Rational Approximations Package for REDUCE

Lisa Temme
Wolfram Koepf
e-mail: `koepf@zib.de`
& Alan Barnes
email: `Alan.Barnes45678@gmail.com`

August 1995 & February 2019

1 Periodic Decimal Representation

The division of one integer by another often results in a period in the decimal part. The `rational2periodic` function in this package can recognise and represent such an answer in a periodic representation. The inverse function, `periodic2rational`, converts a periodic representation back to a rational number.

Periodic Representation of a Rational Number

SYNTAX: `rational2periodic(n);`
`rational2periodic(n, b);`

INPUT: `n` is a rational number
`b` is the number base, if absent the default is 10.

RESULT: `periodic({a1,...,an},{b1,...,bm},{c1,...,ck},`$\pm$`b)`
where `{a1,...,an}` is a list of the digits in the integer part,
`{b1,...,bm}` is a list of the digits in the non-periodic part,
`{c1,...,ck}` is a list of the digits in the periodic part
and $\pm$`b` where `b` is the number base $2 \leq \mathrm{b} \leq 16$,
a minus indicating the rational number `n` was negative.

EXAMPLES:

$-59/70$ written as $-0.8\overline{428571}$

```
1:  rational2periodic(-59/70);
 periodic({0}, {8}, {4,2,8,5,7,1}, -10)
```

$1/80$ written as a hexadecimal is $0.0\overline{3}$

```
2:  rational2periodic(1/80,16);
 periodic({0}, {0}, {3}, 16)
```

Normally the operator `periodic` will not be seen as the output will be prettyprinted as $-0.8\overline{428571}$ and $0.0\overline{3}$ (base 16) respectively. Currently pretty-printed output looks better when the switch `FANCY` is `OFF`.

Rational Number of a Periodic Representation

SYNTAX:

`periodic2rational(periodic({a1,...,an},{b1...bm},{c1,...,ck},±b)`

`periodic2rational({a1,...,an},{b1...bm},{c1,...,ck},±b)`

INPUT:

`{a1,...,an}` is a list of the digits in the integer part,
`{b1,...,bm}` is a list of the digits in the non-periodic part,
`{c1,...,ck}` is a list of the digits in the periodic part
and `b` is the number base $2 \leq \mathrm{b} \leq 16$, a minus
indicating the rational number result should be negative.
If the base is omitted, 10 is assumed.

RESULT:

A rational number.

EXAMPLES:

$0.8\overline{428571}$ written as $59/70$

```
3:  periodic2rational(periodic({0},{8},{4,2,8,5,7,1}));

 59
---
 70

4:  periodic2rational({0},{8},{4,2,8,5,7,1}, -10);

   59
 - ---
   70
```

Note that `periodic2rational` will produce the correct rational result when passed a parameter for the periodic part which is not minimal. Similarly, a parameter for the periodic part which consists of all 9's (or in base b, all $(b-1)$'s) is treated correctly although such periodic parts are not canonical and are never generated by calls to `rational2periodic`.

For example,

```
periodic2rational({0}, {}, {1, 2, 1, 2});
periodic2rational({0}, {1}, {2, 1});
periodic2rational({0}, {1, 2}, {1, 2, 1, 2});
```

all produce the same rational result, namely $\frac{4}{33}$, as the canonical input

```
periodic2rational({0}, {}, {1, 2});
```

Similarly,

```
periodic2rational({0}, {}, {9});
periodic2rational({0}, {9}, {9});
periodic2rational({0}, {}, {9, 9, 9, 9});
```

all produce the same rational result, namely 1, as the canonical input

```
periodic2rational({1}, {}, {});
```

Although the operators `periodic2rational` and `rational2periodic` work even when `ROUNDED` is `ON`, they are best used when `ROUNDED` is `OFF`. The input to `rational2periodic` should not be a rounded number, otherwise an error results.

For example, the input `rational2periodic(1/7);` will produce the intended periodic representation even with `ROUNDED ON`. However, the input

```
a := 1/7; rational2periodic(a);
```

will result in an error as the simplifier is applied in the assignment and rounds the rational number.

Similarly, although the result of `periodic2rational` will always be a rational number (represented by a `QUOTIENT` prefix form), if the simplifier is applied to the result a rounded value will be produced.

2 Continued Fractions

A continued fraction (see [**?**]) has the general form

$$a_0 + \frac{a_1}{b_1 + \frac{a_2}{b_2 + \frac{a_3}{b_3 + \dots}}} .$$

A more compact way of writing this is as

$$a_0 + \frac{a_1|}{|b_1} + \frac{a_2|}{|b_2} + \frac{a_3|}{|b_3} + \ldots .$$

Even more succinctly:

$$\{a_0,\ \{a_1, b_1\},\ \{a_2, b_2\},\ \ldots\}$$

This is represented in REDUCE as

$$\texttt{contfrac}(\mathit{Expression}, \mathit{Rational\ approximant}, \{\texttt{a0}, \{\texttt{a1}, \texttt{b1}\}, \{\texttt{a2}, \texttt{b2}\}, ...\})$$

The operator `CFRAC` is used to generate a generalised continued fraction expansion of an algebraic expression.

```
cfrac(⟨num⟩)
cfrac(⟨num⟩,⟨length⟩)
cfrac(⟨func⟩,⟨var⟩)
cfrac(⟨func⟩,⟨var⟩,⟨length⟩)
```

INPUT:	⟨*num*⟩	is any real number
	⟨*func*⟩	is a function
	⟨*var*⟩	is the function main variable
	⟨*length*⟩	is the maximum number of terms (continuents) to be generated and is **optional**.

For non-rational function or irrational number input the ⟨*length*⟩ argument specifies the number of continuents (ordered pairs, $\{a_i, b_i\}$), to be returned. Its default value is five. For rational function or rational number input the `length` argument can only truncate the answer, it cannot return additional pairs even if the precision is increased. The default for rational function or rational number input is the complete continued fraction.

For a non-rational function, power series expansion is necessary. The new switch `cf_taylor` controls whether the TAYLOR or the TPS package is used to produce the power series required. By default this switch is OFF and so the TPS package is normally employed. In most cases the choice is not important, but the TPS option is somewhat better at handling cases where the series expansion is rather sparse. In a few cases TPS may fail to produce a series expansion when TAYLOR succeeds and vice-versa.

For numerical input the default value is exact for rational number arguments whilst for irrational or rounded input it is dependent on the precision of the session. The length argument will only take effect if is smaller than the number of ordered pairs which the default value would return.

If the number of continuent pairs returned does not exceed twelve, the result will usually be pretty-printed as a two element list consisting of the convergent followed by a rendering of the traditional continued fraction expansion. For a larger number of pairs the output is of the second element is printed as a list of pairs. Thus, usually the operator contfrac is not seen in the output.

EXAMPLES

```
cfrac(pi, 4);

     355               1
{pi,-----,3 + ----------------}
     113                 1
               7 + ----------
                            1
                    15 + ---
                            1

cfrac(sqrt 2, 5);

          41                 1
{sqrt(2),----,1 + ----------------------}
          29                   1
                   2 + ---------------
                                 1
                        2 + ---------
                                   1
                             2 + ---
                                   2

cfrac(23.696, 4);

  2962   237               1
{------,-----,23 + ---------------}
  125    10                  1
                    1 + ---------
                                1
                         2 + ---
                                3

cfrac((x+2/3)^2/(6*x-5), x, 10);
```

```
       2
   9*x   + 12*x + 4
{-----------------,   exact,
      54*x - 45

    6*x + 13          1
  ---------- + -------------}
      36        24*x - 20
               -----------
                    9

cfrac(e^x, x);


        3      2
  x    x  + 9*x  + 36*x + 60
{e , -----------------------,
            2
       3*x  - 24*x + 60

                  x
 1 + ---------------------------}
                    x
      1 - ---------------------
                      x
           2 + ---------------
                        x
                3 - ---------
                          x
                     2 + ---
                          5
```

The operator `CF` is a synonym for the operator `CONTINUED_FRACTION`.

> `cf(`⟨*num*⟩`)`
> `cf(`⟨*num*⟩`,` ⟨*size*⟩`)`
> `cf(`⟨*num*⟩`,` ⟨*size*⟩`,` ⟨*numterms*⟩`)`

The meaning of the arguments is the same as for the operator `CONTINUED_FRACTION`: the original number to be expanded ⟨*num*⟩, an optional maximum size ⟨*size*⟩ permitted for the denominator of the convergent and an optional maximum number of continuents ⟨*numterms*⟩ to be generated.

The output is in the same format as that of `CFRAC` described above. As with the operator `CFRAC` output of `CF` is normally pretty-printed so the operator `confract` will not be seen.

The accessor operators `CF_EXPRESSION`, `CF_CONVERGENT` and `CF_CONTINUENTS` allow the various parts of a continued fraction object ⟨*cf_object*⟩ (as returned by any of the operators `cf`, `cfrac`, `continued_fraction` and `cf_euler`) to be extracted.

These three operators return, respectively, the originating expression of the continued fraction object, the last convergent of the continued fraction, a list of its continuents (that is a list of pairs of partial numerators and denominators).

The operator `CF_CONVERGENTS` returns a list of all the convergents of the expansion.

`cf_expression(`⟨*cf_object*⟩`)`
`cf_convergent(`⟨*cf_object*⟩`)`
`cf_continuents(`⟨*cf_object*⟩`)`
`cf_convergents(`⟨*cf_object*⟩`)`

EXAMPLES

```
2: cf(6/11);

  6     6          1
{----,----,---------------}
  11    11           1
            1 + ---------
                       1
                  1 + ---
                       5

3: a := cf(pi,1000);

          355             1
a := {pi,-----,3 + ----------------}
          113               1
                   7 + ----------
                                1
                        15 + ---
                                1
```

```
4: cf_convergents a;

    22   333   355
{3,----,-----,-----}
    7    106   113

5: cf_continuents a;

{3,7,15,1}

6: precision 20;

12

7: cf pi;

     21053343141
{pi,-------------,{3,7,15,1,292,1,1,1,2,1,3,1,14,2,1,1,2,2,2,2,1}}
     6701487259
```

The operator `CF_EULER` is used to generate a generalised continued fraction expansion of an algebraic expression using a formula due to Leonhard Euler (see [**?**])).

> `cf_euler(`⟨*func*⟩`,`⟨*var*⟩`)`
> `cf_euler(`⟨*func*⟩`,`⟨*var*⟩`,`⟨*length*⟩`)`

INPUT:	⟨*func*⟩	is a function
	⟨*var*⟩	is the function main variable
	⟨*length*⟩	is the maximum number of continuents to be generated and is **optional**.

The meaning of the parameters is similar to those of `CFRAC`, but the continued fraction expansion generated will usually be different. Note that unlike `CFRAC`, `CF_EULER` cannot currently generate continued fraction expansion of numbers and for a rational function argument (with a non-constant denominator) the expansion will not be exact.

A number of operators are provided for transforming their continued fraction argument ⟨*cf_object*⟩ into an equivalent expansion, that is one with exactly the same convergents. They all accept as their single argument any continued fraction object ⟨*cf_object*⟩. These are:

`cf_unit_denominators`	converts all partial denominators to 1.
`cf_unit_numerators`	converts all partial numerators to 1.
`cf_remove_fractions`	converts the denominators of the partial numerators and partial denominators in the continuents to 1.
`cf_remove_constant`	removes the zeroth continuent (if non-zero) absorbing it into the first continuent pair.

The operator `CF_TRANSFORM` is a general purpose function for transforming its continued fraction argument ⟨*cf_object*⟩ into an equivalent expansion. Unlike the four preceding operators it requires a second argument: a list of multipliers used to modify the partial numerators and denominators of the original expansion.

`cf_transform(`⟨*cf_object*⟩`,`⟨*multiplier-list*⟩`)`

To understand the operation of `cf_transform` consider first the special case where ⟨*multiplier-list*⟩ is a list of the form $\{1, 1, \ldots, 1, l_n, 1, \ldots, 1\}$ whose nth element is l_n. Only the nth continuent pair $\{a_n,\ b_n\}$ and (n+1)th partial numerator a_{n+1} are altered and become $\{l_n a_n,\ l_n b_n\}$ and $l_n a_{n+1}$ respectively. For a ⟨*multiplier-list*⟩ that has more than one non-unit element, the above transformations are applied sequentially from left to right.

If the number of continuent pairs in the ⟨*cf_object*⟩ is greater than the length of the ⟨*multiplier-list*⟩, the latter is (in effect) padded with 1's. Conversely if it is shorter, the surplus elements of ⟨*multiplier-list*⟩ are ignored.

The operator `CF_EVEN_ODD` splits its continued fraction argument ⟨*cf_object*⟩ into two continued fraction objects: namely its even and odd parts (in that order) which are returned as a two-element list.

`cf_even_odd(`⟨*cf_object*⟩`)`

The convergents of the even part are the even-numbered convergents of the original expansion and those of the odd part are the odd-numbered ones (except the zeroth convergent which is necessarily zero). For the continued fraction expansions generated by the operators `cf` and `cfrac` with a numerical first argument ⟨*num*⟩. The convergents of the even part form a monotonically increasing sequence whilst those of the odd part (after the zeroth) form a monotonically decreasing sequence.

EXAMPLES

```
cf_remove_fractions(cf_euler(e^x, x, 4));
```

```
        3      2
  x    x  + 3*x  + 6*x + 6
{e , --------------------,
              6

                   1
 --------------------------------------}
                     x
  1 - --------------------------------
                          x
       (x + 1) - --------------------
                              2*x
                  (x + 2) - -------
                              x + 3
```

```
a := cf_remove_fractions(cf_euler(4*atan x, x, 4));
```

```
a := {4*atan(x),

               7       5        3
        - 60*x  + 84*x  - 140*x  + 420*x
      ------------------------------------,
                      105

                                 4*x
      -------------------------------------------------------}
                                   2
                                  x
       1 + ------------------------------------------------
                                           2
                 2                       9*x
            ( - x  + 3) + --------------------------------
                                                    2
                                   2            25*x
                            ( - 3*x  + 5) + -------------
                                                   2
                                              - 5*x  + 7
```

```
b := (a where x => 1);

          304            4
```

```
b := {pi,-----,---------------------}
          105               1
                1 + ----------------
                              9
                     2 + ----------
                                25
                          2 + ----
                                2

c := cf(pi, 0, 6);

          104348                   1
c := {pi,--------,3 + -------------------------------}
          33215                       1
                       7 + ------------------------
                                         1
                            15 + -----------------
                                            1
                                  1 + -----------
                                               1
                                      292 + ---
                                               1

cf_remove_constant c;

     104348                 22
{pi,--------,--------------------------------}
     33215                     1
              7 + -------------------------
                                22
                   333 + -----------------
                                     1
                          1 + -----------
                                        1
                              292 + ---
                                        1

c:= cf(pi, 0, 8)$
d := cf_even_odd c;

           208341                15
d := {{pi,--------,3 + ----------------------},
```

```
     66317                      292
                106 - --------------
                                 15
                       4687 - -----
                                 585

     312689              22
{pi,--------,-------------------------}}
     99532                  1
            7 + -------------------
                             22
                 355 - -----------
                                 1
                        294 - ---
                                 3
```

```
cf_convergents c;

    22   333   355   103993   104348   208341   312689
{3,----,-----,-----,--------,--------,--------,--------}
    7    106   113   33102    33215    66317    99532

cf_convergents first d;

    333   103993   208341
{3,-----,--------,--------}
    106   33102    66317

cf_convergents second d;

    22   355   104348   312689
{0,----,-----,--------,--------}
    7    113   33215    99532
```

3 Padé Approximation

The Padé approximant represents a function by the ratio of two polynomials (see [**?**] §4.2). The coefficients of the powers occuring in the polynomials are determined by the coefficients in the Taylor series expansion of the function (see [**?**]). Given a power series

$$f(x) = c_0 + c_1(x - h) + c_2(x - h)^2 \dots$$

and the degree of numerator, n, and of the denominator, d, the `pade` function finds the unique coefficients $a_i,\ b_i$ in the Padé approximant

$$\frac{a_0 + a_1 x + \cdots + a_n x^n}{b_0 + b_1 x + \cdots + b_d x^d}\ .$$

SYNTAX: `pade(f, x, h, n, d);`

INPUT:

`f`	the funtion to be approximated
`x`	the function variable
`h`	the point at which the approximation is evaluated
`n`	the (specified) degree of the numerator
`d`	the (specified) degree of the denominator

RESULT: Padé Approximant, ie. a rational function.

ERROR MESSAGES:

`***** not yet implemented`

The Taylor series expansion for the function, f, has not yet been implemented in the REDUCE Taylor Package.

`***** no Pade Approximation exists`

A Padé Approximant of this function does not exist.

`***** Pade Approximation of this order does not exist`

A Padé Approximant of this order (ie. the specified numerator and denominator orders) does not exist but one of a different order may exist.

EXAMPLES

```
23: pade(sin(x),x,0,3,3);


          2
 x*( - 7*x  + 60)
------------------
       2
   3*(x  + 20)

24: pade(tanh(x),x,0,5,5);
```

```
     4        2
 x*(x  + 105*x  + 945)
-----------------------
      4       2
 15*(x  + 28*x  + 63)

25: pade(atan(x),x,0,5,5);

        4        2
 x*(64*x  + 735*x  + 945)
--------------------------
         4       2
 15*(15*x  + 70*x  + 63)

26: pade(exp(1/x),x,0,5,5);

***** no Pade Approximation exists

27: pade(factorial(x),x,1,3,3);

***** not yet implemented

28: pade(asech(x),x,0,3,3);

           2                        2                  2
- 3*log(x)*x  + 8*log(x) + 3*log(2)*x  - 8*log(2) + 2*x
--------------------------------------------------------
                              2
                          3*x  - 8

29: taylor(ws-asech(x),x,0,10);

              11
log(x)*(0 + O(x  ))

     13    6      43    8     1611    10       11
 + (-----*x  + ------*x  + -------*x   + O(x  ))
     768          2048        81920

30: pade(sin(x)/x^2,x,0,10,0);
```

```
***** Pade Approximation of this order does not exist

31:  pade(sin(x)/x^2,x,0,10,2);

     10         8          6            4             2
( - x   + 110*x  - 7920*x  + 332640*x  - 6652800*x

  + 39916800)/(39916800*x)

32: pade(exp(x),x,0,10,10);



  10         9          8            7             6
(x   + 110*x  + 5940*x  + 205920*x  + 5045040*x

              5                4                 3
  + 90810720*x  + 1210809600*x  + 11762150400*x

                 2
  + 79394515200*x  + 335221286400*x + 670442572800)/

     10         9          8            7             6
    (x   - 110*x  + 5940*x  - 205920*x  + 5045040*x

                 5                4
         - 90810720*x  + 1210809600*x

                    3              2
         - 11762150400*x + 79394515200*x

         - 335221286400*x + 670442572800)

33: pade(sin(sqrt(x)),x,0,3,3);

(sqrt(x)*
            3            2
  (56447*x  - 4851504*x  + 132113520*x - 885487680))\

              3         2
    (7*(179*x  - 7200*x  - 2209680*x - 126498240))
```

References

[1] Jones, W B.; Thron, W.J.,
Continued fractions. Analytic Theory and Applications, (Encyclopedia of Mathematics and its Applications, Vol 11), Addison-Wesley Publishing Company, Reading, Massachusetts, 1980.

[2] L Euler L., *Introductio in analysin infinitorum, Vol 1, Chapter 18*, 1748.

[3] Baker(Jr.), G.A. and Graves-Morris, P.,
Padé Approximants, Part I: Basic Theory, (Encyclopedia of Mathematics and its Applications, Vol 13), Addison-Wesley Publishing Company, Reading, Massachusetts, 1981.