PDQ Unleashed: No More Limitations

Joseph K. Horn

The PDQ Algorithm, introduced at HHC 2003, finds the best fractions that approximate any given decimal number within any specified error limit. The algorithm leveraged off the computer's limitations to generate answers that always agreed with the input, which was also limited in length by the host computer.

The PDQ2 Algorithm, presented here at HHC 2004 for the first time, removes those two limitations. Namely, the input can be a decimal containing any number of digits (regardless of the host computer's limitations), and the output can be of any desired accuracy (not merely up to the computer's innate accuracy). For example, a 12-digit HP calculator can now find the best fraction that approximates a 30-digit decimal to a user-specified tolerance of $\pm 10^{-20}$, or even $\pm 10^{-40}$ or anything the user may wish. The PDQ2 Algorithm is therefore a superset of the functionality of the PDQ Algorithm. PDQ will still be useful for real-world work (e.g. gear ratios), but PDQ2 is necessary for a complete understanding of the relationship between decimal numbers and integer ratios.

HOW THE ORIGINAL PDQ ALGORITHM WORKS

Input decimal number $\xrightarrow{\text{gets named}}$ *Input*

Input error limit $\xrightarrow{\text{gets named}}$ *ErrorLimit*

Step 1: Express the input this way: $\boxed{\frac{Input}{1}}$

Step 2: Multiply that by $\boxed{\frac{10}{10}}$ until the decimal becomes an integer:

$$\frac{Input \times 10^{n}}{10^{n}} \xrightarrow{\text{becomes}} \frac{\text{integer}}{\text{integer}} \xrightarrow{\text{gets named}} \frac{Numerator}{Denominator}$$

It is very important to realize that at this point $\boxed{\frac{Numerator}{Denominator}}$ is *exactly equal* to $\boxed{Input}$.

Step 3: Using Euclid's Algorithm, expand $\frac{Numerator}{Denominator}$ into its continued-fraction equivalent $\{ a_0 \ a_1 \ a_2 \ a_3 \ \ldots \}$, using each partial quotient a_n to generate the next convergent $\frac{p_n}{q_n}$.

Step 4: As soon as $\left| Input - \frac{p_n}{q_n} \right| < ErrorLimit$, stop. This is the **"PDQ Exit Test"**.

Step 5: Use bisection to see if any of the intermediate convergents $\frac{p_n - \left(1 \ldots \frac{a_{n-1}}{2}\right)(p_{n-1})}{q_n - \left(1 \ldots \frac{a_{n-1}}{2}\right)(q_{n-1})}$ are also below the error limit, and if so, find the best one. This intermediate convergent (or $\frac{p_n}{q_n}$ if none is found) $\xrightarrow{\text{gets named}} \frac{P}{Q}$, the final answer.

LIMITATIONS OF PDQ

The "PDQ Exit Test" in step 4 above introduces roundoff error in two ways: (a) by performing the floating-point division $\frac{p_n}{q_n}$, and (b) by performing the floating-point subtraction $Input - \frac{p_n}{q_n}$. This fact is actually used by PDQ as a feature, not a bug: it indicates when to stop, namely, as soon as the algorithm's accuracy is the same as the machine's accuracy.

However, PDQ is thereby limited in three ways:

Limitation #1: None of the "better" fractions between the final (approximate) answer and the actual (mathematically equal) answer can be found by PDQ. Although they obviously exist, they all look equivalent to PDQ because of the roundoff error in Step 4.

Limitation #2: The user is limited to specifying error limits that can be expressed as floating-point decimals (called *ErrorLimit* above). This is also contrary to what the real world wants, namely, to specify not an error limit but a *Tolerance* ratio (e.g. ± three parts per million, or ± 3/16 inches). Rarely do specs call for "

Limitation #3: The user is limited to specifying original decimal numbers (called *Input* above) that can likewise be expressed as a floating-point decimal. This poses a real problem for exploration of long decimals and irrational numbers.

An obvious solution is to use a computer with more floating-point digits... but even then, the intrinsic limitations of PDQ remain. The only way to break through all these limitations is to eliminate all floating-point operations, and rewrite the algorithm to operate entirely in the integer domain. The PDQ2 Algorithm does this.

PDQ2: NO MORE LIMITATIONS

The limitations are all contained in the PDQ Exit Test: $\boxed{\left| Input - \frac{p_n}{q_n} \right| < ErrorLimit}$

All we have to do is (a) replace Decimal with a ratio of two integers, (b) replace ErrorLimit with a Tolerance expressed as the ratio of two integers, and (c) multiply both sides by all the denominators to eliminate all divisions.

(a) Remember from Step 2 above that $\boxed{\frac{Numerator}{Denominator}}$ is *exactly equal* to $\boxed{Input}$. Therefore, the Exit Test can be rewritten as $\boxed{\left| \frac{Numerator}{Denominator} - \frac{p_n}{q_n} \right| < ErrorLimit}$.

(b) The user can be asked to input the Tolerance as a ratio (as is probably expected by most people), or if it's input as a decimal it can be converted to a ratio by recirsively applying PDQ to it (with a tolerance of zero). Thus $\boxed{ErrorLimit \xrightarrow{\text{is replaced by}} Tolerance \xrightarrow{\text{which is expressed as}} \frac{A}{B}}$. This allows the Exit Test to be rewritten again, as $\boxed{\left| \frac{Numerator}{Denominator} - \frac{p_n}{q_n} \right| \le \frac{A}{B}}$.

(c) Since the denominators are all positive integers, we can multiply both sides by them all, obtaining the new PDQ2 Exit Test:

$$\boxed{Numerator \cdot q_n \cdot B \; - \; p_n \cdot Denominator \cdot B \; \le \; A \cdot Denominator \cdot q_n}$$

This can be simplified slightly to this:

$$\boxed{B\left(Numerator \cdot q_n \; - \; p_n \cdot Denominator\right) \; \le \; A \cdot Denominator \cdot q_n}$$

No divisions at all! No subtractions of floating-point numbers, only integers! This enables the PDQ2 algorithm to run identically on all computers regardless of how many floating-point digits they use, since none are used.

Finally, the user can also input ratios of integers of any size, and obtain *all* the "best" fractions that approximate it, even those that the computer at hand would be unable to distinguish

EXAMPLE 1:

The HP48's description in Craig Finseth's "HP Database" Hall of Fame includes this bug report:

```
The sequence:

     STD 19 1/x e-12 - \->Q

Produces a fraction that clearly has not been reduced to
lowest terms.
```

Namely, it returns this ludicrous fraction:

1.0101010101E20 / 1.91919191922E21

The HP49 "solves" this problem by converting these to integers and reducing it to:

3367003367 / 63973063974

But that's just the same bad answer with a cosmetic makeover. Neither of these fractions evaluate to the original input; they are both off by 1 in the last place.

The best possible answer (and therefore, to my mind, the ONLY answer) is:

2719978241 / 51679586580

which exactly evaluates to the input. Although this answer can be found with my old DEC2FRAC program, it would take a lot of trial and error to do so since DEC2FRAC requires you to know the upper limit of the denominator. It can also be found with the Stern-Brocot Tree method, but it would take a very long time. It is missed entirely by the continued fraction method. The only way to find it within one's lifetime is with →PDQ, which finds the answer in 3.4 seconds.

EXAMPLE 2:

According to the HP manuals, the accuracy of the →Q function can be reduced by using smaller FIX settings, and the highest accuracy that can be obtained is in STD mode.

Therefore, in STD mode, the →Q command should always give results that are of the same or greater accuracy than at smaller FIX settings.

But such is not the case. Although STD →Q returns ratios of larger integers than FIX →STD, they are not necessarily more accurate.

For example (first in STD mode, then in 10 FIX mode):

3 √ ln DUP STD →Q ⟶ 750826905227/1366864203080

→NUM – ⟶ -.000000000003 (off by 3 in the 12th place)

3 √ ln DUP 10 FIX →Q ⟶ 379013/689985
→NUM – ⟶ -.000000000001 (off by 1 in the 12th place)

(The best fraction, 782647/1424792, is missed by →Q and DEC2FRAC, but is found by →PDQ.)

Therefore STD →Q is here less accurate than 10 FIX →Q, even though the ratio it yields contains larger integers and therefore ought to be more accurate.

This is just one example. This behavior is not rare, however. STD →Q is less accurate than 10 FIX →Q in an uncountable number of cases; it seems that any randomly picked number has roughly an 8% chance of
showing the same behavior.

EXAMPLE 3:

2 √ STD →Q ⟶ 941664/665857 (accurate to all 12 digits) ...
....But the fraction 665857/470832 is accurate to all 12 digits too,
yet there is no way to obtain this "better" result from →Q.

3 √ STD →Q ⟶ 978122/564719 (accurate to all 12 digits) ...
....But the fraction 716035/413403 is accurate to all 12 digits too,
yet there is no way to obtain this "better" result from →Q.

Examples of this kind abound. Of course, →PDQ finds the best answers. It *always* does.

EXAMPLE 4:

All of the following inputs to the following functions are converted exactly to the best possible fraction by →PDQ, but cannot be obtained by the continued fraction method:

$\sqrt{x}$:
2,3,12,13,24,26,32,47,48,51,54,56,57,58,59,62,63,65,67,70,72,74,78,82,83,84,85,89,96,102...

$\sqrt[3]{x}$:
2,4,6,9,11,13,14,16,31,36,37,41,47,55,58,65,70,71,74,82,86,88,90,91,92,98,103...

LN(X):
3,5,12,15,23,24,26,31,32,33,35,37,41,49,52,58,66,73,77,82,85,87,89,91,93,95,97,101...

LOG(X):
2,4,5,7,8,15,17,18,21,26,34,35,37,42,45,46,49,53,56,61,70,74,76,77,83,84,86,87,91,92,93,96,98,101...

SIN(X degrees):
2,4,6,11,12,14,16,17,19,20,26,27,31,33,38,40,44,51,57,60,62,63,64,65,67,71,72,74,76,81,87,88...

TAN(X degrees):
2,3,5,9,11,14,15,16,17,19,24,28,31,32,36,39,40,47,52,53,54,59,60,62,66,69,75,76,79...

ACOSH(X):
6,8,9,11,13,15,16,19,25,27,33,37,39,45,52,55,57,61,63,66,69,71,83,86,87,107...

ASINH(X):
2,4,7,9,16,17,21,22,30,36,38,44,47,49,52,55,56,59,60,65,67,68,69,71,79,80,88,89,90,92,93,99,103...

APPENDIX:

Here's the actual continued-fraction expansion of the actual value of π. Calculated by Derive with NUMBER.MTH loaded. Accuracy set to 500 digits. Notice the large partial quotient 20776, unmentioned in any literature I've seen (old books say 292 is the largest known partial quotient; newer books are mum on the subject).

```
 00:  3 7 15 1 292 1 1 1 2 1
 10:  3 1 14 2 1 1 2 2 2 2
 20:  1 84 2 1 1 15 3 13 1 4
 30:  2 6 6 99 1 2 2 6 3 5
 40:  1 1 6 8 1 7 1 2 3 7
 50:  1 2 1 1 12 1 1 1 3 1
 60:  1 8 1 1 2 1 6 1 1 5
 70:  2 2 3 1 2 4 4 16 1 161
 80:  45 1 22 1 2 2 1 4 1 2
 90:  24 1 2 1 3 1 2 1 1 10
100:  2 5 4 1 2 2 8 1 5 2
110:  2 26 1 4 1 1 8 2 42 2
120:  1 7 3 3 1 1 7 2 4 9
130:  7 2 3 1 57 1 18 1 9 19
140:  1 2 18 1 3 7 30 1 1 1
150:  3 3 3 1 2 8 1 1 2 1
160:  15 1 2 13 1 2 1 4 1 12
170:  1 1 3 3 28 1 10 3 2 20
180:  1 1 1 1 4 1 1 1 5 3
190:  2 1 6 1 4 1 120 2 1 1
200:  3 1 23 1 15 1 3 7 1 16
210:  1 2 1 21 2 1 1 2 9 1
```

```
220:  6 4 127 14 5 1 3 13 7 9
230:  1 1 1 1 1 5 4 1 1 3
240:  1 1 29 3 1 1 2 2 1 3
250:  1 1 1 3 1 1 10 3 1 3
260:  1 2 1 12 1 4 1 1 1 1
270:  7 1 1 2 1 11 3 1 7 1
280:  4 1 48 16 1 4 5 2 1 1
290:  4 3 1 2 3 1 2 2 1 2
300:  5 20 1 1 5 4 1 436 8 1
310:  2 2 1 1 1 1 1 5 1 2
320:  1 3 6 11 4 3 1 1 1 2
330:  5 4 6 9 1 5 1 5 15 1
340:  11 24 4 4 5 2 1 4 1 6
350:  1 1 1 4 3 2 2 1 1 2
360:  1 58 5 1 2 1 2 1 1 2
370:  2 7 1 15 1 4 8 1 1 4
380:  2 1 1 1 3 1 1 1 2 1
390:  1 1 1 1 9 1 4 3 15 1
400:  2 1 13 1 1 1 3 24 1 2
410:  4 10 5 12 3 3 21 1 2 1
420:  34 1 1 1 4 15 1 4 44 1
430:  4 20776  1 1 1 1 1 1 1 23
440:  1 7 2 1 94 55 1 1 2 1
450:  1 3 1 1 32 5 1 14 1 1
460:  1 1 1 3 50 2 16 5 1 2
470:  1 4 6 3 1 3 3 1 2 2
480:  2 5 2 2 2 28 1 1 13 1
490:  5 43 1 ...
```

REFERENCES FOR MATHEMATICAL ARCHEOLOGY :

Filler comment: "2549491779/811528438 will yield 18 accurate digits for pi." *65 NOTES*, V1 N7 (December 1974) P1b. [Note: π 17.3 PDQ yields a better answer: 1068966896/340262731.]

"Fractions" by David A. Kemper, *65 NOTES*, V3 N5 P14. An HP-65 program for D→F and F→D. Uses continued fractions, 1/FP(x).

"Fractions" by David A. Kemper, *65 NOTES*, V3 N9 P27. An HP-97 program for D→F and F→D with optional mixed number display for improper fractions. Same logic as his HP-65 program in V3 N5 P14.

"HP-25 Decimal to Fraction" by John Kennedy & Jim Davidson, *65 NOTES*, V4 N1 P16-19. "The decimal to fraction conversion appearing in this month's HP-25 Library (elsewhere in this issue) has minimal execution time and handles the computations in an astute manner. ... The main idea behind such a program is to multiply F [the decimal number] by successive integers until an integer is obtained." Sample running time: π → 355/113 in 16 seconds; 104348/33215 in 45 minutes! [Note: π 9.3 PDQ → 104348/33215 in .87 seconds].

"Number Theory" (P17b-19c) and "Decimal-to-Fraction Conversion (HP-25)" (P20c) by John Kennedy and Jim Davidson, *65 NOTES*, V4 N6. Program on page 61. Introduced and thoroughly explained the continued fraction method. "Of course every number represented in a digital computer is a rational number. A decimal number in the HP-25 is really a fraction whose numerator is the integer with the same digits as x and whose denominator is an appropriate power of 10." [THEY ALMOST HAD IT!] Sample running time: $\pi \rightarrow$ 104348/33215 in 10 seconds. "Since the calculator works with rational numbers only, it is always possible to achieve an exact (at least to the limits of the machine) rational approximation." [Note: Not true! E.g. .9999999975 and just about any other value with a sufficiently large partial quotient.]

"Decimals & Fractions" by John Kennedy, *PPC Journal* V5 N6 (July 1978) P17b. "If one were to choose a number at random from the real number system then the probability that the number chosen is transcendental is 1 and the probability that the number chosen is a rational or non-transcendental irrational is 0. So perhaps you can begin to understand why such numbers are so important."

"34C Decimal to Fraction" by Thomas S. Cox, *PPC Calculator Journal*, V7 N2 (Feb-Mar 1980) P11d. Same logic as horrible HP-25 program in V4N1P19; $\pi \rightarrow$ 355/113 in 19 seconds.

"Converting a Decimal to a Fraction" by Stanley Becker, *PPC Calculator Journal* V9 N5 (Aug 1982) P22b. TI-59 program. Stores all the partial quotients like HP's →Q function does and like the HP32SII does. An input of the golden ratio takes 33 registers and 10 minutes. Continued fractions, 1/FP(x).

"HP-75 Number To Fraction" by Anthony J. Mechelynck, *Computer Journal of PPC*, V2 N4 (Jul/Aug 1983) P25d. Continued Fractions, 1/FP(x).

"HP-15C Decimal to Fraction Conversion" by Bob Hall, *PPC Journal* V12 N1 P13

"Decimal to Fraction Conversion Using Farey Fractions" by Bob Hall, *PPC Journal* V12 N4 (April 1985) P38a-c.

"DF" HP-41 Decimal-to-Fraction program, PPC ROM. Continued fractions, 1/FP(x).

"Decimal Number to Fraction Once More" by Roger K. Heinsohn, *CHHU Chronicle* V2 N7 (Nov/Dec 1985) P42b. HP-41 program. Uses same technique as PPC ROM.

"Decimal-To-Fraction Conversion" by Joseph K. Horn, *HPX Exchange*, V1 N4 (July 1987 - March 1988) P5c. "For example, what fraction approximates PI to 8 digits? CALL D2F(PI,8,A,B) and then DISP A;B. 103993/33102 is a good approximation of PI." [Note: π 8 PDQ yields a better answer: 100798/32085.]

HP-67 Math Pac by HP: Uses continued fractions for D→F on pages 02-01 through 02-06.

Sparcom *Math Pro Pac*; uses Joe Horn's DEC2FRAC algorithm in its FRACT function which finds intermediate convergents but suffers from 1/FP(x) error accumulation.

Hardy & Wright's *Number Theory*, chapters X (Continued Fractions) and XI (Approximation of Irrationals by Rationals) brush "intermediate convergents" under the rug, since they have low Hurwitz accuracies. "Good approximation" and "best approximation" are defined on page 151.

The "final word" on the subject (referenced as the definitive work by others) is *Best Lower and Upper Approximations to Irrational Numbers* by Clark Kimberling, professor of Mathematics at the University of Evansville (Indiana). He makes a distinction between "best approximate" and "nearest approximate", to wit: |qa-p| < |ca-b| means that p/q is a "best approximate" to a, which is stronger (or "better") than |a-p/q| < |a-b/c| which means that p/q is nearer a than any b/c having c<q. "Given a positive irrational number a, the principal convergents pi/qi to a are well known to be the best approximates to a." He cites as his expert sources these two books:

Serge Lang, *Introduction to Diophantine Approximations*, Addison-Wesley, Reading, Mass., 1966.

Oskar Perron, *Die Lehre von den Kettenbruchen*, Chelsea, New York, 1950.

Computing Science: On the Teeth of Wheels by Brian Hayes, *American Scientist*, Volume 88, No. 4 (July-August 2000), available online at:
http://www.sigmaxi.org/amsci/issues/Comsci00/compsci2000-07.htm
Covers the Stern-Brocot Tree better than Knuth does. "The Zeroth Law of Gearwork: The number of teeth on a gear must be an integer." Also: "The fact is, the design of simple gear trains is no longer a computationally interesting problem. I am reminded of those prodigies who spent years of their lives calculating digits of the decimal expansion of pi. I cannot help wondering which of my own labors will appear equally quaint and pathetic to some future reader who ransacks libraries for old volumes of *American Scientist*." [Answer: this one; me; 3 years later.]

Concrete Mathematics by Graham, Knuth & Patashnik (1994), pages 116-123 is about the Stern-Brocot Tree. It ends with, "A little fudging is needed at the end to make sure that there aren't infinitely many R's [partial quotients]".

So That's Why 22/7 is Used for PI! by Maurice J. Burke and Diana L. Taggert, *Mathematics Teacher* V95 N3 (March 2002) P164-169. "The next better approximation to pi does not occur until the denominator is 16604." [Note: That's an intermediate convergent, missed by the continued fraction algorithm. PDQ finds it if the accuracy is specified just a hair above 355/113's accuracy; try 6.57477 or thereabouts. In any case, it's proof that the NCTM considers intermediate convergents to be "better", unlike most mathematicians. But the NCTM dismisses the creeping inaccuracy of 1/FP(x) by saying, "More precise does not necessarily mean more useful."