EM-remainder' n (f s) (real-of-int a)
(real x))
                       (λs. EM-remainder n (f s) a) sequentially
      by (rule uniform-limit-EM-remainder[OF deriv int' conv bound']) (insert δ
ε s, auto)
    ultimately show ∃δ>0. cball s δ ⊆ ball z ε ∧ uniform-limit (cball s δ)
                        (λx s. EM-remainder' n (f s) (real-of-int a) (real x))
                        (λs. EM-remainder n (f s) a) sequentially by blast
  qed auto
  with ε show ∃ε>0. (λs. EM-remainder n (f s) a) holomorphic-on ball z ε
    by blast
 qed
 thus (λs::complex. EM-remainder n (f s) a) holomorphic-on U
  by (rule analytic-imp-holomorphic)
qed
```

The following lemma is the first step in the proof of the Euler–MacLaurin formula: We show the relationship between the first-order remainder term and the difference of the integral and the sum.

```
context
  fixes f f' :: real ⇒ 'a :: banach
  fixes a b :: int and I S :: 'a
  fixes Y :: real set
  assumes a ≤ b
  assumes cntbl: countable Y
  assumes cont: continuous-on {real-of-int a..real-of-int b} f
  assumes deriv [derivative-intros]:
          ⋀x::real. x ∈ {a..b} − Y ⟹ (f has-vector-derivative f' x) (at x)
  defines S-def: S ≡ (∑ i∈{a<..b}. f i) and I-def: I ≡ integral {a..b} f
begin

lemma
```

```
  diff-sum-integral-has-integral-int:
    ((λt. (frac t − 1/2) *R f′ t) has-integral (S − I − (f b − f a) /R 2)) {a..b}
proof (cases a = b)
  case True
  thus ?thesis by (simp-all add: S-def I-def has-integral-refl)
next
  case False
  with ‹a ≤ b› have ab: a < b by simp
  let ?A = (λn. {real-of-int n..real-of-int (n+1)}) ‘ {a..<b}
  have division: ?A division-of {of-int a..of-int b}
    using Union-atLeastAtMost-real-of-int[OF ab] by (simp add: division-of-def)
  have cont′ [continuous-intros]: continuous-on A f if A ⊆ {of-int a..of-int b} for
A
    using continuous-on-subset[OF cont that] .

  define d where d = (λx. (f x + f (x + 1)) /R 2 − integral {x..x+1} f)
  have ((λt. (frac t − 1/2) *R f′ t) has-integral d i) {of-int i..of-int (i+1)}
    if i: i ∈ {a..<b} for i
  proof (rule has-integral-spike)
    show (frac x − 1 / 2) *R f′ x = (x − of-int i − 1 / 2) *R f′ x
        if x ∈ {of-int i..of-int (i + 1)} − {of-int (i + 1)} for x
    proof −
      have x ≥ of-int i x < of-int (i + 1) using that by auto
      hence floor x = of-int i by (subst floor-unique) auto
      thus ?thesis by (simp add: frac-def)
    qed
  next
    define h where h = (λx::real. (x − of-int i − 1 / 2) *R f′ x)
    define g where g = (λx::real. (x − of-int i − 1/2) *R f x − integral {of-int
i..x} f)
    have *: ((λx. integral {real-of-int i..x} f) has-vector-derivative f x) (at x within
{i..i+1})
      if x ∈ {of-int i<..<of-int i + 1} for x using that i
      by (intro integral-has-vector-derivative cont′) auto
    have ((λx. integral {real-of-int i..x} f) has-vector-derivative f x) (at x)
      if x ∈ {of-int i<..<of-int i + 1} for x
      using that i at-within-interior[of x {of-int i..of-int (i + 1)}] *[of x] by simp
    hence (h has-integral g (of-int (i + 1)) − g (of-int i)) {of-int i..of-int (i+1)}
      unfolding g-def h-def using that
      by (intro fundamental-theorem-of-calculus-interior-strong[OF cntbl])
       (auto intro!: derivative-eq-intros continuous-intros indefinite-integral-continuous-1
            integrable-continuous-real)
    also have g (of-int (i + 1)) − g (of-int i) = d i
      by (simp add: g-def scaleR-add-right [symmetric] d-def)
    finally show (h has-integral d i) {of-int i..of-int (i + 1)} .
  qed simp-all
  hence *: ⋀I. I∈?A ⟹ ((λx. (frac x − 1 / 2) *R f′ x) has-integral d (⌊Inf I⌋))
I
    by (auto simp: add-ac)
```

```
  have ((λx::real. (frac x − 1 / 2) ∗R f′ x) has-integral (∑ I∈?A. d (⌊Inf I⌋)))
(⋃ ?A)
   by (intro has-integral-Union ∗ finite-imageI) (force intro!: negligible-atLeastAtMostI
pairwiseI)+
  also have ⋃ ?A = {of-int a..of-int b}
    by (intro Union-atLeastAtMost-real-of-int ab)
  also have (∑ I∈?A. d (⌊Inf I⌋)) = (∑ i=a..<b. d i)
    by (subst sum.reindex) (auto simp: inj-on-def)
  also have ... = (1 / 2) ∗R ((∑ i = a..<b. f (real-of-int i)) +
                  (∑ i = a..<b. f (real-of-int (i + 1)))) −
                 (∑ i = a..<b. integral {real-of-int i..1 + real-of-int i} f)
    (is - = - ∗R (?S1 + ?S2) − ?S3)
    by (simp add: d-def algebra-simps sum.distrib sum-subtractf scaleR-sum-right)
  also have ?S1 = (∑ i = a..b. f (real-of-int i)) − f b
    unfolding S-def using ab by (subst sum-atLeastAtMost-int-last) auto
  also have (∑ i = a..b. f (real-of-int i)) = S + f a
    unfolding S-def using ab by (subst sum-atLeastAtMost-int-head) auto
  also have ?S2 = S unfolding S-def
    by (intro sum.reindex-bij-witness[of - λi. i−1 λi. i+1]) auto
  also have (1 / 2) ∗R (S + f a − f b + S) =
              (1/2) ∗R S + (1/2) ∗R S − (f b − f a) /R 2
    by (simp add: algebra-simps)
   also have (1/2) ∗R S + (1/2) ∗R S = S by (simp add: scaleR-add-right
[symmetric])

  also have ?S3 = (∑ I∈?A. integral I f)
    by (subst sum.reindex) (auto simp: inj-on-def add-ac)
  also have ... = I unfolding I-def
   by (intro integral-combine-division-topdown [symmetric] division integrable-continuous-real

            continuous-intros) simp-all
  finally show ?thesis by (simp add: algebra-simps)
qed

lemma diff-sum-integral-has-integral-int′:
  ((λt. pbernpoly 1 t ∗R f′ t) has-integral (S − I − (f b − f a) /R 2 )) {a..b}
  using diff-sum-integral-has-integral-int by (simp add: pbernpoly-def bernpoly-def)

lemma EM-remainder′-Suc-0: EM-remainder′ (Suc 0) f′ a b = S − I − (f b − f
a) /R 2
  using diff-sum-integral-has-integral-int′ by (simp add: has-integral-iff EM-remainder′-def)

end
```

Next, we show that the n-th-order remainder can be expressed in terms of the $n + 1$-th-order remainder term. Iterating this essentially yields the Euler–MacLaurin formula.

```
context
  fixes f f′ :: real ⇒ 'a :: banach and a b :: int and n :: nat and A :: real set
```

assumes *ab*: $a \leq b$ **and** *n*: $n > 0$
assumes *cntbl*: *countable A*
assumes *cont*: *continuous-on* $\{$*of-int a..of-int b*$\}$ *f*
assumes *cont'*: *continuous-on* $\{$*of-int a..of-int b*$\}$ *f'*
assumes *deriv*: $\bigwedge x.\ x \in \{$*of-int a<..<of-int b*$\} - A \Longrightarrow$ (*f has-vector-derivative f' x*) (*at x*)
begin

lemma *EM-remainder'-integral-conv-Suc*:
shows *integral* $\{a..b\}$ ($\lambda t.$ *pbernpoly n t* $*_R$ *f t*) =
(*bernoulli* (*Suc n*) / *real* (*Suc n*)) $*_R$ (*f b* − *f a*) −
integral $\{a..b\}$ ($\lambda t.$ *pbernpoly* (*Suc n*) *t* $*_R$ *f' t*) $/_R$ *real* (*Suc n*)
unfolding *EM-remainder'-def*
proof −
let *?h* = $\lambda i.$ (*pbernpoly* (*Suc n*) (*real-of-int i*) / *real* (*Suc n*)) $*_R$ *f* (*real-of-int i*)
define *T* **where** *T* = *integral* $\{a..b\}$ ($\lambda t.$ (*pbernpoly* (*Suc n*) *t* / *real* (*Suc n*)) $*_R$ *f' t*)
note [*derivative-intros*] = *has-field-derivative-pbernpoly-Suc'*
let *?A* = *real-of-int* ‘ $\{a..b\} \cup A$

have (($\lambda t.$ *pbernpoly n t* $*_R$ *f t*) *has-integral* (−*T* + (*?h b* − *?h a*))) $\{a..b\}$
proof (*rule integration-by-parts-interior-strong*[*OF bounded-bilinear-scaleR*])
from *cntbl* **show** *countable ?A* **by** *simp*
from ‹*n* > *0*› **show** *continuous-on* $\{$*of-int a..of-int b*$\}$ ($\lambda t.$ *pbernpoly* (*Suc n*) *t* / *real* (*Suc n*))
by (*intro continuous-intros*) *auto*
show *continuous-on* $\{$*of-int a..of-int b*$\}$ *f* **by** *fact*
show (*f has-vector-derivative f' t*) (*at t*) **if** *t* ∈ $\{$*of-int a<..<of-int b*$\} - ?A$ **for** *t*
using *deriv*[*of t*] *that* **by** *auto*
have ($\lambda t.$ *pbernpoly* (*Suc n*) *t* $*_R$ *f' t*) *integrable-on* $\{a..b\}$
by (*intro integrable-EM-remainder' cont'*)
hence ($\lambda t.$ (*1* / *real* (*Suc n*)) $*_R$ *pbernpoly* (*Suc n*) *t* $*_R$ *f' t*) *integrable-on* $\{a..b\}$
by (*rule integrable-cmul*)
also have ($\lambda t.$ (*1* / *real* (*Suc n*)) $*_R$ *pbernpoly* (*Suc n*) *t* $*_R$ *f' t*) =
($\lambda t.$ (*pbernpoly* (*Suc n*) *t* / *real* (*Suc n*)) $*_R$ *f' t*)
by (*rule ext*) (*simp add*: *algebra-simps*)
finally show (($\lambda t.$ (*pbernpoly* (*Suc n*) *t* / *real* (*Suc n*)) $*_R$ *f' t*)
has-integral ?h b − *?h a* − (− *T* + (*?h b* − *?h a*))) $\{a..b\}$
using *integrable-EM-remainder'*[*of a b f' Suc n*]
by (*simp add*: *has-integral-iff T-def*)
qed (*insert ab n*, *auto intro*!: *derivative-eq-intros*
simp: *has-real-derivative-iff-has-vector-derivative* [*symmetric*] *not-le elim*!: *Ints-cases*)
also have *?h b* − *?h a* = (*bernoulli* (*Suc n*) / *real* (*Suc n*)) $*_R$ (*f b* − *f a*)
using *n* **by** (*simp add*: *algebra-simps bernoulli'-def*)
finally have *integral* $\{a..b\}$ ($\lambda t.$ *pbernpoly n t* $*_R$ *f t*) = ... − *T*
by (*simp add*: *has-integral-iff*)

also have T = *integral* $\{a..b\}$ $(\lambda t.\ (1\ /\ real\ (Suc\ n))\ *_R\ (pbernpoly\ (Suc\ n)\ t)\ *_R\ f'\ t)$
by (*simp add*: *T-def*)
also have $\ldots$ = *integral* $\{a..b\}$ $(\lambda t.\ pbernpoly\ (Suc\ n)\ t\ *_R\ f'\ t)\ /_R\ real\ (Suc\ n)$
by (*subst integral-cmul*) (*simp-all add*: *divide-simps*)
finally show *?thesis* **.**
qed

lemma *EM-remainder'-conv-Suc*:
EM-remainder' *n f a b* =
$((-1)\ \hat{}\ Suc\ n\ *\ bernoulli\ (Suc\ n)\ /\ fact\ (Suc\ n))\ *_R\ (f\ b - f\ a)\ +$
EM-remainder' $(Suc\ n)\ f'\ a\ b$
by (*simp add*: *EM-remainder'-def EM-remainder'-integral-conv-Suc scaleR-diff-right scaleR-add-right field-simps del*: *of-nat-Suc*)

end

context
fixes *f f'* :: *real* $\Rightarrow$ *'a* :: *banach* **and** *a* :: *int* **and** *n* :: *nat* **and** *A* :: *real set* **and** *C*
assumes *n*: $n > 0$
assumes *cntbl*: *countable A*
assumes *cont*: *continuous-on* $\{of\text{-}int\ a..\}$ *f*
assumes *cont'*: *continuous-on* $\{of\text{-}int\ a..\}$ *f'*
assumes *lim*: $(f \longrightarrow C)$ *at-top*
assumes *deriv*: $\bigwedge x.\ x \in \{of\text{-}int\ a<..\} - A \Longrightarrow (f\ has\text{-}vector\text{-}derivative\ f'\ x)\ (at\ x)$
begin

lemma
shows *EM-remainder-converges-iff-Suc-converges*:
EM-remainder-converges $n\ f\ a \longleftrightarrow$ *EM-remainder-converges* $(Suc\ n)\ f'\ a$
and *EM-remainder-conv-Suc*:
EM-remainder-converges $n\ f\ a \Longrightarrow$
EM-remainder $n\ f\ a$ =
$((-1)\ \hat{}\ Suc\ n\ *\ bernoulli\ (Suc\ n)\ /\ fact\ (Suc\ n))\ *_R\ (C - f\ a)\ +$
EM-remainder $(Suc\ n)\ f'\ a$
proof (*rule iffI*)
define *g* **where** $g = (\lambda x.\ ((-1)\ \hat{}\ Suc\ n\ *\ bernoulli\ (Suc\ n)\ /\ fact\ (Suc\ n))\ *_R\ (f\ x - f\ a))$
define *G* **where** $G = ((-1)\ \hat{}\ Suc\ n\ *\ bernoulli\ (Suc\ n)\ /\ fact\ (Suc\ n))\ *_R\ (C - f\ a)$
have *limit-g*: $(g \longrightarrow G)$ *at-top* **unfolding** *g-def G-def* **by** (*intro tendsto-intros lim*)
have $*$: *eventually* $(\lambda x.$ *EM-remainder'* $n\ f$ $(real\text{-}of\text{-}int\ a)$ $(real\text{-}of\text{-}int\ x) =$
$g\ x +$ *EM-remainder'* $(Suc\ n)\ f'$ $(real\text{-}of\text{-}int\ a)$ $(real\text{-}of\text{-}int\ x))$ *at-top*
using *eventually-ge-at-top*[*of a*]

```
 proof eventually-elim
   case (elim b)
   thus ?case
   using EM-remainder'-conv-Suc[OF elim n cntbl continuous-on-subset[OF cont]

          continuous-on-subset[OF cont'] deriv] by (auto simp: g-def)
 qed

 {
   assume EM-remainder-converges n f a
   then obtain L
     where L: ((λb. EM-remainder' n f (real-of-int a) (real-of-int b)) ⟶ L)
at-top
     by (auto simp: EM-remainder-converges-def)
   have ∗: ((λb. EM-remainder' (Suc n) f' (real-of-int a) (real-of-int b)) ⟶ L
− G) at-top
   proof (rule Lim-transform-eventually)
     show ∀F x in at-top. EM-remainder' n f (real-of-int a) (real-of-int x) − g x
=
            EM-remainder' (Suc n) f' (real-of-int a) (real-of-int x)
       using ∗ by (simp add: algebra-simps)
     show ((λx. EM-remainder' n f (real-of-int a) (real-of-int x) − g x) ⟶ L
− G) at-top
       by (intro tendsto-intros filterlim-compose[OF limit-g] L)
   qed
  from ∗ show EM-remainder-converges (Suc n) f' a unfolding EM-remainder-converges-def
..
   from ∗ have EM-remainder (Suc n) f' a = L − G by (rule EM-remainder-eqI)
   moreover from L have EM-remainder n f a = L by (rule EM-remainder-eqI)
   ultimately show EM-remainder n f a = G + EM-remainder (Suc n) f' a by
(simp add: G-def)
 }
 {
   assume EM-remainder-converges (Suc n) f' a
   then obtain L
     where L: ((λb. EM-remainder' (Suc n) f' (real-of-int a) (real-of-int b)) ⟶
L) at-top
     by (auto simp: EM-remainder-converges-def)
   have ∗: ((λb. EM-remainder' n f (real-of-int a) (real-of-int b)) ⟶ G + L)
at-top
   proof (rule Lim-transform-eventually)
    show ∀F x in at-top. g x + EM-remainder' (Suc n) f' (real-of-int a) (real-of-int
x) =
            EM-remainder' n f (real-of-int a) (real-of-int x)
       using ∗ by (subst eq-commute)
     show ((λx. g x + EM-remainder' (Suc n) f' (real-of-int a) (real-of-int x))
⟶ G + L) at-top
       by (intro tendsto-intros filterlim-compose[OF limit-g] L)
   qed
```

thus *EM-remainder-converges n f a* **unfolding** *EM-remainder-converges-def* **..**
}
qed

end

1.3 The conventional version of the Euler–MacLaurin formula

The following theorems are the classic Euler–MacLaurin formula that can be found, with slight variations, in many sources (e. g. [1, 2, 3]).

context
fixes *f* :: *real* ⇒ *'a* :: *banach*
fixes *fs* :: *nat* ⇒ *real* ⇒ *'a*
fixes *a b* :: *int* **assumes** *ab*: *a* ≤ *b*
fixes *N* :: *nat* **assumes** *N*: *N* > *0*
fixes *Y* :: *real set* **assumes** *cntbl*: *countable Y*
assumes *fs-0* [*simp*]: *fs 0* = *f*
assumes *fs-cont* [*continuous-intros*]:
⋀*k*. *k* ≤ *N* ⟹ *continuous-on* {*real-of-int a..real-of-int b*} (*fs k*)
assumes *fs-deriv* [*derivative-intros*]:
⋀*k x*. *k* < *N* ⟹ *x* ∈ {*a..b*} − *Y* ⟹ (*fs k has-vector-derivative fs* (*Suc k*) *x*) (*at x*)
begin

theorem *euler-maclaurin-raw-strong-int*:
defines *S* ≡ (∑ *i*∈{*a*<..*b*}. *f* (*of-int i*))
defines *I* ≡ *integral* {*of-int a..of-int b*} *f*
defines *c'* ≡ λ*k*. (*bernoulli'* (*Suc k*) / *fact* (*Suc k*)) *$*_R$* (*fs k b* − *fs k a*)
shows *S* − *I* = (∑ *k*<*N*. *c' k*) + *EM-remainder' N* (*fs N*) *a b*
proof −
define *c* :: *nat* ⇒ *'a*
where *c* = (λ*k*. ((−*1*) ^ (*Suc k*) ∗ *bernoulli* (*Suc k*) / *fact* (*Suc k*)) *$*_R$* (*fs k b* − *fs k a*))
have *S* − *I* = (∑ *k*<*m*. *c k*) + *EM-remainder' m* (*fs m*) *a b* **if** *m* ≥ *1 m* ≤ *N* **for** *m*
using *that*
proof (*induction m rule*: *dec-induct*)
case *base*
with *ab cntbl fs-cont*[*of 0*] **show** *?case* **using** *fs-deriv*[*of 0*] *N* **unfolding** *One-nat-def*
by (*subst EM-remainder'-Suc-0*[*of - - Y f*]) (*simp-all add*: *algebra-simps S-def I-def c-def*)
next
case (*step n*)
from *step.prems* **have** *S* − *I* = (∑ *k*<*n*. *c k*) + *EM-remainder' n* (*fs n*) *a b*
by (*intro step.IH*) *simp-all*
also have (∑ *k*<*n*. *c k*) = (∑ *k*<*Suc n*. *c k*) +

```
        (((−1) ^ n * bernoulli (Suc n) / fact (Suc n)) *R (fs n b − fs n a))
    (is - = - + ?c) by (simp add: EM-remainder'-Suc-0 c-def)
  also have ... + EM-remainder' n (fs n) a b = (∑k<Suc n. c k) + (?c +
EM-remainder' n (fs n) a b)
    by (simp add: add.assoc)
  also from step.prems step.hyps ab cntbl
   have ?c + EM-remainder' n (fs n) a b = EM-remainder' (Suc n) (fs (Suc
n)) a b
    by (subst EM-remainder'-conv-Suc [where A = Y])
      (auto intro!: fs-deriv fs-cont)
  finally show ?case .
 qed
 from this[of N] and N
   have S − I = sum c {..<N} + EM-remainder' N (fs N) (real-of-int a)
(real-of-int b) by simp
 also have sum c {..<N} = sum c' {..<N}
 proof (intro sum.cong refl)
  fix k :: nat
  show c k = c' k
   by (cases even k)
     (auto simp: c-def c'-def bernoulli'-def algebra-simps bernoulli-odd-eq-0)
 qed
 finally show ?thesis .
qed

end

theorem euler-maclaurin-strong-raw-nat:
 assumes a ≤ b 0 < N countable Y fs 0 = f
  (⋀k. k ≤ N ⟹ continuous-on {real a..real b} (fs k))
  (⋀k x. k < N ⟹ x ∈ {real a..real b} − Y ⟹
    (fs k has-vector-derivative fs (Suc k) x) (at x))
 shows (∑i∈{a<..b}. f (real i)) − integral {real a..real b} f =
       (∑k<N. (bernoulli' (Suc k) / fact (Suc k)) *R (fs k (real b) − fs k (real
a))) +
       EM-remainder' N (fs N) (real a) (real b)
proof −
 have (∑i∈{int a<..int b}. f (real-of-int i)) −
       integral {real-of-int (int a)..real-of-int (int b)} f =
       (∑k<N. (bernoulli' (Suc k) / fact (Suc k)) *R
         (fs k (real-of-int (int b)) − fs k (real-of-int (int a)))) +
       EM-remainder' N (fs N) (real-of-int (int a)) (real-of-int (int b))
   using assms by (intro euler-maclaurin-raw-strong-int[where Y = Y] assms)
simp-all
 also have (∑i∈{int a<..int b}. f (real-of-int i)) = (∑i∈{a<..b}. f (real i))
   by (intro sum.reindex-bij-witness[of - int nat]) auto
 finally show ?thesis by simp
qed
```

1.4 The "Concrete Mathematics" version of the Euler–MacLaurin formula

As explained in *Concrete Mathematics* [3], the above form of the formula has some drawbacks: When applying it to determine the asymptotics of some concrete function, one is usually left with several different unwieldy constant terms that are difficult to get rid of.

There is no general way to determine what these constant terms are, but in concrete applications, they can often be determined or estimated by other means. We can therefore simply group all the constant terms into a single constant and have the user provide a proof of what it is.

locale *euler-maclaurin-int* =
  fixes $F\ f$ :: *real* ⇒ *'a* :: *banach*
  fixes *fs* :: *nat* ⇒ *real* ⇒ *'a*
  fixes *a* :: *int*
  fixes *N* :: *nat* **assumes** *N*: $N > 0$
  fixes *C* :: *'a*
  fixes *Y* :: *real set* **assumes** *cntbl*: *countable Y*
  assumes *fs-0* [*simp*]: $fs\ 0 = f$
  assumes *fs-cont* [*continuous-intros*]:
    $\bigwedge k.\ k \leq N \Longrightarrow$ *continuous-on* {*real-of-int a..*} (*fs k*)
  assumes *fs-deriv* [*derivative-intros*]:
    $\bigwedge k\ x.\ k < N \Longrightarrow x \in \{of\text{-}int\ a..\} - Y \Longrightarrow (fs\ k$ *has-vector-derivative* $fs\ (Suc\ k)\ x)\ (at\ x)$
  assumes *F-cont* [*continuous-intros*]: *continuous-on* {*of-int a..*} *F*
  assumes *F-deriv* [*derivative-intros*]:
    $\bigwedge x.\ x \in \{of\text{-}int\ a..\} - Y \Longrightarrow (F$ *has-vector-derivative* $f\ x)\ (at\ x)$
  assumes *limit*:
    $((\lambda b.\ (\sum k{=}a..b.\ f\ k) - F\ (of\text{-}int\ b) -$
      $(\sum i{<}N.\ (bernoulli'\ (Suc\ i)\ /\ fact\ (Suc\ i))\ *_R\ fs\ i\ (of\text{-}int\ b))) \longrightarrow C)$ *at-top*
begin

context
  fixes $C'\ T$
  defines $C' \equiv -f\ a + F\ a + C + (\sum k{<}N.\ (bernoulli'\ (Suc\ k)\ /\ fact\ (Suc\ k))\ *_R\ (fs\ k\ (of\text{-}int\ a)))$
    and $T \equiv (\lambda x.\ \sum i{<}N.\ (bernoulli'\ (Suc\ i)\ /\ fact\ (Suc\ i))\ *_R\ fs\ i\ x)$
begin

lemma *euler-maclaurin-strong-int-aux*:
  assumes *ab*: $a \leq b$
  defines $S \equiv (\sum k{=}a..b.\ f\ (of\text{-}int\ k))$
  shows $S - F\ (of\text{-}int\ b) - T\ (of\text{-}int\ b) = EM\text{-}remainder'\ N\ (fs\ N)\ (of\text{-}int\ a)\ (of\text{-}int\ b) + (C - C')$
proof (*cases* $a = b$)
  case *True*
  thus *?thesis* **unfolding** *C'-def* **by** (*simp add*: *S-def EM-remainder'-def T-def*)

next
case *False*
with *assms* **have** *ab*: $a < b$ **by** *simp*
define T' **where** $T' = (\sum k{<}N.\ (bernoulli'\ (Suc\ k)\ /\ fact\ (Suc\ k))\ *_R\ (fs\ k\ (of\text{-}int\ a)))$
have $(\sum i{\in}\{a{<}..b\}.\ f\ (of\text{-}int\ i)) - integral\ \{of\text{-}int\ a..of\text{-}int\ b\}\ f =$
$(\sum k{<}N.\ (bernoulli'\ (Suc\ k)\ /\ fact\ (Suc\ k))\ *_R\ (fs\ k\ (of\text{-}int\ b) - fs\ k\ (of\text{-}int\ a))) +$
$EM\text{-}remainder'\ N\ (fs\ N)\ (of\text{-}int\ a)\ (of\text{-}int\ b)$ **using** *ab*
by (*intro euler-maclaurin-raw-strong-int* [**where** $Y = Y$] N *cntbl fs-0*
continuous-on-subset[*OF fs-cont*] *fs-deriv*) *auto*
also have $(f\ has\text{-}integral\ (F\ b - F\ a))\ \{of\text{-}int\ a..of\text{-}int\ b\}$ **using** *ab*
by (*intro fundamental-theorem-of-calculus-strong*[*OF cntbl*])
(*auto simp*: *intro*!: *continuous-on-subset*[*OF F-cont*] *has-vector-derivative-at-within derivative-intros*)
hence $integral\ \{of\text{-}int\ a..of\text{-}int\ b\}\ f = F\ (of\text{-}int\ b) - F\ (of\text{-}int\ a)$
by (*simp add*: *has-integral-iff*)
also have $(\sum k{<}N.\ (bernoulli'\ (Suc\ k)\ /\ fact\ (Suc\ k))\ *_R\ (fs\ k\ (of\text{-}int\ b) - fs\ k\ (of\text{-}int\ a))) =$
$T\ (of\text{-}int\ b) - T'$
by (*simp add*: *T-def T'-def algebra-simps sum-subtractf*)
also have $(\sum i{\in}\{a{<}..b\}.\ f\ (of\text{-}int\ i)) = S - f\ (of\text{-}int\ a)$
unfolding *S-def* **using** *ab* **by** (*subst sum-atLeastAtMost-int-head*) *auto*
finally show *?thesis* **by** (*simp add*: *algebra-simps C'-def T'-def*)
qed

lemma *EM-remainder-limit*:
assumes *ab*: $a \le b$
defines $D \equiv EM\text{-}remainder'\ N\ (fs\ N)\ (of\text{-}int\ a)\ (of\text{-}int\ b)$
shows $EM\text{-}remainder\ N\ (fs\ N)\ b = C' - D$
and *EM-remainder-converges*: $EM\text{-}remainder\text{-}converges\ N\ (fs\ N)\ b$
proof −
note *limit*
also have $((\lambda b.\ (\sum k = a..b.\ f\ (of\text{-}int\ k)) - F\ (of\text{-}int\ b) -$
$(\sum i{<}N.\ (bernoulli'\ (Suc\ i)\ /\ fact\ (Suc\ i))\ *_R\ fs\ i\ (of\text{-}int\ b))) \longrightarrow C)\ at\text{-}top =$
$((\lambda b.\ (\sum k = a..b.\ f\ (of\text{-}int\ k)) - F\ (of\text{-}int\ b) - T\ (of\text{-}int\ b)) \longrightarrow C)\ at\text{-}top$
unfolding *T-def* **..**
also have $eventually\ (\lambda x.\ (\sum k{=}a..x.\ f\ k) - F\ (of\text{-}int\ x) - T\ (of\text{-}int\ x) =$
$EM\text{-}remainder'\ N\ (fs\ N)\ (of\text{-}int\ a)\ (of\text{-}int\ x) + (C - C'))\ at\text{-}top$
(**is** $eventually\ (\lambda x.\ ?f\ x = ?g\ x)$ -)
using *eventually-gt-at-top*[*of b*]
by *eventually-elim* (*rule euler-maclaurin-strong-int-aux*, *insert ab*, *simp-all*)
hence $(?f \longrightarrow C)\ at\text{-}top \longleftrightarrow (?g \longrightarrow C)\ at\text{-}top$ **by** (*intro filterlim-cong refl*)
finally have $((\lambda x.\ ?g\ x - (C - C')) \longrightarrow (C - (C - C')))\ at\text{-}top$
by (*rule tendsto-diff*[*OF - tendsto-const*])
hence $*$: $((\lambda x.\ EM\text{-}remainder'\ N\ (fs\ N)\ (of\text{-}int\ a)\ (of\text{-}int\ x)) \longrightarrow C')\ at\text{-}top$
by *simp*

```
  have ((λx. EM-remainder' N (fs N) (of-int a) (of-int x) - D) ⟶ C' - D)
at-top
    by (intro tendsto-intros ∗)
  also have eventually (λx. EM-remainder' N (fs N) (of-int a) (of-int x) - D =
                        EM-remainder' N (fs N) (of-int b) (of-int x)) at-top
    (is eventually (λx. ?f x = ?g x) -) using eventually-ge-at-top[of b]
  proof eventually-elim
    case (elim x)
    have EM-remainder' N (fs N) (of-int a) (of-int x) =
          D + EM-remainder' N (fs N) (of-int b) (of-int x)
      using elim ab unfolding D-def
    by (intro EM-remainder'-combine [symmetric] continuous-on-subset[OF fs-cont])
auto
    thus ?case by simp
  qed
  hence (?f ⟶ C' - D) at-top ⟷ (?g ⟶ C' - D) at-top by (intro
filterlim-cong refl)
  finally have ∗: ... .
  from ∗ show EM-remainder-converges N (fs N) b unfolding EM-remainder-converges-def
..
  from ∗ show EM-remainder N (fs N) b = C' - D
    by (rule EM-remainder-eqI)
qed

theorem euler-maclaurin-strong-int:
  assumes ab: a ≤ b
  defines S ≡ (∑k=a..b. f (of-int k))
  shows S = F (of-int b) + C + T (of-int b) - EM-remainder N (fs N) b
proof -
  have S = F (of-int b) + T (of-int b) + - (C' - EM-remainder' N (fs N) (of-int
a) (of-int b)) + C
    using euler-maclaurin-strong-int-aux[OF ab] by (simp add: algebra-simps S-def)
  also have C' - EM-remainder' N (fs N) (of-int a) (of-int b) = EM-remainder
N (fs N) b
    using ab by (rule EM-remainder-limit(1) [symmetric])
  finally show ?thesis by simp
qed

end
end
```

The following version of the formula removes all the terms where the associated Bernoulli numbers vanish.

```
locale euler-maclaurin-int' =
  fixes F f :: real ⇒ 'a :: banach
  fixes fs :: nat ⇒ real ⇒ 'a
  fixes a :: int
  fixes N :: nat
```

fixes $C :: 'a$
fixes $Y :: real\ set$ **assumes** *cntbl*: $countable\ Y$
assumes *fs-0* [*simp*]: $fs\ 0 = f$
assumes *fs-cont* [*continuous-intros*]:
$\bigwedge k.\ k \leq 2*N+1 \Longrightarrow continuous\text{-}on\ \{real\text{-}of\text{-}int\ a..\}\ (fs\ k)$
assumes *fs-deriv* [*derivative-intros*]:
$\bigwedge k\ x.\ k \leq 2*N \Longrightarrow x \in \{of\text{-}int\ a..\} - Y \Longrightarrow (fs\ k\ has\text{-}vector\text{-}derivative\ fs\ (Suc\ k))\ (at\ x)$
assumes *F-cont* [*continuous-intros*]: $continuous\text{-}on\ \{of\text{-}int\ a..\}\ F$
assumes *F-deriv* [*derivative-intros*]:
$\bigwedge x.\ x \in \{of\text{-}int\ a..\} - Y \Longrightarrow (F\ has\text{-}vector\text{-}derivative\ f\ x)\ (at\ x)$
assumes *limit*:
$((\lambda b.\ (\sum k{=}a..b.\ f\ k) - F\ (of\text{-}int\ b) - (\sum i{<}2*N+1.\ (bernoulli'\ (Suc\ i)\ /\ fact\ (Suc\ i)) *_R fs\ i\ (of\text{-}int\ b))) \longrightarrow C)\ at\text{-}top$
begin

sublocale *euler-maclaurin-int* $F\ f\ fs\ a\ 2*N+1\ C\ Y$
by *standard* (*insert cntbl fs-0 fs-cont fs-deriv F-cont F-deriv limit*, *simp-all*)

theorem *euler-maclaurin-strong-int'*:
assumes $a \leq b$
shows $(\sum k{=}a..b.\ f\ (of\text{-}int\ k)) =$
$F\ (of\text{-}int\ b) + C + (1\ /\ 2) *_R f\ (of\text{-}int\ b) +$
$(\sum i{=}1..N.\ (bernoulli\ (2*i)\ /\ fact\ (2*i)) *_R fs\ (2*i-1)\ (of\text{-}int\ b)) -$
$EM\text{-}remainder\ (2*N+1)\ (fs\ (2*N+1))\ b$
proof −
have $(\sum k{=}a..b.\ f\ (real\text{-}of\text{-}int\ k)) =$
$F\ (of\text{-}int\ b) + C + (\sum i{<}2*N+1.\ (bernoulli'\ (Suc\ i)\ /\ fact\ (Suc\ i)) *_R fs\ i\ (of\text{-}int\ b)) -$
$EM\text{-}remainder\ (2*N+1)\ (fs\ (2*N+1))\ b$
by (*rule euler-maclaurin-strong-int*)
(*simp-all only*: *lessThan-Suc-atMost Suc-eq-plus1* [*symmetric*] *assms*)
also have $\{..<2*N+1\} = insert\ 0\ \{1..2*N\}$ **by** *auto*
also have $(\sum i{\in}\ldots.\ (bernoulli'\ (Suc\ i)\ /\ fact\ (Suc\ i)) *_R fs\ i\ (of\text{-}int\ b)) =$
$(1\ /\ 2) *_R f\ (of\text{-}int\ b) +$
$(\sum i{\in}\{1..2*N\}.\ (bernoulli'\ (Suc\ i)\ /\ fact\ (Suc\ i)) *_R fs\ i\ (of\text{-}int\ b))$
by (*subst sum.insert*) (*auto simp*: *assms bernoulli'-def*)
also have $(\sum i{\in}\{1..2*N\}.\ (bernoulli'\ (Suc\ i)\ /\ fact\ (Suc\ i)) *_R fs\ i\ (of\text{-}int\ b)) =$
$(\sum i{\in}\{1..N\}.\ (bernoulli'\ (2*i)\ /\ fact\ (2*i)) *_R fs\ (2*i-1)\ (of\text{-}int\ b))$
proof (*rule sym*, *rule sum.reindex-bij-witness-not-neutral*)
fix i **assume** $i \in \{1..2*N\} - \{i{\in}\{1..2*N\}.\ even\ i\}$
thus $2 * ((i + 1)\ div\ 2) - 1 = i\ (i + 1)\ div\ 2 \in \{1..N\} - \{\}$
by (*auto elim*!: *oddE*)
qed (*auto simp*: *bernoulli-odd-eq-0 bernoulli'-def algebra-simps*)
also have $\ldots = (\sum i{\in}\{1..N\}.\ (bernoulli\ (2*i)\ /\ fact\ (2*i)) *_R fs\ (2*i-1)\ (of\text{-}int\ b))$
by (*intro sum.cong refl*) (*auto simp*: *bernoulli'-def*)

finally show *?thesis* **by** (*simp only*: *add-ac*)
qed

end

For convenience, we also offer a version where the sum ranges over natural numbers instead of integers.

lemma *sum-atLeastAtMost-of-int-nat-transfer*:
(∑ *k*=*int a*..*int b*. *f* (*of-int k*)) = (∑ *k*=*a*..*b*. *f* (*of-nat k*))
by (*intro sum.reindex-bij-witness*[*of* - *int nat*]) *auto*

lemma *euler-maclaurin-nat-int-transfer*:
fixes *F* **and** *f* :: *real* ⇒ *'a* :: *real-normed-vector*
assumes ((λ*b*. (∑ *k*=*a*..*b*. *f* (*real k*)) − *F* (*real b*) − *T* (*real b*)) ⟶ *C*) *at-top*
shows ((λ*b*. (∑ *k*=*int a*..*b*. *f* (*of-int k*)) − *F* (*of-int b*) − *T* (*of-int b*)) ⟶ *C*) *at-top*
proof −
have ∗: ((λ*b*. (∑ *k*=*int a*..*int b*. *f* (*of-int k*)) − *F* (*of-int* (*int b*)) − *T* (*of-int* (*int b*)))
⟶ *C*) *at-top* **using** *assms* **by** (*subst sum-atLeastAtMost-of-int-nat-transfer*) *simp*
thus *?thesis* **by** (*rule filterlim-int-of-nat-at-topD*)
qed

locale *euler-maclaurin-nat* =
fixes *F f* :: *real* ⇒ *'a* :: *banach*
fixes *fs* :: *nat* ⇒ *real* ⇒ *'a*
fixes *a* :: *nat*
fixes *N* :: *nat* **assumes** *N*: *N* > *0*
fixes *C* :: *'a*
fixes *Y* :: *real set* **assumes** *cntbl*: *countable Y*
assumes *fs-0* [*simp*]: *fs 0* = *f*
assumes *fs-cont* [*continuous-intros*]:
⋀*k*. *k* ≤ *N* ⟹ *continuous-on* {*real a*..} (*fs k*)
assumes *fs-deriv* [*derivative-intros*]:
⋀*k x*. *k* < *N* ⟹ *x* ∈ {*real a*..} − *Y* ⟹ (*fs k has-vector-derivative fs* (*Suc k*) *x*) (*at x*)
assumes *F-cont* [*continuous-intros*]: *continuous-on* {*real a*..} *F*
assumes *F-deriv* [*derivative-intros*]:
⋀*x*. *x* ∈ {*real a*..} − *Y* ⟹ (*F has-vector-derivative f x*) (*at x*)
assumes *limit*:
((λ*b*. (∑ *k*=*a*..*b*. *f k*) − *F* (*real b*) −
(∑ *i*<*N*. (*bernoulli'* (*Suc i*) / *fact* (*Suc i*)) ∗$_R$ *fs i* (*real b*))) ⟶ *C*) *at-top*
begin

sublocale *euler-maclaurin-int F f fs int a N C Y*
by *standard* (*insert N cntbl fs-cont fs-deriv F-cont F-deriv*
euler-maclaurin-nat-int-transfer[*OF limit*], *simp-all*)

theorem *euler-maclaurin-strong-nat*:
assumes *ab*: $a \leq b$
defines $S \equiv (\sum k{=}a..b.\ f\ (real\ k))$
shows $S = F\ (real\ b) + C + (\sum i{<}N.\ (bernoulli'\ (Suc\ i)\ /\ fact\ (Suc\ i)) *_R fs\ i\ (real\ b)) -$
$EM\text{-}remainder\ N\ (fs\ N)\ (int\ b)$
using *euler-maclaurin-strong-int*[*of int b*]
by (*simp add*: *assms sum-atLeastAtMost-of-int-nat-transfer*)

end

locale *euler-maclaurin-nat'* =
fixes $F\ f :: real \Rightarrow 'a :: banach$
fixes $fs :: nat \Rightarrow real \Rightarrow 'a$
fixes $a :: nat$
fixes $N :: nat$
fixes $C :: 'a$
fixes $Y :: real\ set$ **assumes** *cntbl*: *countable Y*
assumes *fs-0* [*simp*]: $fs\ 0 = f$
assumes *fs-cont* [*continuous-intros*]:
$\bigwedge k.\ k \leq 2{*}N{+}1 \Longrightarrow continuous\text{-}on\ \{real\ a..\}\ (fs\ k)$
assumes *fs-deriv* [*derivative-intros*]:
$\bigwedge k\ x.\ k \leq 2{*}N \Longrightarrow x \in \{real\ a..\} - Y \Longrightarrow (fs\ k\ has\text{-}vector\text{-}derivative\ fs\ (Suc\ k)\ x)\ (at\ x)$
assumes *F-cont* [*continuous-intros*]: *continuous-on* $\{real\ a..\}\ F$
assumes *F-deriv* [*derivative-intros*]:
$\bigwedge x.\ x \in \{real\ a..\} - Y \Longrightarrow (F\ has\text{-}vector\text{-}derivative\ f\ x)\ (at\ x)$
assumes *limit*:
$((\lambda b.\ (\sum k{=}a..b.\ f\ k) - F\ (real\ b) -$
$(\sum i{<}2{*}N{+}1.\ (bernoulli'\ (Suc\ i)\ /\ fact\ (Suc\ i)) *_R fs\ i\ (real\ b))) \longrightarrow C)\ at\text{-}top$
begin

sublocale *euler-maclaurin-int'* *F f fs int a N C Y*
by *standard* (*insert cntbl fs-cont fs-deriv F-cont F-deriv*
euler-maclaurin-nat-int-transfer[*OF limit*], *simp-all*)

theorem *euler-maclaurin-strong-nat'*:
assumes $a \leq b$
shows $(\sum k{=}a..b.\ f\ (real\ k)) =$
$F\ (real\ b) + C + (1\ /\ 2) *_R f\ (real\ b) +$
$(\sum i{=}1..N.\ (bernoulli\ (2{*}i)\ /\ fact\ (2{*}i)) *_R fs\ (2{*}i{-}1)\ (real\ b)) -$
$EM\text{-}remainder\ (2{*}N{+}1)\ (fs\ (2{*}N{+}1))\ b$
using *euler-maclaurin-strong-int'*[*of b*]
by (*simp add*: *assms sum-atLeastAtMost-of-int-nat-transfer*)

end

1.5 Bounds on the remainder term

The following theorems provide some simple means to bound the remainder terms. In practice, better bounds can often be obtained e. g. for the n-th remainder term by expanding it to the sum of the first discarded term in the expansion and the $n + 1$-th remainder term.

lemma
fixes $f :: real \Rightarrow 'a :: \{real\text{-}normed\text{-}field, banach\}$
and $g\ g' :: real \Rightarrow real$
assumes *cntbl*: *countable* Y
assumes *pbernpoly-bound*: $\forall x.\ |pbernpoly\ n\ x| \leq D$
assumes *cont-f*: *continuous-on* $\{a..\}\ f$
assumes *cont-g*: *continuous-on* $\{a..\}\ g$
assumes *cont-g'*: *continuous-on* $\{a..\}\ g'$
assumes *limit-g*: $(g \longrightarrow C)$ *at-top*
assumes *f-bound*: $\bigwedge x.\ x \geq a \Longrightarrow norm\ (f\ x) \leq g'\ x$
assumes *deriv*: $\bigwedge x.\ x \in \{a..\} - Y \Longrightarrow (g\ has\text{-}field\text{-}derivative\ g'\ x)\ (at\ x)$
shows *norm-EM-remainder-le-strong-int*:
$\forall x.\ of\text{-}int\ x \geq a \longrightarrow norm\ (EM\text{-}remainder\ n\ f\ x) \leq D\ /\ fact\ n * (C - g\ x)$
and *norm-EM-remainder-le-strong-nat*:
$\forall x.\ real\ x \geq a \longrightarrow norm\ (EM\text{-}remainder\ n\ f\ (int\ x)) \leq D\ /\ fact\ n * (C - g\ x)$
proof −
from *pbernpoly-bound* **have** *D*: $norm\ (pbernpoly\ n\ x) \leq D$ **for** x **by** *auto*
from *this*[*of 0*] **have** *D-nonneg*: $D \geq 0$ **by** *simp*
define D' **where** $D' = D\ /\ fact\ n$
from *D-nonneg* **have** *D'-nonneg*: $D' \geq 0$ **by** (*simp add*: *D'-def*)

have *bound*: $norm\ (EM\text{-}remainder'\ n\ f\ x\ y) \leq D' * (g\ y - g\ x)$
if *xy*: $x \geq a\ x \leq y$ **for** $x\ y :: real$
proof −
have ∗: $\bigwedge u.\ u \in \{x..y\} - Y \Longrightarrow (g\ has\text{-}vector\text{-}derivative\ g'\ u)\ (at\ u\ within\ \{x..y\})$
using *that*
by (*metis Icc-subset-Ici-iff in-mono deriv*
has-real-derivative-iff-has-vector-derivative has-vector-derivative-at-within Diff-iff)
have $norm\ (EM\text{-}remainder'\ n\ f\ x\ y) = norm\ (integral\ \{x..y\}\ (\lambda t.\ pbernpoly\ n\ t *_R f\ t))\ /\ fact\ n$
by (*simp add*: *EM-remainder'-def*)
also have $(\lambda t.\ D * g'\ t)\ integrable\text{-}on\ \{x..y\}$ **using** *xy*
by (*intro integrable-continuous-real continuous-intros continuous-on-subset*[*OF cont-g'*])
auto
hence $norm\ (integral\ \{x..y\}\ (\lambda t.\ pbernpoly\ n\ t *_R f\ t)) \leq integral\ \{x..y\}\ (\lambda t.\ D * g'\ t)$ **using** *D D-nonneg xy*
by (*intro integral-norm-bound-integral integrable-EM-remainder'*
continuous-on-subset[*OF cont-f*]) (*auto intro*!: *mult-mono f-bound*)

```
    also have ... = D * integral {x..y} g' by simp
    also have (g' has-integral (g y - g x)) {x..y}
      using * xy
     by (intro fundamental-theorem-of-calculus-strong[OF cntbl] continuous-on-subset[OF
cont-g]) auto
    hence integral {x..y} g' = g y - g x by (simp add: has-integral-iff)
    finally show ?thesis by (simp add: D'-def divide-simps)
  qed

  have lim: ((λy. EM-remainder' n f x (of-int y)) ⟶ EM-remainder n f x)
at-top
    if x: x ≥ a for x :: int
  proof -
    have (λn. g (real n)) ⟶ C
      by (rule filterlim-compose[OF limit-g filterlim-real-sequentially])
    hence Cauchy: Cauchy (λn. g (real n)) using convergent-eq-Cauchy by blast
    have Cauchy (λy. EM-remainder' n f x (int y))
    proof (rule CauchyI', goal-cases)
      case (1 ε)
      define ε' where ε' = (if D' = 0 then 1 else ε / (2*D'))
      from ‹ε > 0› D'-nonneg have ε': ε' > 0 by (simp add: ε'-def divide-simps)
      from CauchyD[OF Cauchy this] obtain M
        where M: ⋀m n. m ≥ M ⟹ n ≥ M ⟹ norm (g (real m) - g (real n))
< ε' by blast
      show ?case
     proof (intro CauchyI' exI[of - max (max 0 M) (nat x)] allI impI, goal-cases)
        case (1 k l)
        have EM-remainder' n f x k + EM-remainder' n f k l = EM-remainder' n
f x l
          using 1 x by (intro EM-remainder'-combine continuous-on-subset[OF
cont-f]) auto
        hence EM-remainder' n f x l - EM-remainder' n f x k = EM-remainder'
n f k l
          by (simp add: algebra-simps)
        also from 1 x have norm ... ≤ D' * (g l - g k) by (intro bound) auto
        also have g l - g k ≤ norm (g l - g k) by simp
        also from 1 have ... ≤ ε' using M[of l k] by auto
        also from ‹ε > 0› have D' * ε' ≤ ε / 2 by (simp add: ε'-def)
        also from ‹ε > 0› have ... < ε by simp
          finally show ?case by (simp add: D'-nonneg mult-left-mono dist-norm
norm-minus-commute)
      qed
    qed
    then obtain L where (λy. EM-remainder' n f x (int y)) ⟶ L
      by (auto simp: Cauchy-convergent-iff convergent-def)
    from filterlim-int-of-nat-at-topD[OF this]
      have ((λy. EM-remainder' n f x (of-int y)) ⟶ L) at-top by simp
   moreover from this have EM-remainder n f x = L by (rule EM-remainder-eqI)
    ultimately show ((λy. EM-remainder' n f x (of-int y)) ⟶ EM-remainder
```

n f x) *at-top*
by *simp*
qed

have ∗: *norm* (*EM-remainder n f x*) ≤ *D′* ∗ (*C* − *g x*) **if** *x*: *x* ≥ *a* **for** *x* :: *int*
proof (*rule tendsto-le*)
show ((λ*y*. *D′* ∗ (*g* (*of-int y*) − *g* (*of-int x*))) ⟶ *D′* ∗ (*C* − *g* (*of-int x*))) *at-top*
by (*intro tendsto-intros filterlim-compose*[*OF limit-g*])
show ((λ*y*. *norm* (*EM-remainder′ n f x* (*of-int y*))) ⟶ *norm* (*EM-remainder n f x*)) *at-top*
using *x* **by** (*intro tendsto-norm lim*)
show *eventually* (λ*y*. *norm* (*EM-remainder′ n f* (*of-int x*) (*of-int y*))
≤ *D′* ∗ (*g* (*of-int y*) − *g* (*of-int x*))) *at-top*
using *eventually-ge-at-top*[*of x*] **by** *eventually-elim* (*rule bound*, *insert x*, *simp-all*)
qed *simp-all*
thus ∀ *x*. *of-int x* ≥ *a* ⟶ *norm* (*EM-remainder n f x*) ≤ *D′* ∗ (*C* − *g x*) **by** *blast*

have *norm* (*EM-remainder n f x*) ≤ *D′* ∗ (*C* − *g x*) **if** *x*: *x* ≥ *a* **for** *x* :: *nat*
using *x* ∗[*of int x*] **by** *simp*
thus ∀ *x*. *real x* ≥ *a* ⟶ *norm* (*EM-remainder n f* (*int x*)) ≤ *D′* ∗ (*C* − *g x*) **by** *blast*
qed

lemma
fixes *f* :: *real* ⇒ *′a* :: {*real-normed-field*, *banach*}
and *g g′* :: *real* ⇒ *real*
assumes *cntbl*: *countable Y*
assumes *pbernpoly-bound*: ∀ *x*. |*pbernpoly n x*| ≤ *D*
assumes *cont-f*: *continuous-on* {*a*..} *f*
assumes *cont-g*: *continuous-on* {*a*..} *g*
assumes *cont-g′*: *continuous-on* {*a*..} *g′*
assumes *limit-g*: (*g* ⟶ 0) *at-top*
assumes *f-bound*: ⋀*x*. *x* ≥ *a* ⟹ *norm* (*f x*) ≤ *g′ x*
assumes *deriv*: ⋀*x*. *x* ∈ {*a*..} − *Y* ⟹ (*g has-field-derivative* −*g′ x*) (*at x*)
shows *norm-EM-remainder-le-strong-int′*:
∀ *x*. *of-int x* ≥ *a* ⟶ *norm* (*EM-remainder n f x*) ≤ *D* / *fact n* ∗ *g x*
and *norm-EM-remainder-le-strong-nat′*:
∀ *x*. *real x* ≥ *a* ⟶ *norm* (*EM-remainder n f* (*int x*)) ≤ *D* / *fact n* ∗ *g x*
proof −
have ∀ *x*. *of-int x* ≥ *a* ⟶ *norm* (*EM-remainder n f x*) ≤ *D* / *fact n* ∗ (0 − (−*g x*)) **using** *assms*
by (*intro norm-EM-remainder-le-strong-int*[*OF cntbl pbernpoly-bound - - cont-g′*])
(*auto intro*!: *continuous-intros tendsto-eq-intros derivative-eq-intros*)
thus ∀ *x*. *of-int x* ≥ *a* ⟶ *norm* (*EM-remainder n f x*) ≤ *D* / *fact n* ∗ *g x* **by** *auto*
next

have $\forall x.\ real\ x \geq a \longrightarrow norm\ (EM\text{-}remainder\ n\ f\ (int\ x)) \leq D\ /\ fact\ n * (0 - (-g\ x))$ **using** *assms*
by (*intro norm-EM-remainder-le-strong-nat*[*OF cntbl pbernpoly-bound - - cont-g'*])
(*auto intro*!: *continuous-intros tendsto-eq-intros derivative-eq-intros*)
thus $\forall x.\ real\ x \geq a \longrightarrow norm\ (EM\text{-}remainder\ n\ f\ (int\ x)) \leq D\ /\ fact\ n * g\ x$
by *auto*
qed

lemma *norm-EM-remainder'-le*:
fixes $f :: real \Rightarrow 'a :: \{real\text{-}normed\text{-}field, banach\}$
and $g\ g' :: real \Rightarrow real$
assumes *cont-f*: *continuous-on* $\{a..\}\ f$
assumes *cont-g'*: *continuous-on* $\{a..\}\ g'$
assumes *f-bigo*: *eventually* $(\lambda x.\ norm\ (f\ x) \leq g'\ x)$ *at-top*
assumes *deriv*: *eventually* $(\lambda x.\ (g\ has\text{-}field\text{-}derivative\ g'\ x)\ (at\ x))$ *at-top*
obtains $C\ D$ **where**
eventually $(\lambda x.\ norm\ (EM\text{-}remainder'\ n\ f\ a\ x) \leq C + D * g\ x)$ *at-top*
proof −
note *cont* = *continuous-on-subset*[*OF cont-f*] *continuous-on-subset*[*OF cont-g'*]
from *bounded-pbernpoly*[*of n*] **obtain** D **where** D: $\bigwedge x.\ norm\ (pbernpoly\ n\ x) \leq D$ **by** *blast*
from *this*[*of 0*] **have** *D-nonneg*: $D \geq 0$ **by** *simp*
from *eventually-conj*[*OF f-bigo eventually-conj*[*OF deriv eventually-ge-at-top*[*of a*]]]
obtain *x0* **where** *x0*:
$x0 \geq a\ \bigwedge x.\ x \geq x0 \Longrightarrow norm\ (f\ x) \leq g'\ x$
$\bigwedge x.\ x \geq x0 \Longrightarrow (g\ has\text{-}field\text{-}derivative\ g'\ x)\ (at\ x)$
by (*auto simp*: *eventually-at-top-linorder*)
define C **where** $C = (norm\ (integral\ \{a..x0\}\ (\lambda t.\ pbernpoly\ n\ t *_R f\ t)) - D * g\ x0)\ /\ fact\ n$

have *eventually* $(\lambda x.\ norm\ (EM\text{-}remainder'\ n\ f\ a\ x) \leq C + D\ /\ fact\ n * g\ x)$ *at-top*
using *eventually-ge-at-top*[*of x0*]
proof *eventually-elim*
case (*elim x*)
have $integral\ \{a..x\}\ (\lambda t.\ pbernpoly\ n\ t *_R f\ t) =$
$integral\ \{a..x0\}\ (\lambda t.\ pbernpoly\ n\ t *_R f\ t) +$
$integral\ \{x0..x\}\ (\lambda t.\ pbernpoly\ n\ t *_R f\ t)$ (**is** - = *?I1* + *?I2*) **using** *elim x0*(*1*)
by (*intro Henstock-Kurzweil-Integration.integral-combine* [*symmetric*] *integrable-EM-remainder' cont*) *auto*
also have $norm\ \ldots \leq norm\ ?I1 + norm\ ?I2$ **by** (*rule norm-triangle-ineq*)
also have $norm\ ?I2 \leq integral\ \{x0..x\}\ (\lambda t.\ D * g'\ t)$
using *x0 D D-nonneg*
by (*intro integral-norm-bound-integral integrable-EM-remainder'*)
(*auto intro*!: *integrable-continuous-real continuous-intros cont mult-mono*)
also have $\ldots = D * integral\ \{x0..x\}\ g'$ **by** *simp*
also from *elim* **have** $(g'\ has\text{-}integral\ (g\ x - g\ x0))\ \{x0..x\}$

by (*intro fundamental-theorem-of-calculus*)
(*auto intro*!: *has-field-derivative-at-within*[*OF x0*(*3*)]
simp: *has-real-derivative-iff-has-vector-derivative* [*symmetric*])
hence *integral* {*x0*..*x*} *g'* = *g x* − *g x0* **by** (*simp add*: *has-integral-iff*)
finally have *norm* (*integral* {*a*..*x*} (λ*t*. *pbernpoly n t* ∗$_R$ *f t*)) ≤ *norm ?I1* + *D* ∗ (*g x* − *g x0*)
by *simp-all*
thus *norm* (*EM-remainder' n f a x*) ≤ *C* + *D* / *fact n* ∗ *g x*
by (*simp add*: *EM-remainder'-def field-simps C-def*)
qed
thus *?thesis* **by** (*rule that*)
qed

1.6 Application to harmonic numbers

As a first application, we can apply the machinery we have developed to the harmonic numbers.

definition *harm-remainder* :: *nat* ⇒ *nat* ⇒ *real* **where**
harm-remainder N n = *EM-remainder* (*2*∗*N*+*1*) (λ*x*. −*fact* (*2*∗*N*+*1*) / *x* ^ (*2*∗*N*+*2*)) (*int n*)

lemma *harm-expansion*:
assumes *n*: *n* > *0* **and** *N*: *N* > *0*
shows *harm n* = *ln n* + *euler-mascheroni* + *1* / (*2*∗*n*) −
(∑ *i*=*1*..*N*. *bernoulli* (*2*∗*i*) / ((*2*∗*i*) ∗ *n* ^(*2*∗*i*))) − *harm-remainder N n*
proof −
define *fs* **where** *fs* = (λ*k x*. (−*1*) ^ *k* ∗ *fact k* / *x* ^(*Suc k*) :: *real*)
interpret *euler-maclaurin-nat' ln* λ*x*. *1*/*x fs 1 N euler-mascheroni* {}
proof
fix *k x* **assume** *k* ≤ *2*∗*N x* ∈ {*real 1*..} − {}
thus (*fs k has-vector-derivative fs* (*Suc k*) *x*) (*at x*)
by (*cases k* = *0*)
(*auto intro*!: *derivative-eq-intros*
simp: *fs-def has-real-derivative-iff-has-vector-derivative* [*symmetric*]
field-simps power-diff)
next
have (λ*b*. *harm b* − *ln* (*real b*) −
(∑ *i*<*2*∗*N*+*1*. *bernoulli'* (*Suc i*) ∗ (− *1*) ^ *i* / (*real* (*Suc i*) ∗ (*real b* ^ *Suc i*))))
⟶ (*euler-mascheroni* − (∑ *i*<*2*∗*N*+*1*. *0*))
by (*intro tendsto-diff euler-mascheroni-LIMSEQ tendsto-sum*
real-tendsto-divide-at-top[*OF tendsto-const*]
filterlim-tendsto-pos-mult-at-top[*OF tendsto-const*] *filterlim-pow-at-top*
filterlim-real-sequentially) *auto*
thus (λ*b*. (∑ *k* = *1*..*b*. *1* / *real k*) − *ln* (*real b*) −
(∑ *i*<*2*∗*N*+*1*. (*bernoulli'* (*Suc i*) / *fact* (*Suc i*)) ∗$_R$ *fs i* (*real b*))) ⟶ *euler-mascheroni*
by (*simp add*: *harm-def divide-simps fs-def*)

```
  qed (insert n N, auto intro!: continuous-intros derivative-eq-intros
        simp: fs-def has-real-derivative-iff-has-vector-derivative [symmetric])

  have harm n = (∑ k=1..n. 1 / real k) by (simp add: harm-def divide-simps)
  also have ... = ln (real n) + euler-mascheroni + (1/2) *R (1 / real n) +
                  (∑ i=1..N. (bernoulli (2*i) / fact (2*i)) *R fs (2*i-1) (real n))
-
                    EM-remainder (2*N+1) (fs (2*N+1)) (int n) using n N
    using n by (intro euler-maclaurin-strong-nat') simp-all
  also have (∑ i=1..N. (bernoulli (2*i) / fact (2*i)) *R (fs (2*i-1) (real n)))
=
              (∑ i=1..N. -(bernoulli (2*i) / (real (2*i) * real n ^ (2*i))))
    by (intro sum.cong refl)
       (simp-all add: fs-def divide-simps fact-reduce del: of-nat-Suc power-Suc)
  also have ... = -(∑ i=1..N. bernoulli (2*i) / (real (2*i) * real n ^ (2*i)))
    by (simp add: sum-negf)
  finally show ?thesis unfolding fs-def by (simp add: harm-remainder-def)
qed

lemma of-nat-ge-1-iff: of-nat x ≥ (1 :: 'a :: linordered-semidom) ⟷ x ≥ 1
  using of-nat-le-iff[of 1 x] by (simp del: of-nat-le-iff)

lemma harm-remainder-bound:
  fixes N :: nat
  assumes N: N > 0
  shows ∃ C. ∀ n≥1. norm (harm-remainder N n) ≤ C / real n ^ (2*N+1)
proof -
  from bounded-pbernpoly[of 2*N+1] obtain D where D: ∀ x. |pbernpoly (2*N+1)
x| ≤ D by auto
  have ∀ x. 1 ≤ real x ⟶ norm (harm-remainder N x) ≤
          D / fact (2*N+1) * (fact (2*N) / x ^ (2*N+1))
    unfolding harm-remainder-def of-int-of-nat-eq
  proof (rule norm-EM-remainder-le-strong-nat'[of {}])
    fix x :: real assume x: x ≥ 1
     show norm (-fact (2*N+1) / x ^ (2 * N + 2)) ≤ fact (2*N+1) / x ^
(2*N+2)
      using x by simp
  next
    show ((λx::real. fact (2 * N) / x ^ (2 * N + 1)) ⟶⟶ 0) at-top
      by (intro real-tendsto-divide-at-top[OF tendsto-const] filterlim-pow-at-top fil-
terlim-ident)
         simp-all
 qed (insert N D, auto intro!: derivative-eq-intros continuous-intros simp: field-simps
power-diff)
  hence ∀ x. 1 ≤ x ⟶ norm (harm-remainder N x) ≤ D / (2*N+1) / real x ^
(2*N+1) by simp
  thus ?thesis by blast
qed
```

1.7 Application to sums of inverse squares

In the same vein, we can derive the asymptotics of the partial sum of inverse squares.

lemma *sum-inverse-squares-expansion*:
assumes *n*: $n > 0$ **and** *N*: $N > 0$
shows $(\sum k{=}1..n.\ 1\ /\ real\ k\ \hat{}\ 2) =$
$pi\ \hat{}\ 2\ /\ 6 - 1\ /\ real\ n + 1\ /\ (2 * real\ n\ \hat{}\ 2) -$
$(\sum i{=}1..N.\ bernoulli\ (2{*}i)\ /\ n\ \hat{}\ (2{*}i{+}1)) -$
$EM\text{-}remainder\ (2{*}N{+}1)\ (\lambda x.\ -fact\ (2{*}N{+}2)\ /\ x\ \hat{}\ (2{*}N{+}3))$
$(int\ n)$
proof −
have *3*: $3 = Suc\ (Suc\ (Suc\ 0))$ **by** (*simp add*: *eval-nat-numeral*)
define *fs* **where** $fs = (\lambda k\ x.\ (-1)\ \hat{}\ k * fact\ (Suc\ k)\ /\ x\ \hat{}\ (k{+}2) :: real)$
interpret *euler-maclaurin-nat'* $\lambda x.\ -1/x\ \lambda x.\ 1/x\hat{}2\ fs\ 1\ N\ pi\hat{}2/6\ \{\}$
proof
fix *k x* **assume** $k \le 2{*}N\ x \in \{real\ 1..\} - \{\}$
thus (*fs k has-vector-derivative fs* (*Suc k*) *x*) (*at x*)
by (*cases k* = *0*)
(*auto intro*!: *derivative-eq-intros*
simp: *fs-def has-real-derivative-iff-has-vector-derivative* [*symmetric*]
field-simps power-diff)
next
from *inverse-squares-sums*
have $(\lambda n.\ \sum k{<}n.\ 1\ /\ real\ (Suc\ k)\ \hat{}\ 2) \longrightarrow pi^2\ /\ 6$ **by** (*simp add*: *sums-def*)
also have $(\lambda n.\ \sum k{<}n.\ 1\ /\ real\ (Suc\ k)\ \hat{}\ 2) = (\lambda n.\ \sum k{=}1..n.\ 1\ /\ real\ k\ \hat{}\ 2)$
by (*intro ext sum.reindex-bij-witness*[*of* - $\lambda n.\ n - 1$ *Suc*]) *auto*
finally have $(\lambda b.\ (\sum k = 1..b.\ 1\ /\ real\ k\hat{}2) + 1\ /\ real\ b -$
$(\sum i{<}2{*}N{+}1.\ bernoulli'\ (Suc\ i) * (-1)\ \hat{}\ i\ /\ (real\ b\ \hat{}\ (i{+}2))))$
$\longrightarrow (pi\hat{}2/6 + 0 - (\sum i{<}2{*}N{+}1.\ 0))$
by (*intro tendsto-diff tendsto-add real-tendsto-divide-at-top*[*OF tendsto-const*]
filterlim-tendsto-pos-mult-at-top[*OF tendsto-const*] *filterlim-pow-at-top*
filterlim-real-sequentially tendsto-sum) *auto*
thus $(\lambda b.\ (\sum k = 1..b.\ 1\ /\ real\ k\hat{}2) - (-1\ /\ real\ b) -$
$(\sum i{<}2{*}N{+}1.\ (bernoulli'\ (Suc\ i)\ /\ fact\ (Suc\ i)) *_R fs\ i\ (real\ b))) \longrightarrow$
$pi\hat{}2/6$
by (*simp add*: *harm-def field-simps fs-def del*: *power-Suc of-nat-Suc*)
qed (*insert n N*, *auto intro*!: *continuous-intros derivative-eq-intros*
simp: *fs-def has-real-derivative-iff-has-vector-derivative* [*symmetric*] *power2-eq-square*)

have $(\sum k{=}1..n.\ 1\ /\ real\ k\ \hat{}\ 2) = -1\ /\ real\ n + pi\hat{}2/6 + (1/2) *_R (1\ /\ real\ n\hat{}2) +$
$(\sum i{=}1..N.\ (bernoulli\ (2{*}i)\ /\ fact\ (2{*}i)) *_R fs\ (2{*}i{-}1)\ (real\ n)) -$
$EM\text{-}remainder\ (2{*}N{+}1)\ (fs\ (2{*}N{+}1))\ (int\ n)$ **using** *n N*
using *n* **by** (*intro euler-maclaurin-strong-nat'*) *simp-all*
also have $(\sum i{=}1..N.\ (bernoulli\ (2{*}i)\ /\ fact\ (2{*}i)) *_R (fs\ (2{*}i{-}1)\ (real\ n)))$

=
(∑ i=1..N. −(bernoulli (2∗i) / (real n ^ (2∗i+1))))
by (*intro sum.cong refl*)
(*simp-all add*: *fs-def divide-simps fact-reduce del*: *of-nat-Suc power-Suc*)
also have ... = −(∑ i=1..N. *bernoulli* (2∗i) / *real* n ^ (2∗i+1))
by (*simp add*: *sum-negf*)
finally show *?thesis* **unfolding** *fs-def* **by** (*simp add*: *fs-def 3*)
qed

lemma *sum-inverse-squares-remainder-bound*:
fixes N :: *nat*
assumes N: N > 0
defines R ≡ (λn. *EM-remainder* (2∗N+1) (λx. −*fact* (2∗N+2) / x ^ (2∗N+3)) (*int* n))
shows ∃ C. ∀ n≥1. *norm* (R n) ≤ C / *real* n ^ (2∗N+2)
proof −
have 3: 3 = *Suc* (*Suc* (*Suc* 0)) **by** *simp*
from *bounded-pbernpoly*[*of* 2∗N+1] **obtain** D **where** D: ∀ x. |*pbernpoly* (2∗N+1) x| ≤ D **by** *auto*
have ∀ x. 1 ≤ *real* x ⟶ *norm* (R x) ≤ D / *fact* (2∗N+1) ∗ (*fact* (2∗N+1) / x ^ (2∗N+2))
unfolding *R-def of-int-of-nat-eq*
proof (*rule norm-EM-remainder-le-strong-nat'*[*of* {}])
fix x :: *real* **assume** x: x ≥ 1
show *norm* (−*fact* (2∗N+2) / x ^ (2∗N+3)) ≤ *fact* (2∗N+2) / x ^ (2∗N+3)
using x **by** *simp*
next
show ((λx::*real*. *fact* (2∗N+1) / x ^ (2∗N+2)) ⟶ 0) *at-top*
by (*intro real-tendsto-divide-at-top*[*OF tendsto-const*] *filterlim-pow-at-top filterlim-ident*)
simp-all
qed (*insert N D*, *auto intro*!: *derivative-eq-intros continuous-intros simp*: *field-simps power-diff 3*)
hence ∀ x≥1. *norm* (R x) ≤ D / *real* x ^ (2 ∗ N + 2) **by** *simp*
thus *?thesis* **by** *blast*
qed

end

2 Connection of Euler–MacLaurin summation to Landau symbols

theory *Euler-MacLaurin-Landau*
imports
Euler-MacLaurin
Landau-Symbols.Landau-More
begin

2.1 O-bound for the remainder term

Landau symbols allow us to state the bounds on the remainder terms from the Euler–MacLaurin formula a bit more nicely.

lemma
fixes f :: *real* ⇒ *'a* :: {*real-normed-field*, *banach*}
and g g' :: *real* ⇒ *real*
assumes *fin*: *countable* Y
assumes *cont-f*: *continuous-on* {*a*..} f
assumes *cont-g*: *continuous-on* {*a*..} g
assumes *cont-g'*: *continuous-on* {*a*..} g'
assumes *limit-g*: $(g \longrightarrow 0)$ *at-top*
assumes *f-bound*: $\bigwedge x.\ x \geq a \Longrightarrow$ *norm* $(f\ x) \leq g'\ x$
assumes *deriv*: $\bigwedge x.\ x \in \{a..\} - Y \Longrightarrow (g$ *has-field-derivative* $-g'\ x)$ *(at* $x)$
shows *EM-remainder-strong-bigo-int*: $(\lambda x{::}int.$ *norm* $(EM\text{-}remainder\ n\ f\ x)) \in O(g)$
and *EM-remainder-strong-bigo-nat*: $(\lambda x{::}nat.$ *norm* $(EM\text{-}remainder\ n\ f\ x)) \in O(g)$
proof −
from *bounded-pbernpoly*[*of n*] **obtain** D **where** D: $\forall x.\ |pbernpoly\ n\ x| \leq D$ **by** *auto*
from *norm-EM-remainder-le-strong-int'*[*OF fin D assms*(2−)]
have ∗: $\bigwedge x.\ x \geq a \longrightarrow$ *norm* $(EM\text{-}remainder\ n\ f\ x) \leq D$ / *fact* $n * g\ x$ **by** *auto*
have ∗∗: *eventually* $(\lambda x{::}int.$ *norm* $(EM\text{-}remainder\ n\ f\ x) \leq abs\ (D$ / *fact* $n) *$ *abs* $(g\ x))$ *at-top*
using *eventually-ge-at-top*[*of ceiling a*]
proof *eventually-elim*
case (*elim x*)
with ∗[*of x*] **have** *norm* $(EM\text{-}remainder\ n\ f\ x) \leq D$ / *fact* $n * g\ x$ **by** (*simp add*: *ceiling-le-iff*)
also have ... $\leq$ *abs* $(D$ / *fact* $n * g\ x)$ **by** (*rule abs-ge-self*)
also have ... $=$ *abs* $(D$ / *fact* $n) *$ *abs* $(g\ x)$ **by** (*simp add*: *abs-mult*)
finally show *?case* **.**
qed
thus $(\lambda x{::}int.$ *norm* $(EM\text{-}remainder\ n\ f\ x)) \in O(g)$
by (*intro bigoI*[*of* - *abs D* / *fact n*]) (*auto elim*!: *eventually-mono*)
hence $(\lambda x{::}nat.$ *norm* $(EM\text{-}remainder\ n\ f\ (int\ x))) \in O(\lambda x.\ g\ (of\text{-}int\ (int\ x)))$
by (*rule landau-o.big.compose*) (*fact filterlim-int-sequentially*)
thus $(\lambda x{::}nat.$ *norm* $(EM\text{-}remainder\ n\ f\ x)) \in O(g)$ **by** *simp*
qed

2.2 Asymptotic expansion of the harmonic numbers

We can now show the asymptotic expansion

$$H_n = \ln n + \gamma + \frac{1}{2n} - \sum_{i=1}^{m} \frac{B_{2i}}{2i} n^{-2i} + O(n^{-2m-2})$$

lemma *harm-remainder-bigo*:

assumes $N > 0$
shows *harm-remainder N* $\in O(\lambda n.\ 1\ /\ real\ n\ \hat{}\ (2 * N + 1))$
proof −
from *harm-remainder-bound*[*OF assms*]
obtain C **where** $\forall\, n{\geq}1.\ norm\ (harm\text{-}remainder\ N\ n) \leq C\ /\ real\ n\ \hat{}\ (2 * N + 1)$ **..**
thus *?thesis*
by (*intro bigoI*[*of - C*] *eventually-mono*[*OF eventually-ge-at-top*[*of 1*]]) *auto*
qed

lemma *harm-expansion-bigo*:
fixes N :: *nat*
defines $T \equiv \lambda n.\ ln\ n + euler\text{-}mascheroni + 1\ /\ (2{*}n) - (\sum i{=}1..N.\ bernoulli\ (2{*}i)\ /\ ((2{*}i) * n\ \hat{}\ (2{*}i)))$
defines $S \equiv (\lambda n.\ bernoulli\ (2{*}(Suc\ N))\ /\ ((2{*}Suc\ N) * real\ n\ \hat{}\ (2{*}Suc\ N)))$
shows $(\lambda n.\ harm\ n - T\ n) \in O(\lambda n.\ 1\ /\ real\ n\ \hat{}\ (2 * N + 2))$
proof −
have $(\lambda n.\ harm\ n - T\ n) \in \Theta(\lambda n.\ -S\ n - harm\text{-}remainder\ (Suc\ N)\ n)$
by (*intro bigthetaI-cong eventually-mono*[*OF eventually-gt-at-top*[*of 0*::*nat*]])
(*auto simp*: *T-def harm-expansion*[*of - Suc N*] *S-def*)
also have $(\lambda n.\ -S\ n - harm\text{-}remainder\ (Suc\ N)\ n) \in O(\lambda n.\ 1\ /\ real\ n\ \hat{}\ (2 * N + 2))$
proof (*intro sum-in-bigo*)
show $(\lambda x.\ -S\ x) \in O(\lambda n.\ 1\ /\ real\ n\ \hat{}\ (2 * N + 2))$ **unfolding** *S-def*
by (*rule landau-o.big.compose*[*OF - filterlim-real-sequentially*]) *simp*
have *harm-remainder* (*Suc N*) $\in O(\lambda n.\ 1\ /\ real\ n\ \hat{}\ (2 * Suc\ N + 1))$
by (*rule harm-remainder-bigo*) *simp-all*
also have $(\lambda n.\ 1\ /\ real\ n\ \hat{}\ (2 * Suc\ N + 1)) \in O(\lambda n.\ 1\ /\ real\ n\ \hat{}\ (2 * N + 2))$
by (*rule landau-o.big.compose*[*OF - filterlim-real-sequentially*]) *simp*
finally show *harm-remainder* (*Suc N*) $\in \ldots$ **.**
qed
finally show *?thesis* **.**
qed

lemma *harm-expansion-bigo-simple1*:
$(\lambda n.\ harm\ n - (ln\ n + euler\text{-}mascheroni + 1\ /\ (2 * n))) \in O(\lambda n.\ 1\ /\ n\ \hat{}\ 2)$
using *harm-expansion-bigo*[*of 0*] **by** (*simp add*: *power2-eq-square*)

lemma *harm-expansion-bigo-simple2*:
$(\lambda n.\ harm\ n - (ln\ n + euler\text{-}mascheroni)) \in O(\lambda n.\ 1\ /\ n)$
proof −
have $(\lambda n.\ harm\ n - (ln\ n + euler\text{-}mascheroni + 1\ /\ (2 * n)) + 1\ /\ (2 * n)) \in O(\lambda n.\ 1\ /\ n)$
proof (*rule sum-in-bigo*)
have $(\lambda n.\ harm\ n - (ln\ n + euler\text{-}mascheroni + 1\ /\ (2 * n))) \in O(\lambda n.\ 1\ /\ real\ n\ \hat{}\ 2)$
using *harm-expansion-bigo-simple1* **by** *simp*
also have $(\lambda n.\ 1\ /\ real\ n\ \hat{}\ 2) \in O(\lambda n.\ 1\ /\ real\ n)$

by (*rule landau-o.big.compose*[*OF - filterlim-real-sequentially*]) *simp-all*
finally show (λ*n*. *harm n* − (*ln n* + *euler-mascheroni* + *1* / (*2* ∗ *n*))) ∈ *O*(λ*n*. *1* / *n*) **by** *simp*
qed *simp-all*
thus *?thesis* **by** (*simp add*: *algebra-simps*)
qed

lemma *harm-expansion-bigo-simple′*:
harm =*o* (λ*n*. *ln n* + *euler-mascheroni* + *1* / (*2* ∗ *n*)) +*o* *O*(λ*n*. *1* / *n* ^ *2*)
using *harm-expansion-bigo-simple1*
by (*subst set-minus-plus* [*symmetric*]) (*simp-all add*: *fun-diff-def*)

2.3 Asymptotic expansion of the sum of inverse squares

Similarly to before, we show

$$\sum_{i=1}^{n} \frac{1}{i^2} = \frac{\pi^2}{6} - \frac{1}{n} + \frac{1}{2n^2} - \sum_{i=1}^{m} B_{2i} n^{-2i-1} + O(n^{-2m-3})$$

context
fixes *R* :: *nat* ⇒ *nat* ⇒ *real*
defines *R* ≡ (λ*N n*. *EM-remainder* (*2*∗*N*+*1*) (λ*x*. −*fact* (*2*∗*N*+*2*) / *x* ^(*2*∗*N*+*3*)) (*int n*))
begin

lemma *sum-inverse-squares-remainder-bigo*:
assumes *N* > *0*
shows *R N* ∈ *O*(λ*n*. *1* / *real n* ^(*2* ∗ *N* + *2*))
proof −
from *sum-inverse-squares-remainder-bound*[*OF assms*]
obtain *C*
where ∀ *n*≥*1*. *norm* (*EM-remainder* (*2* ∗ *N* + *1*) (λ*x*. − *fact* (*2* ∗ *N* + *2*) / *x* ^(*2* ∗ *N* + *3*)) (*int n*))
≤ *C* / *real n* ^(*2* ∗ *N* + *2*) **..**
thus *?thesis*
by (*intro bigoI*[*of - C*] *eventually-mono*[*OF eventually-ge-at-top*[*of 1*]]) (*auto simp*: *R-def*)
qed

lemma *sum-inverse-squares-expansion-bigo*:
fixes *N* :: *nat*
defines *T* ≡ λ*n*. *pi* ^ *2* / *6* − *1* / *n* + *1* / (*2*∗*n* ^ *2*) −
(∑ *i*=*1*..*N*. *bernoulli* (*2*∗*i*) / (*n* ^(*2*∗*i*+*1*)))
defines *S* ≡ (λ*n*. *bernoulli* (*2*∗(*Suc N*)) / (*real n* ^(*2*∗*N*+*3*)))
shows (λ*n*. (∑ *i*=*1*..*n*. *1* / *real i* ^ *2*) − *T n*) ∈ *O*(λ*n*. *1* / *real n* ^(*2* ∗ *N* + *3*))
proof −
have *3*: *3* = *Suc* (*Suc* (*Suc 0*)) **by** *simp*

```
  have (λn. (∑ i=1..n. 1 / real i ^ 2) − T n) ∈ Θ(λn. −S n − R (Suc N) n)
unfolding R-def
    by (intro bigthetaI-cong eventually-mono[OF eventually-gt-at-top[of 0::nat]])
       (auto simp: T-def sum-inverse-squares-expansion[of - Suc N] S-def 3
             simp del: One-nat-def)
  also have (λn. −S n − R (Suc N) n) ∈ O(λn. 1 / real n ^ (2 * N + 3))
  proof (intro sum-in-bigo)
    show (λx. − S x) ∈ O(λn. 1 / real n ^ (2 * N + 3)) unfolding S-def
      by (rule landau-o.big.compose[OF - filterlim-real-sequentially]) simp
    have R (Suc N) ∈ O(λn. 1 / real n ^ (2 * Suc N + 2))
      by (rule sum-inverse-squares-remainder-bigo) simp-all
    also have 2 * Suc N + 2 = 2 * N + 4 by simp
    also have (λn. 1 / real n ^ (2 * N + 4)) ∈ O(λn. 1 / real n ^ (2 * N + 3))
      by (rule landau-o.big.compose[OF - filterlim-real-sequentially]) simp
    finally show R (Suc N) ∈ ... .
  qed
  finally show ?thesis .
qed

lemma sum-inverse-squares-expansion-bigo-simple:
  (λn. (∑ i=1..n. 1 / real i ^ 2) − (pi ^ 2 / 6 − 1 / n + 1 / (2*n^2))) ∈ O(λn.
1 / n ^ 3)
  using sum-inverse-squares-expansion-bigo[of 0] by (simp add: power2-eq-square)

lemma sum-inverse-squares-expansion-bigo-simple':
  (λn. (∑ i=1..n. 1 / real i ^ 2)) =o (λn. pi ^ 2 / 6 − 1 / n + 1 / (2*n^2))
+o O(λn. 1 / n^3)
  using sum-inverse-squares-expansion-bigo-simple
  by (subst set-minus-plus [symmetric]) (simp-all add: fun-diff-def)

end

end
```

References

[1] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables.* Dover, New York, tenth printing edition, 1964.

[2] T. M. Apostol. *An Elementary View of Euler's Summation Formula. The American Mathematical Monthly*, 106(5):409–418, 1999.

[3] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science.* Addison–Wesley, Boston, MA, USA, 2nd edition, 1994.