

Approximate Bernoulli numbers with the HP 50g

D.A. Burkett (0db)

4 May 2025

Contents

1	Description	1
2	Summary of programs	2
3	IBERNOULLI with approximate results: program IBERN	2
4	B_n via approximation: program Bn	3
5	B_n via table look-up: programs BnT, BnTL, BnTL2	7
6	About Bernoulli numbers	11
A	Evaluating $\int \ln(\sin(x)) dx$	17
B	References	21

1: Description

Five programs which calculate approximations of the Bernoulli numbers B_n are described. First, the native IBERNOULLI function is considered, resulting in a small program for use with Numeric Approx mode. Program Bn is described next, which finds B_n using approximation methods for a trade-off between speed, accuracy, and program size. The third program, BnT, uses table look-up for the best execution time and accuracy, but the program is rather large. Two more programs, BnTL and BnTL2, return lists of B_n using BnT. A simple asymptotic expression exists to find the ratio B_n/B_{n-2} , which is implemented as program BnRAT. Finally, Appendix A demonstrates the evaluation of an integral with Bernoulli numbers.

A summary of Bernoulli number properties follows the program descriptions. Annotated references are given in the last section.

There are various historical notations for the Bernoulli numbers, but I use the common contemporary notation where $n = 0, 1, 2, 3, \dots$.

Exact values of B_n , as rational numbers, are often required for problems in number theory and combinatorics. The numerators of these fractions increase rapidly. For example, the numerator of B_{370} has 499 digits but the denominator has only two. The 50g CAS can work with these numbers in Exact mode but execution time is longer as a result. For exact Bernoulli numbers you should look into the BERNOLLIFAST package[†] which provides programs for approximate and exact Bernoulli numbers.

Approximate Bernoulli numbers have their uses in evaluating integrals, series and sums, and approximating other functions. The programs described here are designed for those applications but can certainly be used to estimate results for ‘exact’ calculations as well.

[†] John Keith and Gerald Hillier, <https://www.hpcalc.org/details/9502>

My goal for these programs was to provide all the Bernoulli numbers that are representable with the default 50g floating-point format, which has a maximum of $9.99999999999 \times 10^{499}$. B_{372} is the largest representable number, not including B_{373} which is zero. This goal is probably overkill as the majority of practical problems which should be attempted with the 50g won't need more than the first 15 or 20 non-zero Bernoulli numbers.

A strategy decision is required when calculating the Bernoulli numbers. It can be time-consuming to calculate them accurately, so most programs I reviewed build a table of the numbers in one of two ways. The first option is to calculate and save a table of all the numbers which may be needed, or which can be represented in the available number format. The second option is to fill the table as needed, based on the the input n . For example, if the table contains the numbers up to B_{80} and B_{100} is required, then the numbers from $n = 82$ through $n = 100$ are calculated and added to the table. Both options are particularly appropriate when a recursive algorithm for B_n is used.

The resulting table provides the most accurate results but takes a lot of memory. Considering these trade-offs led me to write two programs, one table-based, and a second program which returns a single B_n as quickly and accurately as possible. For the table-based program I decided that if we're going to have a table anyway, just pre-calculate the entire table (with Mathematica) to 12-digit accuracy and eliminate the complication of extending the table on the fly.

The programs are distributed as individual programs, not a library. It probably isn't necessary to have all three programs on the calculator, instead, choose the one that best meets your needs. The descriptions below give the advantages and disadvantages of the three programs.

All of the mathematics in this document is taken from the references, except that I may have done some simple algebra for more efficient computation. The code is mine but I make no representation that it is completely optimized.

2: Summary of programs

Description	Program	Page
Use IBERNOULLI in Num Approx mode	IBERN	2
Find B_n with approximation methods	Bn	3
Find B_n with table lookup	BnT	7
Return a list of B_n values	BnTL	9
Return a list of B_{2n} values	BnTL2	10
Find ratio B_n/B_{n-2} , $n = 2, 4, 6, \dots$	BnRAT	14
Approximation constants for $\int \ln(\sin(x)) dx$	LSlcn	18
Approximate $\int \ln(\sin(x)) dx$ over range 0 to $\pi/2$	LnSin	19
Approximate $\int \ln(\sin(x)) dx$ over range 0 to π	LNSINI	20

3: IBERNOULLI with approximate results: program IBERN

The built-in function IBERNOULLI returns B_n as a ticked expression consisting of an exact fraction. The 50g AUR (*Advanced User's Reference Manual*)

claims that numeric mode must not be set (system flag -3 *clear*) and does not mention APPROX mode (system flag -105). However, experimentation shows that IBERNOULLI does return results with flag -3 *set* without raising error messages, but also silently clears flag -105 . The program IBERN shown below uses IBERNOULLI to find B_n (or a list of B_n), converts the result to approximate floating-point numbers, and restores Approx mode.

IBERN		(101 bytes, #CC55h)
« IBERNOULLI DUP TYPE 5 ==	Find B_n , test for list result	
« 1 « →NUM » DOLIST »	Convert list elements to numbers, or	
« →NUM »	convert single result to number	
IFTE		
-105 SF	Restore Approx mode	
»		

The AUR description for IBERNOULLI notes that “For numbers greater than about 40 the calculation can take a long time”. This is not an exaggeration. Table 1 shows approximate execution times for various values of n . The timing results suggest that IBERNOULLI may use a recursive method requiring the calculation of all the Bernoulli numbers up to B_n .

Table 1: IBERNOULLI Execution Time

n	Time (sec)	n	Time (sec)	n	Time (sec)
0	0.048	16	1.28	36	6.56
1	0.058	18	1.60	38	7.48
2	0.061	20	1.94	40	8.44
3	0.049	22	2.37	42	9.48
4	0.122	24	2.83	50	14.30
6	0.208	26	3.30	60	21.87
8	0.328	28	3.84	70	31.40
10	0.492	30	4.47	80	44.12
12	0.707	32	5.15	90	58.75
14	0.968	34	5.79	100	76.84

IBERNOULLI accuracy is good according to a few spot checks I did; it should be since the B_n are calculated as exact fractions then converted to floating-point numbers.

If you only occasionally need a few approximate Bernoulli numbers with small n , then IBERNOULLI or IBERN may be a good option.

4: B_n via approximation: program Bn

Program Bn calculates the Bernoulli numbers for $0 \leq n \leq 373$. $B_{374} = 1.97344362310E503$, which overflows the 50g maximum floating-point number, so B_{372} is the largest representable Bernoulli number. $B_{373} = 0$, so I included it in the allowable range for n .

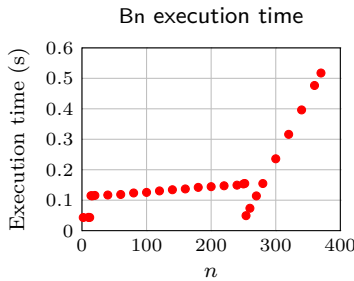
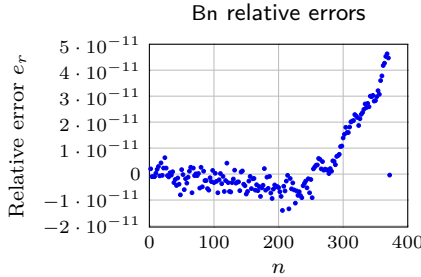
$$I \left| \begin{array}{c} n \end{array} \right| \Rightarrow \left| \begin{array}{c} B_n \end{array} \right|$$

Input n may be an explicit real integer or a ticked variable name containing a real integer. **Bn** is not listable but that can be accomplished with $\{n_1, n_2, \dots n_i\}$ « **Bn** » 1 DOLIST.

Bn uses one of three different approximations, depending on n . For $0 \leq n \leq 12$ a small table look-up is performed. B_1 is returned as a special case to simplify the index calculation for the table. B_{372} is also returned as a special case because the approximation used for large n overflows, even though B_{372} is representable. The approximation methods are detailed below in the code description.

The input n is validated to insure that it is a non-negative real number integer less than or equal to 373. If n is a ticked name it is evaluated and must result in a real integer.

The relative errors and execution time for **Bn** are plotted below. Since only about 180 non-zero Bernoulli numbers can be represented, it is feasible to find the errors for all the numbers. The relative error is defined as $e_r = (b_c - b_a)/b_a$; b_c is the calculated value and b_a is the ‘actual’ value determined with Mathematica. For $n < 200$ the errors are within $\pm 1 \cdot 10^{-11}$. The error increases for $n > 200$ as a result of the different approximation method used in that range. The execution time plot shows the same discontinuity at $n = 252$ caused by the change in method.



The code for **Bn** follows below. The ‘@’ character indicates comments, as consistent with Debug 4x source code. These comments are not in the actual source code. The small letters to the right of the code lines refer to notes following the listing.

Bn is structured as a single CASE statement. After handling ticked variable name inputs, the input is validated and special values of B_n are returned.

Bn	(657 bytes, #CFCDh)
« → n	a
«	
@ Evaluate and save ticked name	
n TYPE 6 == « n EVAL 'n' STO » IFT	b
@ Validate input n	
CASE	
n TYPE 0 ≠ THEN "Bn: n not real number" DOERR END	c
n FP 0 ≠ THEN "Bn: n not integer" DOERR END	d
n 0 < THEN "Bn: n < 0" DOERR END	e
n 373 > THEN "Bn: n > 373" DOERR END	f
@ Return special values $B_1, B_{372}, B_n = 0$	
n 1 == THEN -.5 END	g
n 372 == THEN -5.58475372909E499 END	h
n 2 MOD THEN 0 END	i
@ Find Bn for $0 \leq n \leq 12, n \neq 1$	
n 12 ≤ THEN	j
{1 .166666666667 -0.0333333333333 .0238095238095 -0.0333333333333 .0757575757576 -0.253113553114}	
n 2 / 1 + GET END	
@ Find Bn for $14 \leq n \leq 252$	
n 252 ≤ THEN	k
n ! -2 *	
@	
1 3 n NEG ^ -	
1 5 n NEG ^ - *	
1 7 n NEG ^ - *	
2 n ^ 1 - *	
-9.86960440109 n 2 / ^ *	
/	
END	
@ Find Bn for $n \geq 254$	
-2.9536826173E296	l
254 n FOR m	
m DUP 1 - * * -.0253302959106 *	
2 STEP	
@	
END	
@	
» »	

- a Save n to local variable
- b Evaluate then store n if it is a ticked name
- c Raise error message if n is not a real number
- d Raise error message if n is not an integer. An alternative would be to round n , as many built-in functions do.
- e Raise error message if n is less than 0.
- f Raise error message if n is greater than 373.

- g* Return $B_1 = -1/2$, which simplifies the subsequent calculation of the index into the look-up table below.
- h* Return 12-digit value of B_{372} ; this would result in intermediate overflow during the approximation calculation. This value was calculated with Mathematica.
- i* Return 0 for odd $n > 1$. $n \bmod 2$ returns *true* if n is odd.
- j* Return B_n for even $n \leq 12$ by table look-up. This reduces errors for small n which would result from the subsequent method.
- k* Return B_n for $14 \leq n \leq 252$. The method is described below.
- l* Return B_n for $254 \leq n \leq 372$, with the method described below.

The calculation at segment k is taken from p42 of reference [1], equation 4:8:3. An approximate value of B_n , \hat{B}_n , can be found from

$$\hat{B}_n = \frac{-2(n!)}{(-\pi^2)^{n/2}(2^n - 1)(1 - 3^{-n})(1 - 5^{-n})(1 - 7^{-n})}, \quad n = 2, 4, 6, \dots$$

While this result is intended to be used with a subsequent equation which gives an exact result, it turns out that the accuracy is 10 to 12 significant digits if $n \geq 14$. However, the built-in factorial function `!` overflows for $n > 252$, which limits the useful range. For the record, the subsequent equation (4:8:2 in the reference) which gives an exact result is

$$B_n = \frac{\text{Round}[(n+1)!! \hat{B}_n - \frac{1}{2}] + \frac{1}{2}}{(n+1)!!}$$

where $\text{Round}[x]$ rounds x to the nearest integer and $m!!$ is the double factorial of m .

To find the approximation we need to calculate the expression

$$(1 - 3^{-n})(1 - 5^{-n})(1 - 7^{-n})$$

and the code listing shows this calculation in the most obvious way. During development I tried this alternative:

```
1 { 3 5 7 } n NEG ^ - IILIST
```

which does save 23.5 bytes over the naive version. However, `Bn` execution time is increased by about 45 to 60% which surprised me, as I expected it to be faster than the explicit implementation. But if your priority is to reduce code size then this general method could be useful.

Another aspect of this calculation should be noted: in general, $1 - x^{-n}$ reduces to 1 when x^{-n} is small. For 3^{-n} this occurs when $n \geq 26$, for 5^{-n} when $n \geq 18$, and for 7^{-n} when $n \geq 15$. So, for $n \geq 26$ this calculation could be omitted, but since it takes about 48 mS to execute I left it in, rather than further complicate the program with another `CASE` clause or conditional test.

The code segment starting at *l* finds B_n for $n \geq 254$. The starting point is the relation from reference [5] (p5, (1.1.16)):

$$\begin{aligned} B_{2n} &= (-1)^{n-1} \frac{2(2n)!}{(2\pi)^{2n}} \zeta(2n) \\ &= (-1)^{n-1} \zeta(2n) 2 \prod_{m=1}^{2n} \frac{m}{2\pi} \end{aligned}$$

where $\zeta(x)$ is the Riemann zeta function. For integer $x \geq 38$, $\zeta(x) = 1.00000000000$, so this does not change the result. The term $(-1)^{n-1}$ sets the correct sign, but this is accounted for in the loop. Since we are starting the calculation at $m = 254$, not $m = 1$, we need to start the product calculation with

$$B_{252} = -2 \prod_{m=1}^{252} \frac{m}{2\pi} = -2.95368261730 \cdot 10^{296}$$

Each FOR loop iteration calculates the next value of B_n as

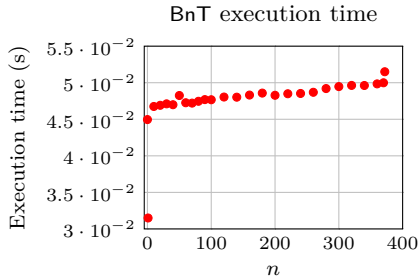
$$B_n = B_{n-2}m(m-1) \left(\frac{-1}{4\pi^2} \right)$$

The FOR loop increments the loop variable m by 2 to skip over the values $B_n = 0$, where n is odd. This results in calculating the product $m(m-1)$, and the factor $1/(2\pi)$ becomes $(1/(2\pi))^2 = 1/(4\pi^2)$. Furthermore, using the negative value $-1/(4\pi^2)$ keeps track of the correct sign of B_n as successive terms are calculated. This term can be reduced to the constant -0.0253302959106 .

Because this method calculates every value of B_n from B_{252} up to the desired value, there are two consequences. The first is that the execution time increases sharply as n increases. The second consequence is that the errors accumulate, as shown in the plot above.

5: B_n via table look-up: programs BnT, BnTL, BnTL2

Program Bn can be somewhat slow and not terribly accurate, so the program described in this section goes to the other extreme and returns all Bernoulli numbers from a large table. This may seem excessive, but, there are only 187 B_n which are representable in the 50g floating-point range, so the list isn't *that* big. Only B_1 is not in the table, to make calculating the table index easier, and it is returned as its exact value -0.5 . The remaining table values were calculated with Mathematica to 12-digit accuracy and imported into the BnT source file. An error plot for BnT is not necessary because the table values are accurate to 12 significant digits. The execution time is shown in figure 5.



The input validation is the same as Bn above, and BnT accepts ticked variable names as input. BnT is not listable, but see programs BnTL and BnTL2 below.

«

{ 1.166666666667 -3.33333333333E-2 2.38095238095E-2
-3.33333333333E-2 7.57575757576E-2 -.253113553114
1.166666666667 -7.09215686275 54.9711779449
-529.124242424 6192.12318841 -86580.2531136
1425517.16667 -27298231.0678 601580873.901
-15116315767.1 429614643061 -1.37116552051E13
4.88332318974E14 -1.92965793419E16 8.41693047574E17
-4.03380718541E19 2.11507486381E21 -1.20866265223E23
7.50086674608E24 -5.03877810148E26 3.65287764848E28
-2.84987693025E30 2.38654274997E32 -2.13999492572E34
2.05009757235E36 -2.09380059113E38 2.27526964885E40
-2.62577102862E42 3.21250821027E44 -4.15982781668E46
5.69206954820E48 -8.21836294198E50 1.25029043272E53
-2.00155832332E55 3.36749829154E57 -5.94709705031E59
1.10119103236E62 -2.13552595453E64 4.33288969866E66
-9.18855282417E68 2.03468967763E71 -4.70038339580E73
1.13180434455E76 -2.83822495707E78 7.40642489797E80
-2.00964548028E83 5.66571700508E85 -1.65845111541E88
5.03688599505E90 -1.58614682377E93 5.17567436175E95
-1.74889218402E98 6.11605199950E100 -2.21227769127E103
8.27227767988E105 -3.19589251114E108 1.27500822234E111
-5.25009230868E113 2.23018178942E116 -9.76845219310E118
4.40983619785E121 -2.05085708865E124 9.82144332798E126
-4.84126007982E129 2.45530888015E132 -1.28069268041E135
6.86761671047E137 -3.78464685820E140 2.14261012507E143
-1.24567271372E146 7.43457875510E148 -4.55357953046E151
2.86121128169E154 -1.84377235520E157 1.21811545362E160
-8.24821871853E162 5.72258779378E165 -4.06685305251E168
2.95960920646E171 -2.20495225652E174 1.68125970729E177
-1.31167362136E180 1.04678940095E183 -8.54328935788E185
7.12878213225E188 -6.08029314555E191 5.29967764248E194
-4.71942591687E197 4.29284137914E200 -3.98767449682E203
3.78197804194E206 -3.66142336837E209 3.61760902724E212
-3.64707726452E215 3.75087554365E218 -3.93458672964E221
4.20882111482E224 -4.59022962206E227 5.10317257726E230
-5.78227623037E233 6.67624821678E236 -7.85353076445E239
9.41068940671E242 -1.14849338735E246 1.42729587428E249
-1.80595595869E252 2.32615353077E255 -3.04957571155E258
4.06858060764E261 -5.52310313220E264 7.62772793964E267
-1.07155711197E271 1.53102008960E274 -2.22448916822E277
3.28626791907E280 -4.93559289560E283 7.53495712008E286
-1.16914851546E290 1.84352614678E293 -2.95368261730E296
4.80793212775E299 -7.95021250459E302 1.33527841874E306
-2.27764064960E309 3.94518403605E312 -6.93852577213E315
1.23889636758E319 -2.24554259917E322 4.13121317607E325
-7.71358134682E328 1.46153606684E332 -2.80990460623E335
5.48095712132E338 -1.08457328409E342 2.17698077565E345

```

-4.43199878612E348 9.15062565772E351 -1.91586735300E355
4.06725630354E358 -8.75422379104E361 1.91017368874E365
-4.22500132027E368 9.47195935255E371 -2.15214997328E375
4.95548577533E378 -1.15622594176E382 2.73340659765E385
-6.54686813533E388 1.58852491244E392 -3.90435480086E395
9.71993868609E398 -2.45076362105E402 6.25789209840E405
-1.61811355208E409 4.23652879522E412 -1.12304706820E416
3.01397178753E419 -8.18843757322E422 2.25191059134E426
-6.26841129204E429 1.76599084520E433 -5.03515443623E436
1.45277935646E440 -4.24149089013E443 1.25296600169E447
-3.74483004748E450 1.13231580670E454 -3.46351084594E457
1.07164338265E461 -3.35382447544E464 1.06159425715E468
-3.39842096922E471 1.10019250200E475 -3.60168637921E478
1.19223517043E482 -3.99034275178E485 1.35028180094E489
-4.61932544347E492 1.59752224397E496 -5.58475372909E499 }
→ n t
«
@
n TYPE 6 == « n EVAL 'n' STO » IFT
@
CASE
n TYPE 0 ≠ THEN "BnT: n not real number" DOERR END
n FP 0 ≠ THEN "BnT: n not integer" DOERR END
n 0 < THEN "BnT: n<0" DOERR END
n 373 > THEN "BnT: n>373" DOERR END
n 1 == THEN -.5 END
n 2 MOD THEN 0 END
@
't' n 2 / 1 + GET
END
» »

```

Lists of Bernoulli numbers

Some applications require a range of Bernoulli numbers, either all the numbers B_n or just the even-indexed numbers B_{2n} . Programs BnTL and BnTL2 both call BnT to return such lists of Bernoulli numbers.

Program BnTL returns a list of B_n from B_a to B_b , including the numbers with odd index. The program accepts five types of input arguments on stack level 1:

BnTL argument types		
Argument	Description	Result
n	Single real integer	B_n
$\{ a, b, \dots, z \}$	List of real integers	$\{ B_a, B_b, \dots, B_z \}$
$[a, b]$	Two-element vector, $a \leq b$	$\{ B_a, B_{a+1}, \dots, B_b \}$
$[a, b]$	Two-element vector, $a \geq b$	$\{ B_a, B_{a-1}, \dots, B_b \}$
'name'	Global variable containing any argument above	

BnTL will return a single Bernoulli number, but BnT should be used instead. If the input argument is a list, the corresponding B_n are returned in the same order as the arguments. This option is useful when a specific series of non-sequential B_n is needed.

If the argument is a two-element vector, a consecutive sequence of Bernoulli numbers is returned. A reversed list can be returned with $a \geq b$.

The indices n , a , b , and c are rounded to integers. BnTL ensures that the input argument is either a real number, a list, or a vector with two elements. Other error messages may be raised by BnT.

BnTL	(555 bytes, #BFADh)
«	
0. 0. "BnTL: Bad arg type" → n t em	
«	
DEPTH 1. < « "BnTL: <1 arg" DOERR » « 'n' ► » IFTE	
TYPE 6. ==	
«	
n VTYPE 't' STO	
0. t == 3. t == OR 5. t == OR	
« n → NUM 'n' STO »	
« em DOERR »	
IFTE	
»	
IFT	
n 0. RND 'n' ► TYPE 't' STO	
CASE	
t 0. == THEN n BnT END	
t 3. == THEN	
n SIZE { 2. } ≠ « "BnTL: Vector size ≠ 2" DOERR » IFT	
n V→ DUP2 > « -1. » « 1. » IFTE	
UNROT { } UNROT	
FOR k k BnT + OVER STEP NIP	
END	
t 5. == THEN n 1. 'BnT' DOLIST END	
em DOERR	
END	
» »	

BnTL is slow for large lists specified by $[a, b]$. Execution time for a list of ten elements $[0, 9]$ is about 0.46 seconds, for 100 elements $[1, 100]$ the time is 4.42 seconds, and for all 374 B_n , $[0, 373]$, the time is 25.4 seconds.

Often the Bernoulli numbers B_{2n} , numbers with only even indices, are needed. Program BnTL2 accomplishes this.

$$1 \left| N \right| \Rightarrow \left| \{B\} \right|$$

BnTL2 takes a single argument N from stack level 1, which may be a 2-element vector $[a, b]$ or a ticked name containing that vector. a and b are rounded to integers and both must be even numbers. If $a < b$ then the

list of Bernoulli numbers $\{ B_a, B_{a+2}, \dots, B_b \}$ is returned. If $a > b$ then the list is $\{ B_a, B_{a-2}, \dots, B_b \}$.

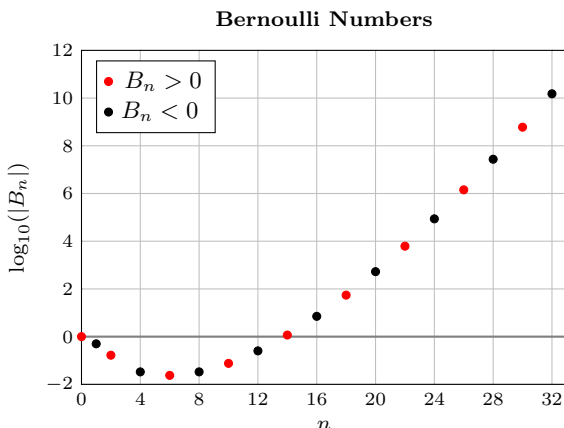
BnTL2 will raise error messages if there is no stack argument, if the argument is not a vector, if the vector does not have exactly two elements, and if both arguments are not even numbers. Other errors may be raised by BnT, which is called by BnTL2.

BnTL2	(379 bytes, #AE4h)
«	
DEPTH 1. < « "BnTL2: <1 arg" DOERR » IFT	
DUP TYPE 6. ==	
«	
DUP VTYPE 3. ≠	
« DROP "BnTL2: arg not vector" »	
« →NUM »	
IFTE	
»	
IFT	
DUP SIZE {2.} ≠ « "BnTL2: Vector size ≠ 2" DOERR » IFT	
0. RND DUP	
AXL 2. MOD « OR » STREAM	
« DROP "BnTL2: arg(s) not even" DOERR » IFT	
V→	
DUP2 > « -2. » « 2. » IFTE	
UNROT {} UNROT	
FOR k	
k BnT + OVER	
STEP	
NIP	
»	

6: About Bernoulli numbers

All of the Bernoulli numbers except B_0 are fractions. For odd numbers $n \geq 3$, $B_n = 0$. The Bernoulli numbers alternate in sign for $n \geq 2$, such B_n is positive for odd $n/2$ and negative for even $n/2$. The magnitude of B_n increases rapidly and values greater than B_{372} overflow the maximum 50g floating-point number. Bernoulli numbers can be defined for $n < 0$ as well as non-integer and complex n , but this seems to be very rarely needed. The first few non-zero values of B_n are listed below.

Exact Bernoulli numbers			
n	B_n	n	B_n
0	1	16	$-3617/51$
1	$-1/2$	18	$43867/798$
2	$1/6$	20	$-174611/330$
4	$-1/30$	22	$854313/138$
6	$1/42$	24	$-236364901/2730$
8	$-1/30$	26	$8553103/6$
10	$5/66$	28	$-23749461029/870$
12	$-691/2730$	30	$8615841276005/14332$
14	$7/6$	32	$-7709321041217/510$



The Bernoulli numbers are ubiquitous in number theory, combinatorics, probability, analysis, and numerical methods. They were first explicitly listed as defined constants by Jaques (often anglicized as James) Bernoulli (1654–1705) in his published *Ars Conjectandi* (1713), an early book on probability. The motivating application was finding finite sums of the form $\sum_{k=1}^n k^p$. Euler further developed properties and applications of the Bernoulli numbers, and incidentally called them ‘Bernoulli’ numbers.

In the equations below, the subscript $2n$ in B_{2n} indicates Bernoulli numbers with even subscripts. For example, the Bernoulli number B_{10} , with the convention used in this document, is equivalent to $B_{2,5}$, with $n = 5$.*

Definitions of B_n

The Bernoulli numbers can be defined in numerous ways, for example, with a generating function expression

$$\frac{t}{e^t - 1} = \sum_{n=0}^{\infty} B_n \frac{t^n}{n!} \quad (6.1)$$

*This *is* confusing, but it results from historical precedents and conventions used by different mathematicians and authors over the centuries. There are better alternative notations, but I won’t use them here just to avoid even more confusion.

or the summation

$$B_{2n} = \sum_{k=2}^{2n+1} \frac{(-1)^{k-1}}{k} \binom{2n+1}{k} \sum_{j=1}^{k-1} j^{2n}, \quad n = 1, 2, 3, \dots \quad (6.2)$$

or as the integral

$$B_{2n} = (-1)^{n+1} \frac{4n}{1 - 2^{1-2n}} \int_0^\infty \frac{t^{2n-1}}{e^{2\pi t} + 1} dt, \quad n = 1, 2, 3, \dots \quad (6.3)$$

or with respect to the Riemann zeta function $\zeta(n)$ (which will be relevant later)

$$B_n = (-1)^{(n+2)/2} \frac{2n!}{(2\pi)^n} \zeta(n), \quad n = 2, 4, 6, \dots \quad (6.4)$$

$$B_{2n} = (-1)^{n-1} \frac{2(2n)!}{(2\pi)^{2n}} \zeta(2n), \quad n = 1, 2, 3, \dots \quad (6.5)$$

$$= (-1)^{n-1} \zeta(2n) \prod_{m=1}^{2n} \frac{m}{2\pi} \quad (6.6)$$

where

$$\zeta(z) = \sum_{k=1}^{\infty} \frac{1}{k^z} \quad (6.7)$$

and finally as a recurrence relation

$$B_n = -n! \sum_{j=0}^{n-1} \frac{B_j}{j!(n+1-j)!}, \quad n = 1, 2, 3, \dots \quad (6.8)$$

which (potentially) enables computation of B_n from preceding values, given that $B_0 = 1$. The built-in `IBERNOULLI` function may use this recurrence relation (or some other) to find exact values for B_n .

These definitions are only a small sample of possible definitions of the Bernoulli numbers. Chapter 24 of reference [2] is the most comprehensive list of definitions, relations, and properties of the Bernoulli numbers of which I am aware.

Asymptotic relations

Asymptotic expressions for a function $f(x)$ are useful to understand the function behavior as x becomes very large, and can be used for computation if the approximation is accurate enough at small enough values of x . The expression $f(x) \sim g(x)$ means that $f(x)$ approaches $g(x)$ as x approaches infinity. The two following asymptotic expressions approximate B_n or B_{2n} .

$$B_n \sim (-1)^{(n+2)/2} \left(\frac{n}{2\pi e} \right)^n \sqrt{8\pi n} \quad (6.9)$$

$$B_{2n} \sim (-1)^{n-1} 4\sqrt{\pi n} \left(\frac{n}{\pi e} \right)^{2n} \quad (6.10)$$

For numerical evaluation (6.10) can be written as

$$B_{2n} \sim (-1)^{n-1} a \sqrt{n} (bn)^{2n} \quad (6.11)$$

with $a = 4\sqrt{\pi} = 7.08981540362$ and $b = 1/(\pi e) = 0.117099663049$. With input $2n$ in stack level 1, this could be coded as

```
« DUP 2 / .117099663049 OVER * ROT ^
OVER √ * 7.08981540362 * -1 ROT 1 - ^ * »
```

This equation lacks accuracy for our values of n , for example, for B_{372} it returns -5.58350280971E499 while the actual value is -5.58475372909E499; so only a few digits are correct.

Equations (6.9) and (6.10) above are the asymptotic equations given in most sources, but reference [12] gives several more accurate approximations, especially at smaller values of n . Two of these are

$$B_n \sim (-1)^{n/2+1} \cdot \exp \left[\left(\frac{1}{2} + n \right) \ln(n) + \left(\frac{1}{2} - n \right) \ln(\pi) + \left(\frac{3}{2} - n \right) \ln(2) - R(n) \right] \quad (6.12)$$

$$R(n) = n \left(1 - \frac{1}{12} \left(1 - \frac{1}{30} \left(1 - \frac{2}{7} n^{-2} \right) n^{-2} \right) n^{-2} \right) \\ B_{2n} \sim (-1)^{n-1} 4\sqrt{n\pi} \left(\frac{n(480n^2 + 9)}{\pi e(480n^2 - 1)} \right)^{2n} \quad (6.13)$$

Equation (6.13) can be rewritten for slightly more efficient evaluation as

$$B_{2n} \sim (-1)^{n-1} a \sqrt{n} \left(bn \left(\frac{10}{480n^2 - 1} + 1 \right) \right)^{2n} \quad (6.14)$$

with $a = 4\sqrt{\pi} = 7.08981540362$ and $b = 1/(\pi e) = 0.117099663049$, and could be coded as

```
« DUP 2 / 10 OVER SQ 480 * 1 - / 1 +
OVER .117099663049 * * ROT ^ OVER
√ * 7.08981540364 * -1 ROT 1 - ^ * »
```

Luschny writes in [12] that this approximation can give 18 correct digits for $n \geq 38$, but only if enough digits are carried in the intermediate calculations. With the 50g we have only 12 digits and the code above gives $B_{252} = -2.95368261995E296$ and $B_{372} = -5.58475373398E499$, with the correct digits underlined; so only about eight digits are correct.

Can we do any better with equation (6.12)? Unfortunately not. Equation (6.12) was coded as follows for testing.

```
« DUP -1 OVER 2 / 1 + ^ UNROT .5 +
OVER LN * OVER NEG .5 OVER +
1.14472988585 * SWAP 1.5 + .69314718056 * + +
OVER SQ INV -.285714285714 OVER * 1 + OVER *
-.0333333333333 * 1 + * -.0833333333333 * 1 +
ROT * - EXP * »
```

This gives $B_{100} = -2.83822495937E78$ and $B_{372} = -5.58475372620E499$, with the correct digits underlined.

The ratio B_n/B_{n-2}

An asymptotic expression for the ratio of two Bernoulli numbers is

$$\frac{B_n}{B_{n-2}} \sim \frac{-n(n-1)}{4\pi^2}, \quad n = 2, 4, 6, \dots \quad (6.15)$$

which can be used to quickly and accurately calculate the ratio of the two Bernoulli numbers without finding the numbers themselves. For $n \geq 46$ the relative error of the approximation is less than $4.3 \cdot 10^{-14}$. The ratio can be found without overflow for B_n much greater than $n = 374$. For $n < 46$ the ratio is calculated directly from the two Bernoulli numbers. The program below, BnRAT, implements these ideas. No input validation is done but the IFERR construction traps all errors. BnRAT overflows for $n \geq 10^{249}$.

n	\Rightarrow	B_n/B_{n-2}
<div style="display: flex; justify-content: space-between;"> BnRAT (145.5 bytes, #F66h) </div>		
<pre> « ERR0 IFERR DUPDUP 46 < « BnT SWAP 2 – BnT / » « 1 – * –2.53302959106E–2 * » IFTE THEN "BnRAT: " ERRM + DOERR END » </pre>		

Bernoulli polynomials

The Bernoulli numbers are related to the Bernoulli polynomials $B_n(x)$; one identity uses B_n to generate the Bernoulli polynomials:

$$B_n(x) = \sum_{j=0}^n \binom{n}{j} x^{n-j} B_j \quad (6.16)$$

Evaluating $B_n(0)$ gives B_n .

Application: sums of powers of integers

Bernoulli numbers can be used to derive closed-form expressions for the sums of powers of integers, namely

$$S = \sum_{k=1}^n k^p = \frac{(B + n + 1)^{[p+1]} - B^{[p+1]}}{p + 1} \quad (6.17)$$

where the notation $q^{[r]}$ has a special interpretation: raise the expression q to the power r , then replace all terms of the form B^m with Bernoulli number B_m . For example, let $p = 2$ and $n = 5$ then

$$(B + n + 1)^{[p+1]} = (B + 5 + 1)^{2+1} = B^3 + 18B^2 + 108B + 216$$

which is interpreted as

$$B_3 + 18B_2 + 108B_1 + 216$$

For a given value of p we can find an explicit expression for the sum as a function of n . Leaving n undefined in the example above gives

$$S = \frac{(1 + B + n)^{[3]} - B^{[3]}}{3}$$

which, after expanding and collecting, gives

$$S = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$$

which can be efficiently evaluated with the `PEVAL` command with the coefficient vector `[0.333333333333 0.5 0.166666666667 0]`.

Equation (6.17) can be also be used to find partial sums of the form

$$S_p = \sum_{k=m}^n k^p \quad \text{since} \quad S_p = \sum_{k=1}^n k^p - \sum_{k=1}^{m-1} k^p$$

Making the substitutions and simplifying gives

$$S_p = \frac{(B + n + 1)^{[p+1]} - (B + m)^{[p+1]}}{p + 1} \quad (6.18)$$

For example, if $p = 2$, then (6.18) gives

$$S_p = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6} - \frac{m^3}{3} + \frac{m^2}{2} - \frac{m}{6}$$

Application: trigonometric series and integrals

Bernoulli numbers can be used to define trigonometric functions as infinite sums; the following examples are taken from [4].

$$\begin{aligned} \tan(z) &= \sum_{k=1}^{\infty} \frac{2^{2k}(2^{2k} - 1)|B_{2k}|}{(2k)!} z^{2k-1}, & z^2 < \frac{\pi^2}{4} \\ \cot(z) &= \frac{1}{z} - \sum_{k=1}^{\infty} \frac{2^{2k}|B_{2k}|}{(2k)!} z^{2k-1}, & z^2 < \pi^2 \\ \csc(z) &= \frac{1}{\sin(z)} = \frac{1}{z} + \sum_{k=1}^{\infty} \frac{2(2^{2k-1} - 1)|B_{2k}|}{(2k)!} z^{2k-1}, & z^2 < \pi^2 \\ \tanh(z) &= \sum_{k=1}^{\infty} \frac{2^{2k}(2^{2k} - 1)B_{2k}}{(2k)!} z^{2k-1}, & z^2 < \frac{\pi^2}{4} \\ \operatorname{csch}(z) &= \frac{1}{\sinh(z)} = \frac{1}{z} - \sum_{k=1}^{\infty} \frac{2(2^{2k-1})B_{2k}}{(2k)!} z^{2k-1}, & z^2 < \pi^2 \end{aligned}$$

Similarly the Bernoulli numbers appear in integrals involving trigonometric functions, such as

$$\begin{aligned}\int \ln[\sin(x)] dx &= x \ln(x) - x - \sum_{n=1}^{\infty} \frac{2^{2n-1} |B_{2n}|}{n(2n+1)!} x^{2n+1}, & 0 < x < \pi \\ \int \frac{x}{\sinh(x)} dx &= x + \sum_{n=1}^{\infty} (-1)^n \frac{2(2^{2n-1} - 1) B_{2n}}{(2n+1)!} x^{2n+1}, & x^2 < \pi^2 \\ \int_0^{\infty} \frac{x^{2n-1}}{\sinh(x)} dx &= \frac{2^{2n} - 1}{2n} \pi^{2n} |B_{2n}|, & n = 1, 2, 3, \dots\end{aligned}$$

An asymptotic expression for the digamma function $\Psi(x)$ uses the Bernoulli numbers:

$$\Psi(x) \sim \ln(x) - \sum_{j=1}^{\infty} \frac{B_j}{j x^j}$$

Application: the Euler-MacLaurin formula

Some authors write that the Euler-MacLaurin formula is the ‘most important’ application of the Bernoulli numbers, that may be; in any event the formula can be written (reference [2], p63) as

$$\begin{aligned}\sum_{j=a}^n f(j) &= \int_a^n f(x) dx + \frac{1}{2} f(a) + \frac{1}{2} f(n) \\ &+ \sum_{s=1}^{m-1} \frac{B_{2s}}{(2s)!} \left(f^{(2s-1)}(n) - f^{(2s-1)}(a) \right) \\ &+ \int_a^n \frac{B_{2m} - \tilde{B}_{2m}(x)}{(2m)!} f^{(2m)}(x) dx\end{aligned}\tag{6.19}$$

where a , m , and n are integers with $n > a$ and $m > 0$. $f^{(k)}(x)$ is the k ’th derivative of $f(x)$ which must be absolutely integrable over $[a, n]$. $\tilde{B}_k(x)$ is the k ’th Bernoulli polynomial evaluated with $B_k(x - \lfloor x \rfloor)$; $\lfloor x \rfloor$ is the floor function. However, this formula may also be defined in other ways, as shown in the references.

Broadly, this formula finds extensive use in evaluating a sum as an integral or an integral as a sum, either analytically or for numeric approximation. The rapid growth of the Bernoulli numbers often results in a series which converges very quickly, which is always good for numerical approximation. Reference [15] shows some examples.

A: Evaluating $\int \ln(\sin(x)) dx$

As mentioned in section 6 above, Bernoulli numbers can be used to define the integral

$$\int \ln[\sin(x)] dx = x \ln(x) - x - \sum_{n=1}^{\infty} \frac{2^{2n-1} |B_{2n}|}{n(2n+1)!} x^{2n+1}, \quad 0 < x < \pi$$

with x in radians. This appendix describes programs implementing this relationship to numerically approximate the definite integral

$$\int_a^b \ln[\sin(x)] dx \approx f(x) \Big|_{x=a}^{x=b}$$

with

$$\begin{aligned} f(x) &= x \ln(x) - x - \sum_{n=1}^k \frac{2^{2n-1} |B_{2n}|}{n(2n+1)!} x^{2n+1} \\ &= x \ln(x) - x - \sum_{n=1}^k c_n x^{2n+1} \end{aligned}$$

The approximation (usually) gives accurate results much more quickly than using the the built-in numerical integration command `f`.

Incidentally, the integral can be expressed in closed form as

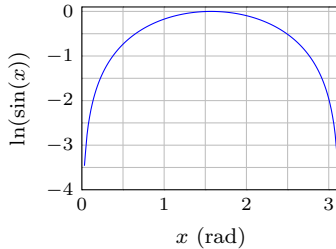
$$\int \ln[\sin(x)] dx = x \left[\ln(\sin(x)) - \ln(1 - e^{2ix}) \right] + \frac{i}{2} \left(x^2 + \text{Li}_2(e^{2ix}) \right)$$

where $\text{Li}_2(z)$ is the dilogarithm function, defined here as

$$\text{Li}_2(z) = \sum_{k=1}^{\infty} \frac{z^k}{k^2}$$

Although the integral expression contains the imaginary unit i , the imaginary parts of the result cancel out when a and b are real and the integral is evaluated.

The plot below shows the integrand function $\ln(\sin(x))$. The function is horizontally symmetric about $x = \pi/2$ and has vertical asymptotes at $x = 0$ and $x = \pi$.



To use the approximation $f(x)$ we need to find the coefficients c_n and determine a value for k which gives sufficient accuracy over the domain of the integrand. Since the integrand function is symmetric about $x = \pi/2$, I tested implementations of $f(x)$ (in Mathematica) for the range $0 < x < \pi/2$ and found that $k = 15$ seemed to give good results.

Although I used Mathematica, the c_n coefficients can be calculated with the 50g using the program `BnT` for the Bernoulli numbers. Program `LSIn` finds c_n for index n . For n up to 15 the returned c_n may be wrong by a few digits in the last place. The maximum n without overflow is 126. `LSIn` does no input validation or error handling.

$$I \left| \begin{array}{c} n \end{array} \right| \Rightarrow \left| \begin{array}{c} c_n \end{array} \right|$$

LSlcn	(71 bytes, #789Eh)
«	
DUP 2. * 2. OVER 1. - ^	Find $A = 2^{2n-1}$
OVER 1. + FACT /	Find $B = A/(2n-1)!$
ROT / SWAP BnT ABS *	Find $c_n = (B/n) B_{2n} $
»	

To find the integral value we use $f(b) - f(a)$. Program LnSin accomplishes this for $0 \leq x \leq \pi/2$. No input validation or error trapping is done and RAD mode must be set. The returned result has the correct sign, but if $a > b$ then same result is returned with the sign reversed, according to the usual definition.

LnSin has not been tested for accuracy, but the several cases I tried indicate that the results seem to be accurate to 11 or 12 digits until both a and b approach $\pi/2$. Fewer significant digits are returned in this case because $f(a)$ is close to $f(b)$ and destructive cancellation occurs from evaluating $f(b) - f(a)$.

$$\begin{array}{c} 2 \\ 1 \end{array} \left| \begin{array}{c} a \\ b \end{array} \right| \Rightarrow \left| \begin{array}{c} I \end{array} \right|$$

LnSin	(389.5 bytes, #A07Dh)
«	
[2.61848968056E-18 0. 2.95990336617E-17 0. 3.37906230773E-16 0. 3.90195113664E-15 0. 4.56648865593E-14 0. 5.42979790586E-13 0. 6.58124167158E-12 0. 8.16713558455E-11 0. 1.04409275485E-9 0. 1.38743864154E-8 0. 1.94343628687E-7 0. 2.93944738389E-6 0. 5.0390526581E-5 0. 1.11111111111E-3 0. .055555555555555555 0. 0. 0.]	c_n coefficients vector, saved in $\leftarrow c$
« DUPDUP LN * OVER - $\leftarrow c$ ROT PEVAL - »	$f(x)$ program, saved in p
$\rightarrow \leftarrow c$ p	Save local variables
«	
PUSH	Save system flags
-20. CF	Set silent underflow to 0
p EVAL SWAP p EVAL -	$I = f(b) - f(a)$
POP	Restore system flags
» »	

The coefficient list (saved in local variable $\leftarrow c$) is structured for use by PEVAL which expects the coefficients in descending power order. The alternating zeros in the list are present because only odd powers of x are summed. The first sum term is $c_1 x^3$ so the last three coefficients are zero.

The code to evaluate $f(x)$ is saved in local variable p since it is used twice. The coefficient vector must be stored in a compiled local variable so that it is visible to p. System flag -20 is set to *clear* so that underflows

from PEVAL do not raise an error message. PUSH and POP save and restore the user's setting for flag -20.

To find the integral for $a = .01$ and $b = 1$, use

$$.01 \ 1 \ \text{LnSin} \implies -0.726406130019$$

The correct result is the same except that the last two digits are 20 instead of 19. An example of the loss of precision is given with $a = 1.5$ and $b = 1.57$. LnSin returns $-5.916953 \cdot 10^{-5}$; the correct result to 12 digits is $-5.91695320691 \cdot 10^{-5}$.

The accuracy of LnSin degrades rapidly for $x > \pi/2$, but we can use the symmetry of the integrand to manipulate the integration limits to find the integral when one or both of the integration bounds are greater than $\pi/2$. Suppose that we want to integrate

$$I = \int_{\alpha}^{\beta} \ln[\sin(x)] dx$$

then there are four possibilities to consider:

Case	α	β	Method and limits a, b
1	$\alpha \leq \pi/2$	$\beta \leq \pi/2$	Use LnSin with $a = \alpha, b = \beta$
2	$\alpha \leq \pi/2$	$\beta > \pi/2$	$I = \int_{\alpha}^{\pi/2} + \int_{\pi-\beta}^{\pi/2}$
3	$\alpha > \pi/2$	$\beta < \pi/2$	Implies $\alpha > \beta$, see below
4	$\alpha > \pi/2$	$\beta > \pi/2$	Use $a = \pi - \beta, b = \pi - \alpha$

If $\alpha > \beta$ we return $-\int_{\beta}^{\alpha}$, corresponding to the usual definition. This is accomplished by testing the input limits a and b , swapping them if $a > b$, and setting local variable swap, which takes care of case 3.

$$\begin{array}{|c|} \hline 2 \mid \alpha \\ \hline 1 \mid \beta \\ \hline \end{array} \implies \begin{array}{|c|} \hline I \\ \hline \end{array}$$

LNSINI	(625 bytes, #475Bh)
«	
1)	DEPTH 2. < « "LNSINI: <2 ARGS" DOERR » IFT
2)	EVAL DUP TYPE ROT EVAL DUP TYPE ROT OR
3)	« DROP2 "LNSINI: Arg(s) not real number" DOERR » IFT
4)	1.5707963268 0. → b a pd2 swap
«	
5)	a b 2. →LIST DUP
6)	0. < SWAP π > OR «OR» STREAM
7)	« "LNSINI: Arg(s) <0 or > π " DOERR » IFT
8)	a b > « 1. 'swap' STO a b 'a' STO 'b' STO » IFT
CASE	
9)	a b == THEN 0. END
10)	a pd2 ≤ b pd2 ≤ AND THEN a b LnSin END
11)	a pd2 ≤ b pd2 > AND THEN a pd2 LnSin π b - pd2 LnSin + END
12)	a pd2 > b pd2 > AND THEN π b - π a - LnSin END
END	
13)	swap « NEG » IFT
»»	

For variety and illustration `LNSINI` does some input argument validation. Line 1 verifies that at least two arguments are on the stack. Lines 2 and 3 evaluate the arguments if they are ticked names, and verify that both are real numbers. Line 4 saves the local variables; `pd2` is $\pi/2$. Lines 5 to 7 verify that both arguments are in the range zero to π . Line 8 swaps α and β if necessary and sets `swap` to `true`. Line 9 returns zero for the integral if the integration limits are equal. Lines 10, 11, and 12 find the integral for cases 1, 2, and 4, respectively. Program `LnSin`, described above, is used to perform the integrations. Line 13 negates the integral if the integration limits were swapped.

B: References

This bibliography consists of references I used to develop the code in this note, references which are good introductions to the Bernoulli numbers, and additional sources of interesting aspects and background. References [2] and [10] both have good bibliographies for additional reading. Reference [14] is an exhaustive bibliography up to 2007 which can be searched by author.

- [1] *An Atlas of Functions*, 2e, KEITH OLDHAM, JAN MYLAND, JEROME SPANIER, Springer, 2009
Chapter 4 discusses the Bernoulli numbers and Chapter 19 the Bernoulli polynomials. Equation 4:8:3 is used by program `Bn` to find the Bernoulli numbers for $14 \leq n \leq 252$. A few definitions, relationships and limits are given.
- [2] *NIST Handbook of Mathematical Functions*, FRANK W. J. OLVER, DANIEL W. LOZIER, RONALD F. BOISVERT, CHARLES W. CLARK, eds., Cambridge University Press, 2010.
Chapter 24 documents the Bernoulli and Euler numbers and polynomials; dozens of relationships and identities are given, along with many references to additional sources. This book is the successor to the famous *Handbook of Mathematical Functions* by Abramowitz and Stegun. The handbook is available at <https://dlmf.nist.gov>, with the Bernoulli numbers chapter at <https://dlmf.nist.gov/24>.
- [3] *Tables of Integrals and Other Mathematical Data*, 4e, HERBERT BRISTOL DWIGHT, The Macmillan Company, 1961.
Several integrals and series used in the Applications section of this paper were taken from this handbook. Some care is required because, oddly, Dwight defines all Bernoulli numbers as positive and omits B_0 . Dwight uses the notation B_n with $n = 1, 2, \dots$; $B_1 = 1/6$, $B_2 = 1/30$, so the Bernoulli numbers equal to zero are omitted. The formulas from this handbook were modified for consistency with the notation I used.
- [4] *Table of Integrals, Series and Products*, 8e, I. S. GRADSHTEYN, I. M. RYZHIK, DANIEL ZWILLINGER (ed.), VICTOR MOLL (scientific ed.), Academic Press, 2014.
This handbook is the latest edition of Gradshteyn and Ryzhik's well-known tables. Zwillinger has done his usual careful job in updating earlier editions, in particular to use more modern notation and conventions for special functions. Of course there is a very brief summary

of Bernoulli numbers, but more interesting is the occurrence of B_n in sums and integrals. Compared to earlier versions, the print is quite legible, the paper quality is good, and the binding seems reliable so far. Considering all this and with over 1100 pages, the price is surprisingly and pleasantly low. Some idea of the reader engagement with this book is that acknowledgments are given to over 650 people who contributed corrections and addenda.

- [5] *Computation of Special Functions*, SHANJIE ZHANG, JIANMING JIN, Wiley-Interscience, 1996.

Chapter 1 describes properties and a few applications of the Bernoulli numbers. Several programs are given for the calculation of B_n and the program BERNOB (p6) was the starting point for program Bn. Also has a table of B_n from $n = 0$ to $n = 80$ with 13 significant digits.

- [6] *Mathematical Methods for Physicists*, 2e, GEORGE ARFKEN, Academic Press, 1970.

This reference gives a brief, efficient description of definitions of Bernoulli numbers and Bernoulli polynomials. The Euler-Maclauren integration formula is developed and the connection between the Bernoulli numbers and the Riemann zeta function is summarized.

- [7] *The Bernoulli Numbers: A Brief Primer*, NATHANIEL LARSON, <https://www.whitman.edu/Documents/Academics/Mathematics/2019/Larson-Balof.pdf>, 2019

This thorough 47-page introductory paper summarizes all the common properties and applications of the Bernoulli numbers. Section 8 derives the Euler-MacLaurin summation formula and section 9 gives application examples.

- [8] *Introduction on Bernoulli's numbers*, PASCAL SEBAH, XAVIER GOURDON, <https://scipp.ucsc.edu/~haber/archives/physics116A10/bernoulli.pdf>

Another brief introduction (14 pages) to the Bernoulli numbers, but with some brief historical notes. Discusses the relation to the zeta function, Clausen-von Staudt's theorem, and the Euler-Maclaurin formula, as well as Fermat's last theorem. Section 6 gives the series for $\cot(z)$, $\tanh(z)$, and $\tan(z)$, but numerical tests show that the formula for $\tan(z)$ (at least) is incorrect.

- [9] *The On-Line Encyclopedia of Integer Sequences* (OEIS), oeis.org. Searching for 'bernoulli' gives 1780 results at the time of this writing, including the denominator of B_n (A027642) and the numerator of B_n (A027641).

- [10] *Bernoulli Number*, ERIC W. WEISSTEIN, Mathworld, <https://mathworld.wolfram.com/BernoulliNumber.html>

A long Mathworld entry including several definitions and properties of the Bernoulli numbers, with many number theory results and a long list of references. One curiosity is that the largest computed Bernoulli number is B_{108} , by D. Harver in 2008.

- [11] *Bernoulli number*, https://en.wikipedia.org/wiki/Bernoulli_number
A rather typical Wikipedia page for mathematical topics. All of the

common definitions and relations are given, but many of the equations are duplicated for the two possible values for B_1 ; see reference [13] below for more details. A brief history is given. Many applications of the Bernoulli numbers are very briefly summarized including combinatoric results. In general the Wikipedia article could be somewhat useful, but other sources are better.

- [12] *An approximation of the Bernoulli numbers*, PETER LUSCHNY,
<https://www.luschny.de/math/primes/bernincl.html>
 This short paper gives compact expressions for asymptotic approximations of the Bernoulli numbers, which do not involve factorials or the zeta function. The approximations are more accurate than the standard expressions given in equations (6.9) or (6.10); unfortunately for our application they require more than the 12 digits of precision that are available on the 50G in Num Approx mode.
- [13] *The Bernoulli Manifesto*, PETER LUSCHNY, updated 2021,
<http://luschny.de/math/zeta/The-Bernoulli-Manifesto.html>
 Luschny mounts a passionate, extensive argument that the Bernoulli numbers have been mis-defined, one aspect of which is the sign of B_1 : $B_1 = 1/2$ or $B_1 = -1/2$? But there is much more to his argument than that. From the document: “*My primary concern is to find a substantive definition of the Bernoulli numbers. A definition that, if it is to be, even goes beyond the paradigmatic example given by Bernoulli himself and embeds naturally into the wider field of mathematics.*”
- [14] *A Bibliography of Bernoulli Numbers*, KARL DILCHER,
 ILJA SH. SLAVUTSKII, <https://www.mscs.dal.ca/~dilcher/bernoulli.html>
 A massive bibliography of Bernoulli number papers and books dating from 1713 to 2007; in 2005 there were 2970 entries by 1493 authors. The complete file can be opened as a web page for searching. From the description: *Basically, this bibliography contains papers that (i) deal primarily with Bernoulli numbers and related number and polynomial sequences, and (ii) those in which these numbers and polynomials are applied. ... Also, the focus of this bibliography is somewhat slanted toward number theory, ...*
- [15] *Gamma: Exploring Euler’s Constant*, JULIAN HAVIL, Princeton University Press, 2003.
 Havil discusses the Bernoulli numbers and the Euler-MacLaurin formula in chapter 10, including a few examples of applying the formula. His development of the Bernoulli numbers is limited to their use in summing powers of integers, the Euler-MacLaurin formula, and calculation of Euler’s γ constant (unsurprisingly, for the latter). Havil sprinkles in interesting historical details as his exposition develops.
- [16] *A Source Book in Mathematics*, DAVID EUGENE SMITH, Dover, 1959.
 The section *Bernoulli on “Bernoulli Numbers”* (p85) in chapter 1 reprints an english translation of Bernoulli’s description and development of the Bernoulli numbers from his *Ars Conjectandi*, with a short background preface by Smith.