Small Fry – Let's Be Rational

Valentín Albillo (HPCC #1075)

There are times both in real-life applications (engineering, f.i.) and pure math calculations where we need to convert a real value, given either in symbolic form or as a decimal number, into a rational approximation which agrees with it to some specified precision. The following short **HP-71B** subprogram does just that:

```
100 SUB DEC2FRC(X,N,D,W) @ V=1 @ N=1 @ D=0 @ Y=INF @ Z=ABS(X) @ F=SGN(X) @ X=Z
110 C=INT(X) @ IF FP(X) THEN X=1/FP(X) @ S=N ELSE N=(N*C+U)*F @ D=D*C+V @ END
120 T=D @ N=N*C+U @ U=S @ D=D*C+V @ V=T @ R=N/D @ IF ABS(R/Z-1)<=W THEN N=N*F @ END
130 IF R=Y OR MAX(N,D)>1E12 THEN N=U*F @ D=V ELSE Y=R @ GOTO 110
```

DEC2FRC: The Convert-Real-To-Fraction subprogram

This 4-line (*229 bytes*) subprogram converts a given *real* value to the simplest equivalent *fraction* within a user-specified max. relative error. Its calling syntax is:

```
CALL DEC2FRC(X,N,D,W) , where:

     X    input : real value to convert to fractional form
     N    output: integer numerator of the simplest fraction
     D    output: integer denominator of the simplest fraction
     W    input : maximum relative error (0 means maximum accuracy)
```

It uses ***continued fractions*** to generate *convergents* until one meets the specified tolerance, which is also guaranteed to be in its lowest terms, i.e.: numerator and denominator have no common factor.

Examples: Convert *p* to a rational within **1E-7**, then with maximum accuracy:

```
>CALL DEC2FRC(PI,N,D,1E-7) @ N;"/";D,N/D;PI
     355 / 113              3.14159292035  3.14159265359

>CALL DEC2FRC(PI,N,D,0) @ N;"/";D,N/D;PI
     1146408 / 364913       3.14159265359  3.14159265359
```

You can also use it to perform exact rational arithmetic ...

```
>CALL DEC2FRC(1+1/2+1/3+1/4+1/5+1/6+1/7,N,D,0) @ N;"/";D
      363 / 140
```

... or to reduce fractions to lowest terms:

```
>CALL DEC2FRC(-15318/16169,N,D,0) @ N;"/";D
      -18 / 19
```

Specifying maximum precision (**W=0**) will frequently get you an exact fit but sometimes at the expense of unnecessarily large terms. Compare these two runs:

```
>CALL DEC2FRC(0.66666666668,N,D,0) @ N;"/";D
     16666666667 / 25000000000    {= 0.66666666668 }

>CALL DEC2FRC(0.66666666668,N,D,1E-10) @ N;"/";D
     2 / 3  {= 0.66666666667 }
```

As a bonus, you can see the continued fraction representation of a given value by doing this simple modification: change line

```
110 C=INT(X) @ IF FP(X) ...
```

to:

```
110 C=INT(X) @ DISP C; @ IF FP(X) ...
```

Then, upon calling it, all terms of the continued fraction will be displayed, e.g.:

```
>CALL DEC2FRC(PI,N,D,0) @ DISP
     3  7  15  1  292  1  1  1  2  1
```

so *p* 's continued fraction is **(3,7,15,1,292,1,1,1,2,1,...)** which evaluates to:

```
>3+1/(7+1/(15+1/(1+1/(292+1/(1+1/(1+1/(1+1/(2+1/1)))))))), PI
     3.14159265359    3.14159265359
```

Yet another simple modification will output the successive convergents as they're being computed. If you made the previous modification first restore line 110 to its original form (remove the `DISP C; @` statement), and then change line

```
120 T=D @ N=N*C+U @ U=S @ D=D*C+V @ V=T @ R=N/D @ IF ABS ...
```

to:

```
120 T=D @ N=N*C+U @ U=S @ D=D*C+V @ V=T @ R=N/D @ DISP N;D @ IF ABS ...
```

and then, upon calling it, all successive convergents will be displayed:

```
>CALL DEC2FRC(PI,N,D,0)
     3  1             {3/1            = 3            }
     22  7            {22/7           = 3.14285714286}
     333  106         {333/106        = 3.14150943396}
     355  113         {355/113        = 3.14159292035}
     103993  33102    {103993/33102   = 3.14159265301}
     104348  33215    {104348/33215   = 3.14159265392}
     208341  66317    {208341/66317   = 3.14159265347}
     312689  99532    {312689/99532   = 3.14159265362}
     833719  265381   {833719/265381  = 3.14159265358}
     1146408  364913  {1146408/364913 = 3.14159265359}
```

You might want to alter the routine to accept additional parameters so you can leave the modifications permanently available but optional, offer the choice of just display or return the continued fraction/convergents to the caller, or even convert the routine to **RPN** or **RPL**. But that's left as an exercise for the reader.

Though very capable by itself, **DEC2FRC** can also be used as a building block to create much more complex programs. See ***"Boldly Going - Identifying Constants"*** in this same issue of ***Datafile*** for a particularly impressive application.