Splitting the Bernoulli Numbers

Michael G. Kaarhus

December 25, 2013

Abstract

Here I present my first five Bernoulli Number conjectures. Their effect is to split the non-zero Bernoulli Numbers $\geq$ *Bernoulli*$_2$ into one or the other of two populations. To my knowledge, I am the discoverer of these two populations; no one else has conjectured them. None of the sequences I introduce here were listed at OEIS when I published this.

My Bernoulli-Number-splitting conjectures enable me to obtain the residues modulo 6 of the absolute values of non-zero Bernoulli numerators, without actually calculating the numerators! Maybe that doesn't sound very exciting to you, but maybe you haven't tried calculating very many non-zero Bernoulli numerators. They increase very quickly into huge integers! Using the von Staudt-Clausen Theorem, I need only n to do this.

Conjecture 1

The absolute values of all non-zero Bernoulli numerators are $\equiv \{1 \ \mathit{or} \ 5\}$ mod 6. I checked this conjecture up to *Bernoulli*$_{2000}$ with the following *Maxima* program. It found no exceptions:

```
float(true)$ n:0$ for p:1 thru 1000 step 0 do
(n:n+2, b:bern(n), m:mod(abs(num(b)),6), if (m=5 or m=1) then p:p+1 else
(print("exception at ", n, " abs val mod 6 = ", m), p:4000))$
print("checked to Bern_", n)$
```

Conjecture 2

Those numerators with absolute values $\equiv$ 1 mod 6 have a sequence of denominators different from the sequence of the denominators of numerators having absolute values $\equiv$ 5 mod 6. Neither of these two sequences of denominators have any elements in common. I wrote these *Maxima* programs to help me make Conjecture 2:

```
float(true)$ n:0$ c:6$ for p:1 thru 1000 step 0 do
(n:n+2, b:bern(n), m:mod(abs(num(b)),c),
if m=1 then (d:denom(b), print(p,", ",d), p:p+1))$
```

```
float(true)$ n:0$ c:6$ for p:1 thru 1000 step 0 do
(n:n+2, b:bern(n), m:mod(abs(num(b)),c),
if m=5 then (d:denom(b), print(p,", ",d), p:p+1))$
```

I saved those *Maxima* lists as bfiles. Later I used a different program to make bfiles containing 10k elements. To check the bfiles to see if they share any elements, I wrote a program that makes lists of just the denominators. It removes duplicates from those lists, compares the lists, and returns their intersection. This is my program's output:

```
@pop1 has 10000 elements.  @pop2 has 10000 elements
@uni has 4363 unique elements.  @voj has 6049 unique elements
Shared elements of @uni and @voj are:
0 elements shared
```

There are no shared elements. With these observations, we see that all non-zero Bernoulli Numbers $\geq$ *Bernoulli*$_2$ fall into one or the other of two unique Populations:

- Population I has numerators whose absolute values are $\equiv$ 1 mod 6, and denominators of Population I only.

- Population II has numerators whose absolute values are $\equiv$ 5 mod 6, and denominators of Population II only.

If |*numerator*| $\equiv$ 1 mod 6, it is a Population I Bernoulli Number.
If |*numerator*| $\equiv$ 5 mod 6, it is Population II. The first 10k Population I Bernoulli Numbers are about 1.522 times more abundant than the first 10k of Population II.
To obtain the first 10k Population I numbers, you need to go to *Bernoulli*$_{33136}$.
To obtain the first 10k Population II numbers, you need to go to *Bernoulli*$_{50448}$.
However, in the first 10k instances of each Population,
Population I has 4363 *unique* denominators.
Population II has 6049 *unique* denominators.

To download the first 10,000 Population I denominators: pop1-deno.txt.
To download the first 10,000 Population II denominators: pop2-deno.txt.
To download 4,363 unique Population I denominators: pop1-uni.txt.
To download 6,049 unique Population II denominators: pop2-uni.txt.
To download 10,000 *n* producing Population I numbers: pop1-n.txt.
To download 10,000 *n* producing Population II numbers: pop2-n.txt.
OEIS has now published 2 sequences of *n* that produce these Populations: A233578 and A233579.

The table below lists the first 26 denominators by *n* and population. It shows that for each even *n*, the denominator is either Pop I or Pop II:

n and Denominator

n	Pop I	Pop II
2	6	
4	30	
6	42	
8	30	
10		66
12	2730	
14	6	
16		510
18	798	
20		330
22		138
24	2730	
26	6	
28		870
30		14322
32		510
34	6	
36	1919190	
38	6	
40	13530	
42	1806	
44		690
46		282
48		46410
50		66
52		1590

Because Bernoulli numerators become very large very rapidly, I want a method of determining the Population of a Bernoulli Number, either from n or from the denominator. I have not figured out a method using n. But I did figure out a method using the denominator. von Staudt and Clausen figured out how to obtain denominators from n. So, using their theorem, I actually can determine the Population of a Bernoulli Number, just from n (as we shall see). First, we need a theorem and an intermediate conjecture:

Theorem 1. *All Bernoulli denominators* ≥ 6 *are divisible by 6.*

Proof. This is already proven. See N.J.A. Sloane's entries under FORMULA and REFERENCES in A090801. Sloane wrote, "In particular, all numbers [denominators] after the first two (which are the denominators of B_0 and B_1) are divisible by 6." Since the first denominator of 6 (the denominator of B_2) is after the first two, all Bernoulli denominators ≥ 6 are divisible by 6. □

Conjecture 3:

All Bernoulli denominators ≥ 6, divided by 6, are $\equiv \{1, 5, 7, 11, 13, 17, 19, 23, 25$ *or* $29\}$ mod 30.

Program 3

I wrote this *Maxima* program to check Conjecture 3. It uses the von Staudt-Clausen Theorem to generate Bernoulli denominators, divides each by 6, then checks each quotient mod 30:

```
load(basic)$ i:[6]$ n:0$ for t:1 thru 500 step 0 do (n:n+2,
for p:3 while p-1<=n step 0 do (p:next_prime(p), if mod(n, p-1)=0 then
push(p, i)), a:(product(i[k], k, 1, length(i))), q:a/6, r:mod(q,30),
if (r=1 or r=5 or r=7 or r=11 or r=13 or r=17 or r=19
or r=23 or r=25 or r=29) then (t:t+1, i:[6]) else
(print("exception at ", q, " mod 30 =", r), t:4000))$
print("# Checked to Bernoulli_", n)$
```

The above program checked to $Bernoulli_{1000}$, and found no exceptions to Conjecture 3.

Conjecture 4:

If the Bernoulli denominator, divided by 6, is $\equiv \{1, 5, 7, 13,$ *or* $19\}$ mod 30, then the Bernoulli Number is Population I, and the abs. value of the numerator is $\equiv$ 1 mod 6.

Conjecture 5:

If the Bernoulli denominator, divided by 6, is $\equiv \{11, 17, 23, 25$ *or* $29\}$ mod 30, then the Bernoulli Number is Population II, and the abs. value of the numerator is $\equiv$ 5 mod 6.

Programs 4 and 5

Conjectures 4 and 5 actually give you the residue mod 6 of the abs. value of the Bernoulli numerator from its denominator. Never has been done before! These are the *Maxima* programs I wrote for Conjectures 4 and 5:

```
float(true)$ n:0$ for p:0 thru 500 step 0 do
(n:n+2, b:bern(n), m:mod(abs(num(b)),6), if m=1 then
(d:denom(b), q:d/6, x:mod(q,30),
if (x=1 or x=5 or x=7 or x=13 or x=19) then p:p+1 else
print("exception at ", q, " mod 30 =",x)))$ print("Checked to B_",n)$

float(true)$ n:0$ for p:0 thru 500 step 0 do
(n:n+2, b:bern(n), m:mod(abs(num(b)),6), if m=5 then
```

```
(d:denom(b), q:d/6, x:mod(q,30),
if (x=11 or x=17 or x=23 or x=25 or x=29) then p:p+1 else
print("exception at ", q, " mod 30 =",x)))$ print("Checked to B_",n)$
```

There are no exceptions when the above programs are run. That means Conjecture 4 is true up to at least $Bernoulli_{1730}$, and Conjecture 5 is true up to at least $Bernoulli_{2458}$. The above two programs generate absolute values of Bernoulli numerators, and obtain their residues mod 6. However, if Conjectures 4 and 5 are true, then you don't need to generate numerators. You can use von Staudt-Clausen to generate just the denominators, and obtain the residues mod 6 of the absolute values of the numerators from the denominators! The *Maxima* program below takes as input nothing but n. It uses von Staudt-Clausen, and returns the denominator of $Bernoulli_n$, the residue mod 6 of the absolute value of the numerator of $Bernoulli_n$, and the sign of the numerator of $Bernoulli_n$:

Program 6

```
float(true)$ load(basic)$ i:[6]$ n:64$ if oddp(n) then
(s:n+1, print("ERROR: ",n, " is odd. Using ",s, " instead."), n:s)$
for p:3 while p-1<=n do
(p:next_prime(p), if mod(n, p-1)=0 then push(p,i))$
j:length(i)$ d:product(i[k], k, 1, j)$
print("denominator of Bern_", n," = ",d)$
q:d/6$ x:mod(q,30)$ if (x=1 or x=5 or x=7 or x=13 or x=19) then
print("abs val of numerator of Bern_", n, " is cong. to 1 mod 6.") else
print("abs val of numerator of Bern_", n, " is cong. to 5 mod 6.")$
v:mod(n,4)$ if v=0 then
print("numerator of Bern_", n, " is negative.") else
print("numerator of Bern_", n, " is positive.")$
```

If you enter an odd n; the program increments it so it's even. The method of determining the sign of the numerator is already known; I didn't need a conjecture for it: For even $n \geq 2$, if $n \equiv 0 \bmod 4$, then the numerator of $Bernoulli_n$ is negative. Otherwise, it is positive.

Copyright