Algorithm To Convert A Decimal To A Fraction

by

John Kennedy
Mathematics Department
Santa Monica College
1900 Pico Blvd.
Santa Monica, CA 90405

jrkennedy6@gmail.com
Except for this comment explaining that it is blank for some deliberate reason, this page is intentionally blank!
CONVERTING DECIMALS TO FRACTIONS

Let X denote the original decimal. In the following algorithm description we assume X > 0. In the code example we take into account the cases where X = 0.0 or where X < 0.0 or where X is already an exact integer. We define two recursive sequences, Z_i and D_i and we define one non-recursive sequence N_i. The fractions

\[
\frac{N_i}{D_i}
\]

will approximate the original decimal X. In fact, these fractions oscillate below and above X and converge to X. The sequence Z_i is related to the continued fraction approximation to X and is otherwise used only to help find the D_i which are the important values that the algorithm finds and returns. The sequences Z_i and D_i are initialized with the following values for i = 0 and i = 1.

\[ Z_0 \text{ is undefined. } Z_1 = X \quad D_0 = 0 \quad \text{and} \quad D_1 = 1 \]

For i = 1, 2, 3, ... we calculate the following values in the order Z_{i+1} first, then D_{i+1}, and finally N_{i+1} as shown below.

\[
Z_{i+1} = \frac{1}{Z_i - \text{Int}(Z_i)} \quad \text{Int}() = \text{integer part function}
\]

\[
D_{i+1} = D_i \times \text{Int}(Z_{i+1}) + D_{i-1}
\]

\[
N_{i+1} = \text{Round}(X \times D_{i+1})
\]

Round() = rounds to the nearest integer.

Note that once D_i is accurately known, the corresponding N_i value is trivial to find. The real value of the algorithm is in specifying the calculation of the D_i sequence.

First Example: \( X = \frac{5}{19} = 0.263157894737 \)

\[
\begin{array}{|c|c|c|c|c|}
\hline
i & Z_i & N_i & D_i & \frac{N_i}{D_i} \\
\hline
0 & \text{undefined} & \text{undefined} & 0 & \text{undefined} \\
1 & 0.263157894737 & 0 & 1 & 0.000000000000 \\
2 & 3.8 & 1 & 3 & 0.333333333333 \\
3 & 1.25 & 1 & 4 & 0.250000000000 \\
4 & 4. & 5 & 19 & 0.263157894737 \\
5 & \text{undefined} & \text{undefined} & \text{undefined} & \text{undefined} \\
\hline
\end{array}
\]
Second Example: \( X = \pi = 3.14159265359 \)

<table>
<thead>
<tr>
<th>(i)</th>
<th>(Z_i)</th>
<th>(N_i)</th>
<th>(D_i)</th>
<th>(\frac{N_i}{D_i})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(3.14159265359)</td>
<td>-----</td>
<td>0</td>
<td>(3.00000000000)</td>
</tr>
<tr>
<td>1</td>
<td>(7.06251330592)</td>
<td>22</td>
<td>7</td>
<td>(3.14285714286)</td>
</tr>
<tr>
<td>2</td>
<td>(15.9965944095)</td>
<td>333</td>
<td>106</td>
<td>(3.14150943396)</td>
</tr>
<tr>
<td>3</td>
<td>(1.00341722818)</td>
<td>355</td>
<td>113</td>
<td>(3.14159292035)</td>
</tr>
<tr>
<td>4</td>
<td>(292.63483365)</td>
<td>103993</td>
<td>33102</td>
<td>(3.14159265301)</td>
</tr>
<tr>
<td>5</td>
<td>(1.57521580653)</td>
<td>104348</td>
<td>33215</td>
<td>(3.14159265392)</td>
</tr>
<tr>
<td>6</td>
<td>(1.7384779567)</td>
<td>208341</td>
<td>66317</td>
<td>(3.14159265347)</td>
</tr>
<tr>
<td>7</td>
<td>(1.35413656011)</td>
<td>312689</td>
<td>99532</td>
<td>(3.14159265362)</td>
</tr>
<tr>
<td>8</td>
<td>(2.82376945122)</td>
<td>833719</td>
<td>265381</td>
<td>(3.14159265358)</td>
</tr>
<tr>
<td>9</td>
<td>(1.21393188169)</td>
<td>1146408</td>
<td>365381</td>
<td>(3.14159265359)</td>
</tr>
<tr>
<td>10</td>
<td>(4.67438509913)</td>
<td>4585632</td>
<td>1459652</td>
<td>(3.14159265359)</td>
</tr>
</tbody>
</table>

For this example, the more iterates that are made, the larger the numerators and denominators of the approximating fractions. Since there is no change between the last two fraction approximations (when the fractions are converted back to decimals they yield the same decimal values which appear in the rightmost column) the algorithm can be stopped after the 11th step.

Third Example: \( X = \frac{37}{61} = 0.606557377049 \)

<table>
<thead>
<tr>
<th>(i)</th>
<th>(Z_i)</th>
<th>(N_i)</th>
<th>(D_i)</th>
<th>(\frac{N_i}{D_i})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(0.606557377049)</td>
<td>-----</td>
<td>0</td>
<td>(0.00000000000)</td>
</tr>
<tr>
<td>1</td>
<td>(1.64864864865)</td>
<td>0</td>
<td>1</td>
<td>(0.00000000000)</td>
</tr>
<tr>
<td>2</td>
<td>(1.54166666666)</td>
<td>1</td>
<td>2</td>
<td>(0.50000000000)</td>
</tr>
<tr>
<td>3</td>
<td>(1.84615384618)</td>
<td>2</td>
<td>3</td>
<td>(0.66666666666)</td>
</tr>
<tr>
<td>4</td>
<td>(1.18181818181)</td>
<td>3</td>
<td>5</td>
<td>(0.60000000000)</td>
</tr>
<tr>
<td>5</td>
<td>(5.5)</td>
<td>17</td>
<td>28</td>
<td>(0.607142857143)</td>
</tr>
<tr>
<td>6</td>
<td>(2.)</td>
<td>37</td>
<td>61</td>
<td>(0.606557377049)</td>
</tr>
<tr>
<td>7</td>
<td>undefined!</td>
<td>-----</td>
<td>-----</td>
<td>(\text{undefined!} )</td>
</tr>
</tbody>
</table>
The following code fragment is Turbo Pascal code that converts a decimal to a single fraction.

When converting this code fragment to another language the following remarks may be helpful.

The extended data type can be replaced by any floating point or real number data type.

The Abs function is the Absolute Value function.

The Int function is the Integer Part function. For example, \( \text{Int}(3.75) = 3 \) and \( \text{Int}(−2.3) = −2 \).

The variable \( Z \) is used to represent the above sequence variable \( Z_i \).

The variable FractionNumerator is used to represent the above sequence variable \( N_i \).

The variable FractionDenominator is used to represent the above sequence variable \( D_i \).

The variable PreviousDenominator is used to represent the above sequence variable \( D_{i−1} \).

The value of AccuracyFactor is used to determine how accurate the conversion needs to be. For example, if \( \text{AccuracyFactor} = 0.0005 \) then the conversion should be accurate to 3 decimal places. To get accuracy to 5 decimal places set the \( \text{AccuracyFactor} = 0.000005 \). The higher the \( \text{AccuracyFactor} \) the larger but more accurate is the fraction that is returned.

The code that executes first saves the sign of \( X \) and then takes the absolute value so the algorithm really only works on nonnegative decimals. The first test checks if \( X \) is already an exact whole number. In this case the denominator is set to 1 and the procedure terminates immediately. Note that this case includes the possibility that \( X = 0 \).

Next, the code checks to see if the decimal is smaller than the smallest representable fraction. If so, the smallest representable fraction is returned. Note that if \( X=0 \) the if-statement test would fail to take this case into account, but we have already handled the case where \( X = 0 \). Zero is a special case of the truly smallest representable fraction. So we really mean the smallest nonzero representable fraction!

Next it checks if the decimal is larger than the largest representable fraction. If so, the largest representable fraction is returned.

Failing the above 3 checks, the algorithm finally begins by going into an iteration loop in which the real work is done. This loop is guaranteed to execute at least once. The value \( \text{AccuracyFactor} \) helps determine when to stop with the current fraction approximation. We must also stop if and when \( Z \) becomes an exact integer.
procedure DecimalToFraction (Decimal : extended;
  var FractionNumerator : extended;
  var FractionDenominator : extended;
  AccuracyFactor : extended);

var  DecimalSign         : extended;
     Z                   : extended;
 PreviousDenominator : extended;
 ScratchValue        : extended;

begin
  if Decimal<0.0  then  DecimalSign := -1.0  else  DecimalSign := 1.0;
  Decimal := Abs(Decimal);
  if Decimal=Int(Decimal) then  { handles exact integers including 0 }
  begin
    FractionNumerator := Decimal*DecimalSign;
    FractionDenominator := 1.0;
    Exit
  end;
  if  (Decimal < 1.0E-19)  then  { X=0 already taken care of }
  begin
    FractionNumerator := DecimalSign;
    FractionDenominator := 9999999999999999999.0;
    Exit
  end;
  if  (Decimal > 1.0E+19)  then
  begin
    FractionNumerator := 9999999999999999999.0*DecimalSign;
    FractionDenominator := 1.0;
    Exit
  end;
  Z := Decimal;
  PreviousDenominator := 0.0;
  FractionDenominator := 1.0;
  repeat
    Z := 1.0/(Z - Int(Z));
    ScratchValue := FractionDenominator;
    FractionDenominator := FractionDenominator*Int(Z)+PreviousDenominator;
    PreviousDenominator := ScratchValue;
    FractionNumerator := Int(Decimal*FractionDenominator + 0.5)  { Rounding Function }
    until
      (Abs((Decimal - (FractionNumerator/FractionDenominator))))
        < AccuracyFactor
      OR (Z = Int(Z));
  FractionNumerator := DecimalSign*FractionNumerator
end;  {procedure DecimalToFraction}