π: A 2000-Year Search Changes Direction

Victor Adamchik
Wolfram Research, Inc.
100 Trade Center Dr., Champaign, IL 61820
victor@wolfram.com
&
Stan Wagon
Macalester College
St. Paul, MN 55105
wagon@macalstr.edu

Introduction

One of the charms of mathematics is that it is possible to make elementary discoveries about objects that have been studied for millenia. A most striking example occurred recently when David Bailey of NASA/Ames and Peter Borwein and Simon Plouffe of the Centre for Experimental and Computational Mathematics at Simon Fraser University (henceforth, BBP) discovered a remarkable, and remarkably simple new formula for π. Here is their formula:

$$\pi = \sum_{k=0}^{\infty} \frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right)$$

The formula is not too hard to prove, but that misses the point: Finding it in the first place is what took some effort. BBP report that the formulaàs discovery was "a combination of inspired guessing and extensive searching using the PSLQ integer relation algorithm". The PSLQ algorithm is a method for recognizing whether a constant is a combination of other, more fundamental, constants (it has been implemented in *Mathematica* by Richard Crandall [Cra]). The discoverers sought such a formula because they were aware that it could be used to compute the nth digit of π (in base 16), without computing any prior digits. This goes completely against conventional wisdom, and totally eliminates high-precision requirements from a computation of, say, the billionth hexadecimal digit! The big question now is whether such a method exists for the base-10 digits of π. For a fuller treatment, including several results about other constants and open questions, see [BBP], which can be fetched from Peter Borwein's web site (http://www.cecm.sfu.ca/~pborwein).

In this paper we will discuss the BBP formula and show how *Mathematica* can be used to generate many other formulas of the same sort. Moreover, the *Mathematica*-based methods are symbolic, and so a proof of the formula is a natural consequence of the discovery.

1. A Proof of the New Formula for π

To prove the BBP formula we analyze separately the general form of the four summands. First observe that :

$$\sum_{k=0}^{\infty} \frac{1}{16^k\,(8\,k+i)} = 2^{i/2} \sum_{k=0}^{\infty} \int_0^{\frac{1}{\sqrt{2}}} z^{i+8\,k-1}\, dz = 2^{i/2} \int_0^{\frac{1}{\sqrt{2}}} \left(\sum_{k=0}^{\infty} z^{i+8\,k-1} \right) dz = 2^{i/2} \int_0^{\frac{1}{\sqrt{2}}} \frac{z^{i-1}}{1-z^8}\, dz$$

We can now evaluate the full formula, first introducing the new function:

$$g[\text{i\_}] := 2^{\frac{i}{2}} \int_0^{\frac{1}{\sqrt{2}}} \left(\sum_{k=0}^{\infty} z^{8\,k+i-1} \right) dz$$

Then the BBP formula could be proved in one line:

```
Simplify[4 g[1] - 2 g[4] - g[5] - g[6]]
```

π

This sort of series for π did not arise out of the blue. Of course, there is the famous Leibniz series,

$$\pi = 4 \sum_{k=0}^{\infty} \frac{(-1)^k}{2\,k+1}$$

but that uses powers of -1, not a useful base. There are, however, other well-known series having the same general form, that is, sums of series, each one of which has as general term a reciprocal of an integer power times a rational number over a linear function. Here are four such that are easy to derive. We call these *one-term* series since each involves only one linear term; the BBP π-series is a 4-term series.

$$\log(2) = \sum_{k=0}^{\infty} \frac{1}{2^k}\,\frac{1}{k} \qquad \log(2) = \frac{2}{3} \sum_{k=0}^{\infty} \frac{1}{9^k}\,\frac{1}{2\,k+1}$$

$$\log(3) = \sum_{k=0}^{\infty} \frac{1}{4^k}\,\frac{1}{2\,k+1} \qquad \log\left(\frac{9}{10}\right) = -\sum_{k=0}^{\infty} \frac{1}{10^k}\,\frac{1}{k}$$

The first is simply the Maclaurin series for $\log(1+x)$, at x $=\frac{1}{2}$. The fourth, which we mention to show that base -10 formulas do exist, is also a Maclaurin expansion; this one was used by BBP to compute the 5, 000, 000, 065 th decimal digit of $\log(\frac{9}{10})$ a world record for decimal digits. The other two come from the Gregory series - the Maclaurin series for $\frac{1+x}{1-x}$ - with x equal to $\frac{1}{3}$ for the 9^k series and $\frac{1}{2}$ for the 4^k case. We illustrate the log(3) case; the infinite series represented by the following output must be log(3) because $\frac{1+x}{1-x}$ is 3 when $x = \frac{1}{2}$, and it is clearly the same series as the one for log(3) above.

```
Normal[Series[Log[(1 + x)/(1 - x)], {x, 0, 12}]] /. x → 1/2
```

$$\frac{2\left(\frac{1}{2}\right)^{11}}{11}+\frac{2\left(\frac{1}{2}\right)^{9}}{9}+\frac{2\left(\frac{1}{2}\right)^{7}}{7}+\frac{2\left(\frac{1}{2}\right)^{5}}{5}+\frac{2\left(\frac{1}{2}\right)^{3}}{3}+2\,\frac{1}{2}$$

These series could be used to extract, for example, base-2 digits of log(2) or log(3), but these real numbers do not have the cachet that π does, and apparently no one, prior to BBP, thought of using such a series for digit extraction.

2. Far-Out Hex Digits

It is a fairly routine matter to use the BBP series to extract some far-out hex digits. Consider, for example, the real

$$s = \sum_{k=0}^{\infty} \frac{1}{16^k} \frac{4}{8k+1}$$

The far-out digits (say five digits starting from the dth) arise as the fractional part of
16^d . s. Multiplying 16^d into the sum and then splitting it into a sum from 0 to d and a sum from d+1 to infinity does the job. The second sum

$$s = \sum_{k=d+1}^{\infty} \mathrm{Frac}\left[\frac{1}{16^{k-d}} \frac{4}{8k+1}\right]$$

and it is easy to see that 15 terms of this is more than enough for the accuracy required. The first sum is

$$s = \sum_{k=0}^{d} \mathrm{Frac}\left[\frac{1}{16^{k-d}} \frac{4}{8k+1}\right]$$

and this is computed by reducing the numerator modulo 8k+1. Because we want the fractional part, only the residue counts, and that can be divided by 8k+1 using floating-point arithmetic. Since modular exponentation is speedy via **`PowerMod`**, high-precision is not required. Of course, in the case of π, we have four such series to consider.

The **`DigitExtractor`** code that follows produces five digits for real numbers having representations of the type we are considering: it takes as input the base b (16 in the BBP case) and the coefficient list ({4, 0,0, -2, -1, -1, 0, 0} in the BBP case) and returns the first five base-b digits, starting from the dth. The leading 3 of π is considered to be the 0th digit. The 15 is enough to guarantee that the omitted terms are negligible, at least for the numbers that arise in this article. A **`Do`**-loop is used for **`mainSum`**, as discussed in the appendix. The use of **`N[coeffs]`** guarantees that the computation is done with machine precision reals; this does limit us, however, because adding the fractional parts of, say, 10,000,000 reals each with 16 digits of precision can lead to unacceptable roundoff. The code works fine for d up to one million, buit not ten million. Serious digit hunters could use, say, 32 digits of precision, but this will really slow things down. One can use negative values of b (example in section 4). The last line of the code is to add back any leading zeros that **`RealDigits`** has stripped off. The code is quite short and not optimized for speed. The electronic version of this paper contains a twice faster version.

```
DigitExtractor[d_, b_, coeffs_List] := Module[{
  n = d – 1, mainSum = 0, s = Sign[b], base = Abs[b],
  nc = N[coeffs],
   rd, m = Length[coeffs]},
 Do[mainSum = Mod[mainSum + s^k nc . Table[If[coeffs[[i]] == 0, 0,
   N[PowerMod[base, n – k, (k m + i)]] / (k m + i)], {i, m}], 1],
  {k, 0, n}];
 rd = RealDigits[Mod[mainSum +
    Sum[s^k * nc . (base ^ (n – k) / (k m + Range[m])),
      {k, n + 1, n + 15}], 1], base];
 Take[Join[Array[0 &, –rd[[2]]], rd[[1]]], 5]]
```

We check by examining hexadecimal π.

```
DigitExtractor[0, 16, {4, 0, 0, -2, -1, -1, 0, 0}]
```

```
{3, 2, 4, 3, 15}
```

```
BaseForm[N[Pi], 16]
```

$3.243\mathrm{f}_{16}$

Good. Here are some farther-out digits, and a different approach to verification.

```
DigitExtractor[1000, 16, {4, 0, 0, -2, -1, -1, 0, 0}]
```

```
{3, 4, 9, 15, 1}
```

```
% == RealDigits[N[Pi, 1300], 16][[1, Range[1001, 1005]]]
```

```
True
```

And finally, we go for the millionth hex digit; this takes about six hours on a PowerMac 8100.

```
DigitExtractor[10^6, 16, {4, 0, 0, -2, -1, -1, 0, 0}]
```

```
{2, 6, 12, 6, 5}
```

These digits agree with the ones published in [BBP]. Take a moment and think about what has been done here: using *no* high-precision arithmetic, but only routine operations on 16-digit floating-point numbers, and getting absolutely no information about the early digits of π, we have found the millionth digit of π. In fact, Rabinowitz and Wagon [RW] had published a π-digit algorithm that uses only low-precision integer arithmetic, but one had to compute early digits to get late ones and the memory requirements went up as farther-out digits were sought. Even that was considered somewhat surprising. But if that was counterintuitive, the work of Bailey, Borwein, and Plouffe, is totally mind-blowing!

3. A General Method for Finding Formulas (and Proofs)

```
Off[Solve::svars, General::spell1, $MaxPrecision::mepr];
```

Mathematica`s symbolic power allows us to extend the preceding ideas to get many new formulas (together with proofs!), of which the BBP formula is a special case. Here is one result, where r denotes any real (or complex) number.

$$\pi = \sum_{k=0}^{\infty} \frac{1}{16^k} \left(\frac{4+8r}{8k+1} - \frac{8r}{8k+2} - \frac{4r}{8k+3} - \frac{2+8r}{8k+4} - \frac{1+2r}{8k+5} - \frac{1+2r}{8k+6} + \frac{r}{8k+7} \right)$$

Setting r = 0 yields the BBP formula. The proof is identical to the proof in section 1.

```
Simplify[PowerExpand[(4 + 8 r) g[1] - 8 r g[2] - 4 r g[3] - (2 + 8 r)
g[4] - (1 + 2 r) g[5] - (1 + 2r) g[6] + r g[7]]]
```

π

This unenlightening manipulation totally obscures the question of how we found this formula. Indeed, that used an interesting combination of symbolic methods. First we let a_i, i = 1, . . . , 8, denote undefined constants. The first step is to look at the abstract expression obtained by using these constants on top of the 8 k + i terms in the BBP formula (i running from 1 to 8). It is simplest to use the formulation in terms of integrals (`g(i)`) derived in section 1.

```
aVals = Array[a# &, 8];
rawExpr = aVals.Array[g, 8]
```

$$\sqrt{2}\left(\frac{1}{16}\, i\left(2+\sqrt{2}\right)\pi+\frac{1}{16}\left(-i\sqrt{2}\,\pi+2\sqrt{2}\,\tan^{-1}(2)+4\tan^{-1}\left(\frac{1}{\sqrt{2}}\right)+\sqrt{2}\,\log(2)+\sqrt{2}\,\log\left(\frac{5}{2}\right)-2\log\left(-1+\frac{1}{\sqrt{2}}\right)+2\log\left(1+\frac{1}{\sqrt{2}}\right)\right)\right)a_1+$$
$$2\left(\left(\frac{1}{8}+\frac{i}{8}\right)\pi+\frac{1}{8}\left(-i\pi-2\tan^{-1}(2)+\log(3)\right)\right)a_2+$$
$$2\sqrt{2}\left(-\frac{1}{16}\, i\left(-2+\sqrt{2}\right)\pi+\frac{1}{16}\left(i\sqrt{2}\,\pi+2\sqrt{2}\,\tan^{-1}(2)-4\tan^{-1}\left(\frac{1}{\sqrt{2}}\right)-\sqrt{2}\,\log(2)-\sqrt{2}\,\log\left(\frac{5}{2}\right)-2\log\left(-1+\frac{1}{\sqrt{2}}\right)+2\log\left(1+\frac{1}{\sqrt{2}}\right)\right)\right)a_3+4\left(\frac{i\pi}{8}+\frac{1}{8}\left(-i\pi+\log\left(\frac{5}{3}\right)\right)\right)a_4+$$
$$4\sqrt{2}\left(-\frac{1}{16}\, i\left(-2+\sqrt{2}\right)\pi+\frac{1}{16}\left(i\sqrt{2}\,\pi-2\sqrt{2}\,\tan^{-1}(2)+4\tan^{-1}\left(\frac{1}{\sqrt{2}}\right)-\sqrt{2}\,\log(2)-\sqrt{2}\,\log\left(\frac{5}{2}\right)-2\log\left(-1+\frac{1}{\sqrt{2}}\right)+2\log\left(1+\frac{1}{\sqrt{2}}\right)\right)\right)a_5+$$
$$8\left(\left(-\frac{1}{8}+\frac{i}{8}\right)\pi+\frac{1}{8}\left(-i\pi+2\tan^{-1}(2)+\log(3)\right)\right)a_6+$$
$$8\sqrt{2}\left(\frac{1}{16}\, i\left(2+\sqrt{2}\right)\pi+\frac{1}{16}\left(-i\sqrt{2}\,\pi-2\sqrt{2}\,\tan^{-1}(2)-4\tan^{-1}\left(\frac{1}{\sqrt{2}}\right)+\sqrt{2}\,\log(2)+\sqrt{2}\,\log\left(\frac{5}{2}\right)-2\log\left(-1+\frac{1}{\sqrt{2}}\right)+2\log\left(1+\frac{1}{\sqrt{2}}\right)\right)\right)a_7+2\log\left(\frac{16}{15}\right)a_8$$

This looks like a mess. But actually, it is quite beautiful! This will be clear after some simplifications. We can automate this latter simplification as it follows.

```
LogExpand[expr_] := PowerExpand[expr] /. {Log[n_Integer] :> Apply[#2 .
Log[#1]&, Transpose[FactorInteger[n]]],
Log[v_/;v<0] -> Log[-1] + Log[-v],
ArcTan[1/Sqrt[2]] -> Pi/2 - ArcTan[Sqrt[2]]}
```

We now perform some simplifications on the raw expression.

```
simpler = Together[LogExpand[rawExpr] /. {
  Log[1 - 1/Sqrt[2]] -> -Log[2]/2 - Log[1 + Sqrt[2]],
  Log[1 + 1/Sqrt[2]] -> -Log[2]/2 + Log[1 + Sqrt[2]]}]
```

$$\frac{1}{8}\left(2\sqrt{2}\,\log\left(1+\sqrt{2}\right)a_1+\log(5)\,a_1-2\sqrt{2}\,\tan^{-1}\left(\sqrt{2}\right)a_1+2\tan^{-1}(2)\,a_1+\sqrt{2}\,\pi\, a_1+2\log(3)\,a_2-4\tan^{-1}(2)\,a_2+2\pi\, a_2+4\sqrt{2}\,\log\left(1+\sqrt{2}\right)a_3-2\log(5)\,a_3+4\sqrt{2}\,\tan^{-1}\left(\sqrt{2}\right)a_3+4\tan^{-1}(2)\,a_3-2\sqrt{2}\,\pi\, a_3+4\log(5)\,a_4-4\log(3)\,a_4+8\sqrt{2}\,\log\left(1+\sqrt{2}\right)a_5-4\log(5)\,a_5-8\sqrt{2}\,\tan^{-1}\left(\sqrt{2}\right)a_5-8\tan^{-1}(2)\,a_5+4\sqrt{2}\,\pi\, a_5+8\log(3)\,a_6+16\tan^{-1}(2)\,a_6-8\pi\, a_6+16\sqrt{2}\,\log\left(1+\sqrt{2}\right)a_7+8\log(5)\,a_7+16\sqrt{2}\,\tan^{-1}\left(\sqrt{2}\right)a_7-16\tan^{-1}(2)\,a_7-8\sqrt{2}\,\pi\, a_7-16\log(5)\,a_8-16\log(3)\,a_8+64\log(2)\,a_8\right)$$

Only seven transcendental numbers appear, and also $\pi\sqrt{2}$. We can find these by defining a **`GetTranscentals`** function as follows.

```
GetTranscendentals[expr_] := Union[
  Cases[expr, Pi | _ArcTan | _Log, Infinity]]
```

The vertical bar in the **`Cases`** definition is the disjunction operator for patterns; thus the function finds all matching expressions to the specified forms at any level. Now we can collect on the transcendentals, using the third argument of **`Collect`** to map **`Factor`**, which simplifies things.

```
transcs = GetTranscendentals[simpler]
```

$$\{\pi, \tan^{-1}(2), \tan^{-1}(\sqrt{2}), \log(2), \log(3), \log(5), \log(1+\sqrt{2})\}$$

```
collected = Collect[simpler, transcs, Factor]
```

$$-\frac{\tan^{-1}(\sqrt{2})\,(a_1 - 2a_3 + 4a_5 - 8a_7)}{2\sqrt{2}} + \frac{1}{4}\tan^{-1}(2)\,(a_1 - 2a_2 + 2a_3 - 4a_5 + 8a_6 - 8a_7) + \frac{\log(1+\sqrt{2})\,(a_1 + 2a_3 + 4a_5 + 8a_7)}{2\sqrt{2}} + \frac{1}{8}\pi\left(\sqrt{2}\,a_1 + 2a_2 - 2\sqrt{2}\,a_3 + 4\sqrt{2}\,a_5 - 8a_6 - 8\sqrt{2}\,a_7\right) + \frac{1}{8}\log(5)\,(a_1 - 2a_3 + 4a_4 - 4a_5 + 8a_7 - 16a_8) + \frac{1}{4}\log(3)\,(a_2 - 2a_4 + 4a_6 - 8a_8) + 8\log(2)\,a_8$$

Note the simple form: seven transcendental numbers are multiplied by linear combinations of the a_i. If we can arrange for the multipliers to be, respectively, 0, 1, 0, 0, 0, 0, and 0, then the sum will equal π. To isolate the desired equations, we gather up all the coefficients of the transcendentals, and $\pi\sqrt{2}$ as well. **`Coefficient-List`** returns a matrix; the reader might wish to examine the steps in more detail, but flattening and deleting the 0s gets us all the a-combinations that occur as coefficients.

```
(system = DeleteCases[Flatten[
   CoefficientList[collected,
     Append[transcs, Sqrt[2]]]], 0]) // TableForm
```

$$\begin{array}{l}
\frac{a_1}{4} + \frac{a_3}{2} + a_5 + 2a_7 \\
\frac{a_1}{8} - \frac{a_3}{4} + \frac{a_4}{2} - \frac{a_5}{2} + a_7 - 2a_8 \\
\frac{a_2}{4} - \frac{a_4}{2} + a_6 - 2a_8 \\
8a_8 \\
-\frac{a_1}{4} + \frac{a_3}{2} - a_5 + 2a_7 \\
\frac{a_1}{4} - \frac{a_2}{2} + \frac{a_3}{2} - a_5 + 2a_6 - 2a_7 \\
\frac{a_2}{4} - a_6 \\
\frac{a_1}{8} - \frac{a_3}{4} + \frac{a_5}{2} - a_7
\end{array}$$

The first 6 entries are the logarithmic and arctangent coefficients, the next-to-last entry is the coefficient of π, and the last entry is the coefficient of $\pi\sqrt{2}$. If we want the entire sum to equal π, then we want all but the seventh entry to be 0, with the seventh, the coefficient of π□, being 1.

```
Solve[system == {0, 0, 0, 0, 0, 0, 1, 0}]
```

$$\left\{\left\{a_2 \to a_4 + 2,\ a_6 \to \frac{a_4}{4} - \frac{1}{2},\ a_1 \to 2 - a_4,\ a_3 \to \frac{a_4}{2} + 1,\ a_5 \to \frac{a_4}{4} - \frac{1}{2},\ a_7 \to -\frac{a_4}{8} - \frac{1}{4},\ a_8 \to 0\right\}\right\}$$

This shows that the solution space has dimension 1; we let a_4 be a free parameter, -2 - 8 r.

```
a4 = -2 - 8 r;
solution = Simplify[aVals /. First[%%]]
```

$$\{8r+4,\ -8r,\ -4r,\ -2(4r+1),\ -2r-1,\ -2r-1,\ r,\ 0\}$$

This yields the formula for π at the beginning of this section. If we set r to 0, out pop the BBP coefficients.

```
solution /. r -> 0
```

$$\{4, 0, 0, -2, -1, -1, 0, 0\}$$

Another natural choice is r = -1/2; the resulting formula for π was also known to Bailey, Borwein, and Plouffe.

```
solution /. r -> -1/2
```

$$\left\{0, 4, 2, 2, 0, 0, -\frac{1}{2}, 0\right\}$$

In fact, as pointed out to us by P. Borwein, one can easily go from the two specific formulas to the general one by examining formula$_1$ + r (formula$_2$ - formula$_1$). As a curiosity, we note that r = $\frac{1}{8}$ yields a series for π with first term $\frac{22}{7}$. Thus we can get a series expansion of $\frac{22}{7}$ - π. By arranging that 1 be the value of the coefficient of $\tan^{-1}(2)$ we get a series representation of $\tan^{-1}(2)$:

```
aVals /. First[
  Solve[system == {0, 0, 0, 0, 0, 1, 0, 0}]]
```

$$\left\{8r+3,\ -8r-2,\ -4r-\frac{1}{2},\ -8r-2,\ -2r-\frac{3}{4},\ -2r-\frac{1}{2},\ r+\frac{1}{8},\ 0\right\}$$

Letting r = - $\frac{1}{4}$ leads to several vanishing values:

```
% /. r -> -1/4
```

$$\left\{1, 0, \frac{1}{2}, 0, -\frac{1}{4}, 0, -\frac{1}{8}, 0\right\}$$

thus we get the following representation of $\tan^{-1}(2)$:

$$\tan^{-1}(2) = \sum_{k=0}^{\infty} \frac{1}{16^k}\left(\frac{1}{8k+1} + \frac{1}{2(8k+3)} - \frac{1}{4(8k+5)} - \frac{1}{8(8k+7)}\right)$$

A quick numerical check is comforting; speedy convergence means that 10 terms give agreement to machine precision.

```
Sum[1/16^k {1, 1/2, -1/4, -1/8} . (1 / (8 k + {1, 3, 5, 7})),{k, 0,
15}] == ArcTan[2.]
```

True

The formulas of this section are not particularly useful vis-a-vis π. But the $\tan^{-1}(2)$ result seems to be new. More important, perhaps further investigations using the symbolic manipulations presented here will be useful in extending the incipient theory of digit extraction.

Exercise: Use this method to derive the following two formulas.

$$\log(3) = \sum_{k=0}^{\infty} \frac{1}{16^{k+1}} \left(\frac{16}{4k+1} + \frac{4}{4k+3} \right)$$

$$\log(5) = \sum_{k=0}^{\infty} \frac{1}{16^{k+1}} \left(\frac{16}{4k+1} + \frac{16}{4k+2} + \frac{4}{4k+3} \right)$$

4. And Still More!

■

It turns out to be fruitful to consider generalizations. For example, we may consider alternating series, or bases other than 16. For the former we can vary e (the sign); for the latter we vary n (2n gives the linear coefficient and 2^n gives the base; thus n = 4 yields 8 and 16, respectively, as in section 4). Here is a general version of the function g of section 1.

$$g[\text{i\_, n\_, e\_}] := 2^{\frac{i}{2}} \int_0^{\frac{1}{\sqrt{2}}} \left(\sum_{k=0}^{\infty} e^k z^{2nk+i-1} \right) dz$$

Throughout this and the next section we will use the following functions, which perform various simplifications. It is natural to use such specialized simplification routines in a specific project, because the built-in simplifications cannot anticipate all possibilities. Of course, it takes a bit of work, in a given context, to discover exactly what sort of simplifications will be useful; but once that is done, they may as well be gathered together in functional form for ease of use.

```
LogOfRadicals[expr_] := expr /.
Module[{tmp},
  tmp = First /@ Union[
     Cases[Level[expr, -2], Log[e_] /; !RationalQ[e]]];
  tmp = Select[Flatten[Table[{tmp[[i]], tmp[[j]]},
     {i,Length[tmp] - 1}, {j, i + 1, Length[tmp]}], 1],
  RationalQ[Expand[#[[1]] #[[2]]]]& ];
    Apply[Log[#1] ->-Log[#2] + Log[Expand[Times[##]]]&, tmp, {1}]
]

FullExpand[e_, fun_:Identity] :=
    Collect[LogExpand[LogOfRadicals[e]],
    {Pi, _Log, _ArcTan}, fun]

RationalQ[_Integer | _Rational] := True
RationalQ[_] := False
```

`LogOfRadicals` simplifies combinations of logarithms of radicals

```
LogOfRadicals[Log[3 - 1/Sqrt[3]] + Log[3 + 1/Sqrt[3]]]
```

$$\log\left(\frac{26}{3}\right)$$

while **`FullExpand`** applies several of these simplifications at once.

```
FullExpand[Log[3 - 1/Sqrt[3]] + Log[3 + 1/Sqrt[3]]]
```

$$\log(2) - \log(3) + \log(13)$$

■ n = 2, Alternating: log(5), π, $\tan^{-1}(2)$

The n = 1 case does not yield anything interesting, so let us start with n = 2, with alternating signs (e = -1). There are four terms to add up in this case; FullExpand performs major simplifications. The reader might wish to examine these and the later computations in unsimplified form.

```
Clear[a, Subscript];
n = 2;
aVals = Array[a# &, 2 n];
rawExpr = FullExpand[aVals.Array[g[#1, n, -1] &, 2 n]]
```

$$\frac{\pi a_2}{2} + \tan^{-1}(2)\left(\frac{a_1}{2} - a_2 + a_3\right) - 2\log(2)\,a_4 + \log(5)\left(\frac{a_1}{4} - \frac{a_3}{2} + a_4\right)$$

Things are somewhat simpler than the base-16 case in that the multipliers yield a nonsingular system. Thus we can get four representations at once by extracting and then inverting the matrix of coefficients of the a_i.

```
transcs = GetTranscendentals[rawExpr];
Coefficient[rawExpr, #] & /@ transcs
```

$$\left\{\frac{a_2}{2}, \frac{a_1}{2} - a_2 + a_3, -2\,a_4, \frac{a_1}{4} - \frac{a_3}{2} + a_4\right\}$$

```
Inverse[Transpose[Table[Coefficient[Expand[%], a_i], {i, 4}]]]
```

$$\begin{pmatrix} 2 & 1 & 1 & 2 \\ 2 & 0 & 0 & 0 \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 0 & 0 & -\frac{1}{2} & 0 \end{pmatrix}$$

The columns give representations for π, $\tan^{-1}(2)$, log(2) and log(5), respectively. Note that the π-formula involves only 3 terms. Because log(2) has the well known series we omit its representation from the following list.

$$\pi = \sum_{k=0}^{\infty} \frac{(-1)^k}{4^k} \left(\frac{2}{4k+1} + \frac{2}{4k+2} + \frac{1}{4k+3}\right)$$

$$\tan^{-1}(2) = \sum_{k=0}^{\infty} \frac{(-1)^k}{4^k} \left(\frac{1}{4k+1} + \frac{1}{2(4k+3)}\right)$$

$$\log(5) = \sum_{k=0}^{\infty} \frac{(-1)^k}{4^k} \left(\frac{2}{4k+1} - \frac{1}{4k+3}\right)$$

The π represention can be used to extract base-4 digits, but the base-16 extractor does this anyway, so it is not a computationally important simplification. However, it should be noted that the base-4 formula for π can be split into the case of k even and k odd, in which case a base-16 formula (a six-termer that is a case of the general formula in section 3) falls right out!

In the nonalternating case with n = 2, nothing new arises. We do get a series for log(3), but it is identical to the Gregory series mentioned in section 1.

■ n = 3, Nonalternating: log(7)

```
n = 3;
aVals = Array[a# &, 2 n];
rawExpr = FullExpand[aVals.(g[#1, n, 1] &) /@ Range[2 n]]
```

$$\frac{2}{3}\sqrt{2}\log\left(4+\sqrt{2}\right)a_3+\pi\left(\frac{a_2}{3\sqrt{3}}-\frac{2a_4}{3\sqrt{3}}\right)+\log\left(3-\sqrt{2}\right)\left(-\frac{a_1}{6\sqrt{2}}-\frac{\sqrt{2}\,a_5}{3}\right)+$$
$$\log\left(\frac{3}{2}+\frac{1}{\sqrt{2}}\right)\left(\frac{a_1}{6\sqrt{2}}+\frac{\sqrt{2}\,a_5}{3}\right)+\log\left(1+\frac{1}{\sqrt{2}}\right)\left(\frac{\sqrt{2}\,a_1}{3}+\frac{4\sqrt{2}\,a_5}{3}\right)+$$
$$\tan^{-1}\left(\frac{-1+\sqrt{2}}{\sqrt{3}}\right)\left(\frac{a_1}{\sqrt{6}}+\frac{a_2}{\sqrt{3}}-\frac{2a_4}{\sqrt{3}}-\frac{2\sqrt{2}\,a_5}{\sqrt{3}}\right)+$$
$$\tan^{-1}\left(\frac{1+\sqrt{2}}{\sqrt{3}}\right)\left(\frac{a_1}{\sqrt{6}}-\frac{a_2}{\sqrt{3}}+\frac{2a_4}{\sqrt{3}}-\frac{2\sqrt{2}\,a_5}{\sqrt{3}}\right)+$$
$$\log(7)\left(\frac{a_2}{6}-\frac{\sqrt{2}\,a_3}{3}+\frac{a_4}{3}-\frac{4a_6}{3}\right)+\log(2)\left(\frac{a_1}{2\sqrt{2}}-\frac{\sqrt{2}\,a_3}{3}+\sqrt{2}\,a_5+4a_6\right)$$

```
transcs = GetTranscendentals[rawExpr];

(system = DeleteCases[Flatten[
  CoefficientList[rawExpr,
  Join[transcs, {Sqrt[3], Sqrt[2]}]]], 0]) // TableForm
```

$\frac{2a_3}{3}$

$-\frac{a_1}{12}-\frac{a_5}{3}$

$\frac{a_1}{12}+\frac{a_5}{3}$

$\frac{a_1}{3}+\frac{4a_5}{3}$

$\frac{a_2}{6}+\frac{a_4}{3}-\frac{4a_6}{3}$

$-\frac{a_3}{3}$

$4a_6$

$\frac{a_1}{4}-\frac{a_3}{3}+a_5$

$\frac{2a_4}{3}-\frac{a_2}{3}$

$\frac{a_1}{6}-\frac{2a_5}{3}$

$\frac{a_2}{3}-\frac{2a_4}{3}$

$\frac{a_1}{6}-\frac{2a_5}{3}$

$\frac{a_2}{9}-\frac{2a_4}{9}$

```
aVals /. Solve[system == {0,0,0,0,1,0,0,0,0,0,0,0,0}]
```

$\begin{pmatrix}0 & 3 & 0 & \frac{3}{2} & 0 & 0\end{pmatrix}$

So now we have a proved two-term formula for log(7) (this formula can also be deduced by the methods of [BBP]). We rearrange a little to get the form below.

$$\log(7) = \sum_{k=0}^{\infty} \frac{1}{8^{k+1}} \left(\frac{12}{3\,k+1} + \frac{6}{3\,k+2} \right)$$

A numerical check is comforting.

```
Sum[1 / 8^(k + 1) (12 / (3 k + 1) + 6 / (3 k + 2)), {k, 0, 20}] == Log[7.]
```

```
True
```

To illustrate the general nature of **`DigitExtractor`** , we generate some digits of
log(7). The reader can check that the digits obtained are correct. We must reinterpret the series as involving8^{-k} as opposed to 8^{-k-1}.

```
DigitExtractor[100, 8, {12/8, 6/8, 0}]
```

```
{3, 0, 0, 5, 7}
```

Exercise: Show that the alternating case with n = 3 leads to

$$\pi\sqrt{2} = \sum_{k=0}^{\infty} \frac{(-1)^k}{8^k} \left(\frac{4}{6\,k+1} + \frac{1}{6\,k+3} + \frac{1}{6\,k+5} \right)$$

5. A Powerful Closed Form

While we could examine more general bases by replacing 2 with b in the integrals defined by `g`, a slightly different approach to the general case is more enlightening, both in finding formulas and in trying to get a theoretical handle on the big picture. In this section we indicate this alternative method, and provide a general recipe for searching for series representations for logarithms and π. As discussed, the starting point is the following type of sum:

$$\sum_{k=0}^{\infty} \frac{1}{b^{n\,k}} \sum_{i=1}^{n} \frac{a_i}{i + \text{nk}}$$

where n is a positive integer and b ≥ 1 is an algebraic number. The constants a_i (from the computational point of view we are interested only in rational values) have to be found so that the sum yields a useful combination of logarithms and π. The key point is to represent the above sum as a sum of integrals:

$$\sum_{k=0}^{\infty} \frac{a_i}{b^n} \int_0^1 \frac{x^{i-1}}{b^n - x^n} \, dx$$

Thisi dentity is easily proved by expanding the right-side using the geometric series formula.Now, instead of relying on a symbolic integrator to evaluate the right-handside, we can do it ourselves, which will add both speed and understanding to theinvestigation. Using the well-known algorithm for integrating rational functions the details are somewhat lengthy and are omitted; (the idea is to use the complex roots of $b^n - a^n$ which come in conjugate pairs), we obtain the following complicated, but nevertheless nicely closed, expression for the right-hand side of(*).

GoldenGoose[b_, n_] :=

$$\sum_{i=1}^{n} \frac{1}{n} \left(a_i\, b^i \left(\text{Log}[b] \left(1 + 2 \sum_{k=1}^{\text{Floor}[\frac{n-1}{2}]} \text{Cos}\left[\frac{2\pi i k}{n}\right] \right) - \text{Log}[b-1] - (-1)^i\, \text{If}[\text{EvenQ}[n], \text{Log}[b+1] - \text{Log}[b], 0] - \sum_{k=1}^{\text{Floor}[\frac{n-1}{2}]} \text{Cos}\left[\frac{2\pi i k}{n}\right] \text{Log}\left[b^2 - 2\, b\, \text{Cos}\left[\frac{2 k \pi}{n}\right] + 1\right] + 2 \sum_{k=1}^{\text{Floor}[\frac{n-1}{2}]} \text{Sin}\left[\frac{2 i k \pi}{n}\right] \text{ArcTan}\left[\frac{\text{Sin}[\frac{2k\pi}{n}]}{b - \text{Cos}[\frac{2k\pi}{n}]}\right] \right) \right)$$

The reader might wish to compare specific instances of the formula to results of symbolic integration to provide evidence of correctness. This formula is pretty rich for further investigation. For example, the arctangent is the only place that could yield π. Let see what we can learn by trying to get the argument to arctan to be 1. We would need

$$\frac{\sin\left(\frac{2k\pi}{n}\right)}{b - \cos\left(\frac{2k\pi}{n}\right)} = 1$$

or

$$b = \cos\left(\frac{2k\pi}{n}\right) + \sin\left(\frac{2k\pi}{n}\right) = \sqrt{2}\, \sin\left(\frac{2\pi k}{n} + \frac{\pi}{4}\right)$$

Since we want to have our base, b, be an integer, we need to find an integer n so that

$$\left(\sqrt{2}\, \sin\left(\frac{2\pi k}{n} + \frac{\pi}{4}\right)\right)^n$$

is a positive integer. Simple experiments show that the first two possibilities are: n = 4 and n = 8. The second case (with the base $b^n = 16$) leads to the BBP formula. For n = 4 the base is one. The unit base is a singular case of (*): the integrals are divergent. We can avoid the resulting singularity by making $a_1 + a_2 + a_3 + a_4 = 0$ (one can then view the right side of (*) as

$$\int \frac{p\, x}{1 - x^n}\, dx$$

where p(x) is a polynomial and p(1) = 0; thus the troublesome 1 - x in the denominator is cancelled, since 1 - x divides p(x)). This yields the following expression for the main sum.

$$\pi\left(\frac{a_1}{8} - \frac{a_3}{8}\right) + 2\left(\frac{3\, a_1}{4} + \frac{a_2}{2} + \frac{3\, a_3}{4}\right) \log(2)$$

Setting it equal to π produces a one-parameter formula for π:

$$\pi = \sum_{k=0}^{\infty} \left(\frac{r-4}{4k+3} + \frac{r+4}{4k+1} + \frac{r}{4k+4} - \frac{3r}{4k+2} \right)$$

If we specialize to r = 0, we simply get a rewritten version of the Leibniz series of π. So really this should be viewed as a new formula for 0:

$$0 = \sum_{k=0}^{\infty} \left(\frac{1}{4k+1} - \frac{3}{4k+2} + \frac{1}{4k+3} + \frac{1}{4k+4} \right)$$

since that is how the Leibniz series becomes a one-parameter family of formulas. Despite
the fact that nothing new about π was learned here, it seems to us quite possible that this methodology might lead to nonexistence proofs. The outstanding such development would be a proof that π cannot be represented with this sort of series using powers of 10.

Of course, we can also use this closed form to explore some logarithms. Earlier we obtained several representations of natural logarithms of integers (2, 3, 5, and 7) in base 2. Here we consider other bases. Setting b to 3 and n to 6 yields a representation of log(13). The procedure is similar enough to earlier work that we show only partial output.

```
Clear[Subscript];
{b, n} = {3, 6};
rawExpr = FullExpand[GoldenGoose[b, n]];
transcs = GetTranscendentals[rawExpr];
Collect[rawExpr, transcs][[-2]]
```

$$\log(13) \left(\frac{a_1}{4} + \frac{3 a_2}{4} - \frac{9 a_3}{2} + \frac{27 a_4}{4} + \frac{81 a_5}{4} - \frac{243 a_6}{2} \right)$$

```
system = DeleteCases[Flatten[CoefficientList[rawExpr, transcs]], 0];
First[%]
```

$$\frac{a_1}{4} + \frac{3 a_2}{4} - \frac{9 a_3}{2} + \frac{27 a_4}{4} + \frac{81 a_5}{4} - \frac{243 a_6}{2}$$

```
Array[a# &, n] /. Solve[system == {1, 0, 0, 0, 0, 0}]
```

$$\left(\frac{7}{3} \quad \frac{1}{3} \quad \frac{4}{27} \quad \frac{1}{27} \quad \frac{7}{243} \quad 0 \right)$$

This gives a series for log(13), which we rearrange slightly to clear fractions.

$$\log(13) = \sum_{k=0}^{\infty} \frac{1}{3^{6k+5}} \left(\frac{567}{6k+1} + \frac{81}{6k+2} + \frac{36}{6k+3} + \frac{9}{6k+4} + \frac{7}{6k+5} \right)$$

Finally, recall that a most intriguing question is whether there are any interesting formulas in base 10. So we let b = 10.

```
rawExpr = Collect[
  GoldenGoose[10, 2], _Log, Factor]
```

$5\log(11)(a_1 - 10\,a_2) + 100\log(10)\,a_2 - 5\log(9)(a_1 + 10\,a_2)$

Setting $a_1 = 10\,a_2$ and $a_2 = -\frac{1}{100}$ yields a series that, after simplification, reduces to the Maclaurin expansion of $\log(\frac{9}{10})$ mentioned in section 1. Setting $a_1 = -\frac{1}{5}$ and $a_2 = 0$ yields a series that is a Gregory expansion of $\log(\frac{9}{11})$ (see section 1). So, nothing really new comes from setting b = 10.

6. Conclusion

We believe that our investigations show the power of *Mathematica* in serious symbolic investigations. Not only have we found some new formulas, but their discovery comes automatically with a proof. It is, of course, very satisfying to discover new formulas, and perhaps readers will be inspired to carry these methods farther, perhaps by looking at some new forms. It seems promising to investigate alternating series. Our preliminary investigations yielded very quickly a new formula for π (section 4).

Clearly these investigations can be carried farther. Of course, there are many more open problems than solved ones! Some of them are listed in [BBP]. We'll mention here only that it would be nice if some negative results could be obtained, such as the nonexistence of a two-term series for π, or a base-10 series for π. Our goal was to develop a technique that might yield an approach to such questions. Also, we think we have clearly shown exactly where the BBP and related formulas come from, and how π arises.

Moreover, these ideas might well yield new series representations for other special numbers or functions. Consider the polylogarithmic function

$$\mathrm{Li}_n(z) = \sum_{k=1}^{\infty} \frac{z^k}{k^n}$$

This form is close to the ones we have looked at and could be considered as a starting point for the next generalization. Itàs natural to consider the following:

$$\sum_{k=1}^{\infty} \frac{1}{b^{n\,k}} \frac{1}{k^m} \sum_{i=1}^{n+m} \frac{a_i}{i + \mathrm{nk}}$$

This form of series can be obtained from formula (*) in section 6, integrating the latter m times with respect to x. Since the repeated integration produces m free parameters we add them to the inner sum. Then our algebraic techniques yielded:

$$\mathrm{Li}_2\left(\frac{1}{2}\right) = \sum_{k=1}^{\infty} \frac{1}{2^k\,k^2}\left(-\frac{8}{k+1} - \frac{10}{4\,k+1} - \frac{54}{2\,k+3} + \frac{12}{2\,k+1} + \frac{125}{4\,k+5}\right)$$

$$\mathrm{Li}_3\left(\frac{1}{2}\right) =$$
$$\sum_{k=1}^{\infty} \frac{1}{2^k k^3} \left(-\frac{24}{2k+1} - \frac{625}{2(4k+5)} - \frac{2401}{6(4k+7)} + \frac{5}{4k+1} + \frac{324}{2k+3} + \frac{63}{4k+3} + \frac{32}{3(k+1)}\right)$$

From the computational perspective these are not that valuable, except to provide slightly faster convergence than the definition. But from the theoretical standpoint this gives another method of generating various classes of new series representations. Here are examples:

$$\zeta(3) = \sum_{k=1}^{\infty} \frac{1}{k^3} \left(-\frac{1}{k+1} + \frac{16}{k+2} + \frac{3}{4(2k+1)} - \frac{81}{4(2k+3)}\right)$$

$$\pi^2 = \sum_{k=1}^{\infty} \frac{1}{k^3} \left(-\frac{12}{k+1} + \frac{384}{k+2} + \frac{45}{2(2k+1)} - \frac{1215}{2(2k+3)}\right)$$

$$\pi = \sum_{k=1}^{\infty} \frac{1}{k^3} \left(-\frac{238}{k+1} + \frac{285}{2(2k+1)} - \frac{667}{32(4k+1)} - \frac{5103}{16(4k+3)} + \frac{35625}{32(4k+5)}\right)$$

We are grateful to David Bailey, Peter Borwein, and Simon Plouffe for sharing with us many of their insights and unpublished work in this area,and to Mark Sofroniou and
Michael Trott (WRI) for some helpful observations.

Appendix: Sum Subtleties

In order to compute some far-out base-16 digits using the BBP type of formula, one must pay attention to some subtleties of **`Sum`**. There are three basic ways to form a sum: (1) use **`Sum`**, (2) apply **`Plus`**, or (3) use a **`Do`**-loop. Method (2), an example of which is **`Apply[Plus, Table[i, {i, 1000}]]`**, is useful for small sets, but it is clearly costly in terms of memory since the entire table must be computed and stored. So we will restrict our discussion here to a comparison of mathods (1) and (3). Here they are, with calls to **`MemoryInUse`** to compare the memory required.

```
CheckMemory := Print[-mOld + (mOld = MemoryInUse[])]

mOld = MemoryInUse[];
Sum[i, {i, 1000}]; CheckMemory;
s = 0; Do[s += i, {i, 1000}]; CheckMemory;
```

```
624
```

```
656
```

This output shows a similar memory consumption, but in fact this is not the whole truth. A more careful comparison checks the memory as the sum is being computed, but prints out results only for the first few values of the index (the omitted values are all 0).

```
s1 = Sum[If[i < 4, CheckMemory]; i, {i, 1000}];
CheckMemory; Print[""];

s3 = 0; Do[If[i < 4, CheckMemory]; s3 += i, {i, 1000}];
CheckMemory
```

```
4768

0

0

−3568



832

0

0

416
```

Big surprise! Method (1) has a fairly large internal consumption; the negative value indicates that memory is being freed up after the sum is computed. And the **Do**-loop uses up hardly any memory at all. If the number of terms is increased, the first memory profile increases accordingly, but the second stays where it is, at virtually no consumption. This means that Sum cannot be used to add up, say, a million numbers. A **Do**-loop is essential in such a case.

In fact, **Sum** was designed mostly for symbolic calulations. And **Sum** is very powerful working with symbols rather then performing numerical calculations, as shown by the following example.

```
sum1 = Sum[f[k], {k, 500, 1, -1}]; // Timing
```

```
{0.039993 Second, Null}
```

```
sum2 = 0;
Do[sum2 += f[k], {k, 500, 1, -1}]; // Timing
```

```
{7.10968 Second, Null}
```

The **Do**-loop is much slower in this case because on each iteration, function **Plus** sorts its arguments, while with **Sum** arguments are sorted only one time (since **Plus** is applied only at the end).

```
Clear[s1, s3, sum1, sum2, mOld];
```

Addendum for Electronic Version of Paper

Here is a faster version of DigitExtractor, contributed by WRI's Mark Sofroniou. It is about twice as fast as the version given in the paper, but a little (though only a little) more complicated.

```
DigitExtractor::usage = "DigitExtractor[d, b, coeffs] computes five
base-Abs[b] digits starting from the dth of the real number given by
Sum[1/b^k coeffs . (1/(m k + Range[m])), {k, 1, Infinity}], where m
= Length[coeffs]. The variable term controls the truncation error;
15 should be adequate. All computations are done in machine
precision so d should not be greater than about ten million, or else
roundoff error will be excessive for the goal of 5 digits.";
```

```
DigitExtractor[d_, b_, coeffs_List] :=
  Module[{m, nb, ncoeffs, offsets, posns, pow, reald, sum, terms},
   ter m s = 15; base = Abs[b]; sign = Sign[b]; pow = d - 1; sum = 0;
   n b = N[base]; m = Length[coeffs]; ncoef f s = N[DeleteCases[coeffs, 0]];
   pos n s = Flatten[Position[coeffs, _?(# != 0 &)]]; offsets = posns;
        Do[sum = Mod[sum + sign^(pow - k) *
            (ncoeffs / offsets).(PowerMod[base, k, offsets]), 1];
    offse t s += m, {k, pow, 0, -1}];
        offsets = posns + (pow + 1) m;
        Do[
    sum = Mod[sum + sign^(pow + k) (ncoeffs).(nb^(-k) / offsets), 1];
    offse t s += m, {k, 1, terms}]; reald = RealDigits[sum, base];
   Take[Join[Table[0, {-reald[[2]]}], reald[[1]]], 5]
    ];
```

References

[BBP] D. Bailey, P. Borwein, and S. Plouffe, On the rapid computation of various polylogarithmic constants, *Mathematics of Computation* (forthcoming).

[Cra] R.E. Crandall, *Topics in Advanced Scientific Computation*, Springer/TELOS, New York, 1995.

[RW] S. Rabinowitz and S. Wagon, A spigot algorithm for π, *American Mathemat ical Monthly* **102** (1995) 195-203.