Computing Bernoulli Numbers

William Stein

(joint work with Kevin McGown of UCSD)

February 16, 2006

Bernoulli Numbers

Defined by Jacques Bernoulli in posthumous work *Ars conjectandi Bale, 1713*.

$$\frac{x}{e^x - 1} = \sum_{n=0}^{\infty} \frac{B_n}{n!} x^n$$

$$B_0 = 1, \quad B_1 = -\frac{1}{2} \quad B_2 = \frac{1}{6}, \quad B_3 = 0, \quad B_4 = -\frac{1}{30},$$

$$B_5 = 0, \quad B_6 = \frac{1}{42}, \quad B_7 = 0, \quad B_8 = -\frac{1}{30}, \quad B_9 = 0,$$

Connection with Riemann Zeta Function

For integers $n \geq 2$ we have

$$\zeta(2n) = \frac{(-1)^{n+1}(2\pi)^{2n}}{2 \cdot (2n)!} B_{2n}$$
$$\zeta(1-n) = -\frac{B_n}{n}$$

So for $n \geq 2$ even:

$$|B_n| = \frac{2n!}{(2\pi)^n} \zeta(n) = \pm \frac{n}{\zeta(1-n)}.$$

Computing Bernoulli Numbers – say B_{500}

```
sage: a = maple('bernoulli(500)')     # Wall time: 1.35
sage: a = maxima('bern(500)')         # Wall time: 0.81
sage: a = maxima('burn(500)')         # broken...
sage: a = magma('Bernoulli(500)')     # Wall time: 0.66
sage: a = gap('Bernoulli(500)')       # Wall time: 0.53
sage: a = mathematica('BernoulliB[500]')  #W time: 0.18
  calcbn (http://www.bernoulli.org)   #      Time: 0.020
sage: a = gp('bernfrac(500)')         # Wall time: 0.00 ?!
```

Computing Bernoulli Numbers – say B_{1000}

```
sage: a = maple('bernoulli(1000)')     # Wall time: 9.27
sage: a = maxima('bern(1000)')         # Wall time: 5.49
sage: a = magma('Bernoulli(1000)')     # Wall time: 2.58
sage: a = gap('Bernoulli(1000)')       # Wall time: 5.92
sage: a = mathematica('BernoulliB[1000]') #W time: 1.01
  calcbn (http://www.bernoulli.org)    #      Time: 0.06
sage: a = gp('bernfrac(1000)')         # Wall time: 0.00?!
```

NOTE: Mathematica 5.2 is much faster than Mathematica 5.1 at computing Bernoulli numbers, and the timing is almost identical to PARI (for $n > 1000$), though amusingly Mathematica 5.2 is *slow* for $n < 1000$!

World Records?

Largest one ever computed was $B_{5000000}$ by O. Pavlyk, which was done in Oct. 8, 2005, and whose numerator has 27332507 digits. Computing B_{10^7} is the next obvious challenge.

Bernoulli numbers are really big!

Sloane Sequence A103233:

n	0	1	2	3	4	5	6	7
a(n)	1	1	83	1779	27691	376772	4767554	???

Here $a(n) =$ Number of digits of numerator of B_{10^n}.

Number of Digits

Clausen and von Staudt: $d_n = \text{denom}(B_n) = \prod_{p-1|m} p.$

Number of digits of numerator is

$$\lceil \log_{10}(d_n \cdot |B_n|) \rceil$$

But

$$\log(|B_n|) = \log\left(\frac{2n!}{(2\pi)^n}\zeta(n)\right)$$
$$= \log(2) + \sum_{m=1}^{n} \log(m) - \log(2) - n\log(\pi) + \log(\zeta(n)),$$

and $\zeta(n) \sim 1$. In 10 minutes this gives *two new entries* for Sloane's sequence:

$$a(10^7) = 57675292 \qquad \text{and} \qquad a(10^8) = 676752609.$$

Stark's Observation (after talk)

Use Stirling's formula, which, ammusingly, involves small Bernoulli numbers:

$$\log(\Gamma(z)) = \frac{1}{\log(2\pi)} + \left(z - \frac{1}{2}\right)\log(z) - z + \sum_{n=1}^{\infty} \frac{B_{2n}}{2n(2n-1)z^{2n-1}}.$$

This would make computation of the number of digits of the numerator of B_n pretty easy. See http://mathworld.wolfram.com/StirlingsSeries.html

Tables?

I couldn't find *any* interesting tables at all!

But from
`http://mathworld.wolfram.com/BernoulliNumber.html`
"The only known Bernoulli numbers B_n having prime numerators occur for n=10, 12, 14, 16, 18, 36, and 42 (Sloane's A092132) [...] with no other primes for $n \leq 55274$ (E. W. Weisstein, Apr. 17, 2005)."

So maybe 55274 is the biggest enumeration of B_k's ever? Not anymore... since I just used SAGE to script a bunch of PARI's on my new 64GB 16-core computer, and made a table of B_k for $k \leq 94000$. It's very compressed but takes over 3.4GB.

Buhler et al.

Basically, compute $B_k \pmod{p}$ for all $k \leq p$ and p up to $16 \cdot 10^6$ using clever Newton iteration to find $1/(e^x - 1)$. In particular, "if g is an approximation to f^{-1} then ... $h = 2g - fg^2$" is twice as good. (They also gain a little using other tricks.)

Math 168 Student Project

Figure out why PARI is vastly faster than anything else at computing B_k and explain it to me.

Kevin McGown rose to the challenge.

```
/* assume n even > 0. Faster than standard bernfrac for n >= 6 */
GEN
bernfrac_using_zeta(long n)
{
  pari_sp av = avma;
  GEN iz, a, d, D = divisors(utoipos( n/2 ));
  long i, prec, l = lg(D);
  double t, u;

  d = utoipos(6); /* 2 * 3 */
  for (i = 2; i < l; i++) /* skip 1 */
  { /* Clausen - von Staudt */
    ulong p = 2*itou(gel(D,i)) + 1;
    if (isprime(utoipos(p))) d = muliu(d, p);
  }
  /* 1.712086 = ??? */
  t = log( gtodouble(d) ) + (n + 0.5) * log(n) - n*(1+log2PI) + 1.712086;
  u = t / (LOG2*BITS_IN_LONG); prec = (long)ceil(u);
  prec += 3;
  iz = inv_szeta_euler(n, t, prec);
  a = roundr( mulir(d, bernreal_using_zeta(n, iz, prec)) );
  return gerepilecopy(av, mkfrac(a, d));
}
```

Compute $1/\zeta(n)$ to VERY high precision

```
/* 1/zeta(n) using Euler product. Assume n > 0.
 * if (lba != 0) it is log(bit_accuracy) we _really_ require */
GEN
inv_szeta_euler(long n, double lba, long prec)
{
  GEN z, res = cgetr(prec);
  pari_sp av0 = avma;
  byteptr d =  diffptr + 2;
  double A = n / (LOG2*BITS_IN_LONG), D;
  long p, lim;

  if (!lba) lba = bit_accuracy_mul(prec, LOG2);
  D = exp((lba - log(n-1)) / (n-1));
  lim = 1 + (long)ceil(D);
  maxprime_check((ulong)lim);

  prec++;
  z = gsub(gen_1, real2n(-n, prec));
  for (p = 3; p <= lim;)
  {
    long l = prec + 1 - (long)floor(A * log(p));
    GEN h;

    if (l < 3)         l = 3;
    else if (l > prec) l = prec;
    h = divrr(z, rpowuu((ulong)p, (ulong)n, l));
    z = subrr(z, h);
    NEXT_PRIME_VIADIFF(p,d);
  }
  affrr(z, res); avma = av0; return res;
}
```

What Does PARI Do?

Use

$$|B_n| = \frac{2n!}{(2\pi)^n}\zeta(n)$$

and *tightly bound* precisions needed to compute each quantity.

```
>  (1) Do you know who came up with or implemented the idea
>      in PARI for computing Bernoulli numbers quickly by
>      approximating the zeta function and using Classen
>      and von Staudt's identification of the denominator
>      of the Bernoulli number?

Henri did, and wrote the initial implementation.
I wrote the current one (same idea, faster details).

The idea independently came up (Bill Daly) on pari-dev
as a speed up to Euler-Mac Laurin formulae for zeta or
gamma/loggamma (that specific one has not been tested/
implemented so far).
```

http://www.bernoulli.org/

Bernd C. Kellner's program at `http://www.bernoulli.org/` (2002-2004) also appears to uses

$$|B_n| = \frac{2n!}{(2\pi)^n}\,\zeta(n)$$

but Kellner's program is closed source and noticeably slower than PARI (2.2.10.alpha). He claims his program "calculates Bernoulli numbers up to index $n = 10^6$ extremely quickly."

Also: **Maxima's** documentation claims to have a function `burn` that uses zeta, but it doesn't work (for me).

Kevin McGown Project

The Algorithm: Suppose $n \geq 2$ is even.

1. $K := \frac{2n!}{(2\pi)^n}$
2. $d := \prod_{p-1|n} p$
3. $N := \left\lceil (Kd)^{1/(n-1)} \right\rceil$
4. $z := \prod_{p \leq N} (1 - p^{-n})^{-1}$
5. $a := (-1)^{n/2+1} \lceil dKz \rceil$
6. $B_n = \frac{a}{d}$

What About Generalized Bernoulli Numbers?

```
>      (2) Has a generalization to generalized
>          Bernoulli numbers attached to an integer
>          and Dirichlet character been written
>          down or implemented?

Not to my knowledge.

Cheers,
    Karim.
```

Generalized Bernoulli Numbers

Defined in 1958 by H. W. Leopoldt.

$$\sum_{r=1}^{f-1} \chi(r)\, \frac{te^{rt}}{e^{ft}-1} = \sum_{n=0}^{\infty} B_{n,\chi}\, \frac{t^n}{n!}$$

Here $\chi : (\mathbb{Z}/m\mathbb{Z}) \to \mathbb{C}$ is a Dirichlet character.

These give **values at negative integers** of associated Dirichlet L-functions:

$$L(1-n,\chi) = -\frac{B_{n,\chi}}{n}$$

Kubota-Leopoldt p-adic L-function (p-adic interpolation)...

$B_{n,\psi}$ Very Important to Computing Modular Forms

$$E_{k,\chi,\psi}(q) = c_0 + \sum_{m\geq 1} \left(\sum_{n|m} \psi(n) \cdot \chi(m/n) \cdot n^{k-1} \right) q^m \in \mathbb{Q}(\chi, \psi)[[q]],$$

where

$$c_0 = \begin{cases} 0 & \text{if } L = \text{cond}(\chi) > 1, \\ -\frac{B_{k,\psi}}{2k} & \text{if } L = 1. \end{cases}$$

Theorem

The (images of) the Eisenstein series above generate the Eisenstein subspace $E_k(N, \varepsilon)$, where $N = L \cdot \text{cond}(\psi)$ and $\varepsilon = \chi/\psi$.

The Torsion Subgroup of $J_1(p)$

Let $J_1(p)$ be the Jacobian of the modular curve $X_1(p)$.

Conjecture (Stein)

$$\#J_1(p)(\mathbb{Q})_{\mathrm{tor}} = \frac{p}{2^{p-3}} \cdot \prod_{\chi \neq 1} B_{2,\chi},$$

where the χ have modulus p. (Equivalently, the torsion subgroup is generated by the rational cuspidal subgroup—see Kubert-Lang.)

(This is a generalization of Ogg's conjecture for $J_0(p)$, which Mazur proved.)

Compute $B_{n,\chi}$? One way.

Let N=modulus of χ, assumed > 1.

1. Compute $g = x/(e^{Nx} - 1) \in \mathbb{Q}[[x]]$ to precision $O(x^{n+1})$ by computing $e^{Nx} - 1 = \sum_{m\geq 1} N^m x^m/m!$ to precision $O(x^{n+2})$, and computing the inverse $1/(e^{Nx} - 1)$, e.g., using Newton iteration as in Buhler et al.
2. For each $a = 1, \ldots, N - 1$, compute $f_a = g \cdot e^{ax} \in \mathbb{Q}[[x]]$, to precision $O(x^{k+1})$. This requires computing $e^{ax} = \sum_{m\geq 0} a^m x^m/m!$ to precision $O(x^{k+1})$.
3. Then for $j \leq n$, we have $B_{j,\varepsilon} = j! \cdot \displaystyle\sum_{a=1}^{N-1} \varepsilon(a) \cdot c_j(f_a)$, where $c_j(f_a)$ is the coefficient of x^j in f_a.

This requires arithmetic **only in** $\mathbb{Q}$, except in the last easy step.

Analytic Method

Is there an analytic method to compute $B_{n,\chi}$ that is impressively fast in practice like the one Cohen/Kellner/etc. invented for B_n?

YES.

Analytic Method

Assume χ primitive now.
If

$$K_{n,\chi} := (-1)^{n-1}\, 2n! \left(\frac{N}{2i}\right)^n$$

then

$$B_{n,\chi} = \frac{K_{n,\chi}}{\pi^n\, \tau(\chi)}\, L(n, \overline{\chi})$$

There is a simple formula for a d such that $d \cdot B_{n,\chi}$ is an algebraic integer (analogue of Clausen and von Staudt).
For n large we can compute $L(n, \overline{\chi})$ *very quickly* to high precision; hence we can compute $B_{n,\chi}$ (at least if $\mathbb{Q}(\chi)$ isn't too big, e.g., $\mathbb{Q}(\chi) = \mathbb{Q}$ wouldn't be a problem). (Note, for small n that $L(n, \overline{\chi})$ converges slowly; but then just use the power series algorithm.)
Compute the conjugates of $d \cdot B_{n,\chi}$ approximately; compute minimal polynomial over $\mathbb{Z}$; factor that over $\mathbb{Q}(\chi)$, then recognize the right root from the numerical approximation to $d \cdot B_{n,\chi}$.