[6.27] Find Bernoulli numbers and polynomials

[Note: since this tip was written, Bhuvanesh Bhatt has also written a function to find both Bernoulli numbers and polynomials. Bhuvanesh' function is smaller and handles complex arguments. You can get it at his site (see the "More Resources - Web sites" section for the URL), and at ticalc.org. Also, M. Daveluy has written a Bernoulli number function, which can be found at ti-cas.org.]

Bernoulli numbers are generated from the Benoulli polynomials evaluated at zero. Bernoulli polynomials are defined by the generating function

$$\frac{te^{xt}}{e^t-1} = \Sigma_{n=0}^{\infty} B_n(x)\frac{t^n}{n!}$$

Bernoulli polynomials can also be defined recursively by

$$B_0(x) = 1 \qquad [1]$$

$$\frac{d}{dx}B_n(x) = nB_{n-1}(x) \qquad [2]$$

$$\int_0^1 B_n(x)dx = 0 \quad \text{for } n \geq 1 \qquad [3]$$

The first few Bernoulli polynomials are

$$B_0(x) = 1 \qquad B_3(x) = \frac{2x^3-3x^2+x}{2}$$

$$B_1(x) = \frac{2x-1}{2} \qquad B_4(x) = \frac{30x^4-60x^3+30x^2-1}{30}$$

$$B_2(x) = \frac{6x2-6x+1}{6} \qquad B_5(x) = \frac{6x^5-15x^4+10x^3-x}{6}$$

The nth Bernoulli number is denoted as B_n. The Bernoulli numbers can be defined by the generating function

$$\frac{t}{e^t-1} = \Sigma_{n=0}^{\infty} B_n\frac{t^n}{n!}$$

or, as mentioned above, by evaluating $B_n(0)$. However, a faster method to find Bernoulli numbers on the 89/92+ uses this identity:

$$\xi(2n) = \frac{(-1)^{n-1}B_{2n}(2\pi)^{2n}}{2(2n)!}$$

The notation *2n* is used since this identity is only true for even integers. $\xi(n)$ is the Riemann Zeta function,

$$\xi(n) = \Sigma_{k=1}^{\infty}\frac{1}{k^n}$$

Bernoulli numbers for odd *n* > 1 are zero. The first few non-zero Bernoulli numbers are

$B_0 = 1$ $\quad B_1 = -1/2$ $\quad B_2 = 1/6$ $\quad B_4 = -1/30$ $\quad B_6 = 1/42$

It turns out that the 89/92+ can evaluate $\xi(n)$ very quickly for even *n*, which is what we need. Solving the identity above for B_{2n} gives

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

This function returns the Bernoulli number B_n:

```
bn(n)
Func
©Bernoulli number Bn
©21jun00/dburkett@infinet.com

if n=0:return 1
if n=1:return ⁻1/2
if n<0:return undef
if fpart(n/2)≠0:return 0

(Σ(z^⁻n,z,1,∞)*2*n!)/(((⁻1)^(n/2-1))*(2π)^n)

EndFunc
```

The first three executable lines handle the special cases for B_0 = 1, B_1 = -1/2 and B_n undefined when n<0. The fourth line returns zero for odd *n* where n>1. Note that the expression to find B_n has been transformed from an expression in *2n* to an expression in *n*.

Finding the Bernoulli polynomials is a little more complicated, but can still be done on the 89/92+. The program uses the recursive definition given above in equations [1], [2] and [3]. First, take the antiderivative of equation [2] to find

$$B_n(x) = \int nB_{n-1}(x)dx$$

Since these integrals are simple polynomials, the 89/92+ can easily find the symbolic integral. I use the definite integral of equation [3] to find the constant of integration:

$$ff1 = \int nB_n(x)dx$$

$$ff2 = ff1 - \int_0^1 ff1(x)dx$$

To find the *n*th Bernoulli polynomial requires finding all the (n-1) previous polynomials, so this is time-consuming for higher-order polynomials. For this reason I wrote two different versions of programs to find the Bernoulli polynomials. One version is a function which returns a single polynomial of the order specified by the function argument. The second version builds a table of the polynomials up to a specified order. Individual polynomials can be quickly retrieved from this table by another function.

bnpolys(n) returns a single Bernoulli polynomial of order *n* as a function of *x:*

```
bnpolys(n)
Func
©(n) return Bernoulli polynomial of order n
©27jun00/dburkett@infinet.com

local k,f,g

x-1/2→f
```

```
if n=1:return f

for k,2,n
 ∫(k*f,x)→g
 g-∫(g,x,Ø,1)→f
endfor

return f

EndFunc
```

bnpolys() may not return the polynomial in the form you want. For example, *bnpolys(3)* returns

$$\frac{x(2x^2-3x+1)}{2}$$

Use *expand()* to convert this result to $x^3-\frac{3x^2}{2}+\frac{x}{2}$

or use *comdenom()* to get $\frac{2x^3-3x^2+x}{2}$

bnpolys() slows down for large arguments. For example, *bnpolys(20)* takes about 22 seconds on my HW2 92+ with AMS 2.04, and *bnpolys(50)* takes about 160 seconds. This long execution time occurs because *bnpolys()* must calculate all the polynomials of order n-1, to return the polynomial of order *n.* If you frequently use higher-order polynomials, it is worthwhile to build a table in advance, then use a simple routine that just recalls the appropriate polynomial.

These routines perform those functions. *bnpoly()* builds the list of polynomials, and *bpo()* is a function that returns a given polynomial, after *bnpoly()* is run.

First, use *bnpoly()* to build the list of polynomials:

```
bnpoly(nxx)
prgm
©(n) Fill Bernoulli polynomial list bpoly[] up to n.
©1julØØ/dburkett@infinet.com
©Save polynomials in list bpoly[].

local k,k1,k2,bpdim,ff1,usermode,choice

©Save user's mode; set modes
getmode("all")→usermode
setmode({"Complex Format","Real","Vector
Format","Rectangular","Exact/Approx","Exact","Pretty Print","Off"})

©If bpoly[] exists unarchive it, else create it & initialize it.
if gettype(spfn\bpoly)="NONE" then
 newlist(1)→spfn\bpoly
 x-1/2→spfn\bpoly[1]
else
 unarchiv(spfn\bpoly)
endif

dim(spfn\bpoly)→bpdim
clrio

©Loop to derive Bernoulli polynomials
```

```
if nxx>bpdim then
 augment(spfn\bpoly,newlist(nxx-bpdim))→spfn\bpoly
 for k,bpdim+1,nxx
  ∫(k*spfn\bpoly[k-1],x)→ff1
  ff1-∫(ff1,x,Ø,1)→spfn\bpoly[k]
  disp k
  disp spfn\bpoly[k]
 endfor
endif

©Prompt to archive bpoly[]
1→choice
dialog
 title "BNPOLY"
 dropdown "Archive polynomial list?",{"yes","no"},choice
enddlog

if ok=1 and choice=1 then
 archive spfn\bpoly
endif

©Restore user's modes
setmode(usermode)
clrio
disphome

Endprgm
```

Then, use *bpo()* to recall a particular polynomial:

```
bpo(kk)
func
©(k) Bernoulli polynomial Bk
©1julØØ/dburkett@infinet.com
©Generates "Domain error" if kk>dim(bpoly)

if kk<Ø:return "bpo arg error"

when(kk=Ø,1,spfn\bpoly[kk])

Endfunc
```

These routines must both be installed in a folder called *\spfn*. The list of Bernoulli polynomials will be created in this same folder. To create a list of polynomials, execute

```
bnpoly(n)
```

where *n* is the highest-order polynomial you expect to use. For example, if you use polynomials up to the 20th order, then use *bnpoly(20).* If you later need higher-order polynomials, just run *bnpoly()* again, and it will append the additional polynomials to the list.

As *bnpoly()* creates the polynomials, it displays them in the program I/O screen. After all the polynomials are created, a dialog box is shown with the prompt

```
Archive polynomial list?
```

Answer 'yes' to archive the list in flash memory, or 'no' to leave it in RAM. I provide this option because the list is large for high order polynomials, and there is no advantage to having it in RAM. If you later increase the size of the list, *bnpoly()* will unarchive it for you.

This table gives you some idea of the memory required for tables of various orders of polynomial. The size is in bytes with the list archived. The size of the last polynomial in the list is also shown

Size of bpoly[] and last polynomial

Polynomial order	**Total list size (bytes)**	**Last polynomial size (bytes**
10	400	71
20	1,463	155
30	3,370	264
50	10,525	527
75	28,581	967
100	61,327	1,609

Since the maximum size of an 89/92+ variable is about 64K, the maximum order that can be saved is about 100.

Operation of *bpo()* is simple: just call *bpo(k),* where *k* is the the order of the polynomial to return. For example, if *bnpoly()* was executed to create a list of polynomials up to order 30, then to return the 10-th order polynomial, use

```
    bpo(3Ø)
```

bpo() does two things, beyond simply returning the polynomial in *bpoly[k].* First, it returns an error message if k<0. Second, it returns 1 for k=0, since $B_0(x)$ is 1, and $B_0(x)$ is not stored in *bpoly[].*

I would usually include a test to ensure that *k* is less than the list size, like this:

```
    if kk<Ø or kk>dim(spfn\bpoly): return "bpo arg error"
```

However, it turns out that the 89/92+ are *extremely* slow to find the dimension of lists with large elements, even if the lists have few elements. For example, if the bpoly[] has 70 polynomials, its size is about 24000 bytes, and `dim(spfn\bpoly)` takes over 30 seconds to return the dimension of 70! If bpoly[] has 100 elements, `dim(bpoly)` takes 5 seconds to fail with a "Memory" error message.

This explains the long delay when *bnpoly()* is used to add polynomials to lists that are already of high order.

One potential work-around to this problem would be to use the `try...else...endtry` condtional test to trap the error, but *functions* cannot use `try...endtry`!

So, rather than do a proper test for a valid input argument, I accept that fact that *bpo()* will not fail gracefully, in return for fast execution times for proper arguments.

For more information on Bernoulli numbers and polynomials, these references may be helpful:

Handbook of Mathematical Functions, Milton Abramowitz and Irene A. Stegun, Dover, 1965. This reference defines the Bernoulli numbers and polynomials, has a very complete table of properties, and also tables of the polynomials and numbers. Various applications are shown throughout the book where relevant.

Numerical Methods for Scientists and Engineers, R.W. Hamming, Dover, 1962. Hamming shows another method to generate the Bernoulli numbers using a summation (p190).

Ada and the first computer, Eugene Eric Kim and Betty Alexandra Toole, Scientific American magazine, May 1999. This very interesting article describes a program written in 1843, by Ada, countess of Lovelace, for Charles Babbage's Analytical Engine. The purpose of the program was to calculate the Bernoulli numbers. The analytical engine, a mechanical computer, was never completed, as the British government didn't fund it adequately, and Babbage kept revising the design.

These web sites are also interesting:

http://www.treasure-troves.com/math/BernoulliNumber.html
This site describes the derivation and basic properties of the Bernoulli numbers and polynomials.

http://venus.mathsoft.com/asolve/constant/apery/brnlli.html
This site shows the expression for the tangent function, as a function of the Bernoulli number.

http://www-history.mcs.st-andrews.ac.uk/~history/Mathematicians/Bernoulli_Jacob.html
This site has a very nice biography of Jacob Bernoulli, who was the particular Bernoulli responsible for the Bernoulli numbers and polynomials.