
Sage Reference Manual: Diophantine approximation

Release 6.8

The Sage Development Team

July 29, 2015

CONTENTS

1	Continued fractions	3
2	Indices and Tables	25

The diophantine approximation deals with the approximation of real numbers (or real vectors) with rational numbers (or rational vectors). See the article [Wikipedia article Diophantine_approximation](#) for more information.

CONTINUED FRACTIONS

A continued fraction is a representation of a real number in terms of a sequence of integers denoted $[a_0; a_1, a_2, \dots]$. The well known decimal expansion is another way of representing a real number by a sequence of integers. The value of a continued fraction is defined recursively as:

$$[a_0; a_1, a_2, \dots] = a_0 + \frac{1}{[a_1; a_2, \dots]} = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{\dots}}}$$

In this expansion, all coefficients a_n are integers and only the value a_0 may be non positive. Note that a_0 is nothing else but the floor (this remark provides a way to build the continued fraction expansion from a given real number). As examples

$$\frac{45}{38} = 1 + \frac{1}{5 + \frac{1}{2 + \frac{1}{3}}}$$

$$\pi = 3 + \frac{1}{7 + \frac{1}{15 + \frac{1}{1 + \frac{1}{292 + \frac{1}{\dots}}}}}$$

It is quite remarkable that

- any real number admits a unique continued fraction expansion
- finite expansions correspond to rationals
- ultimately periodic expansions correspond to quadratic numbers (ie numbers of the form $a + b\sqrt{D}$ with a and b rationals and D square free positive integer)
- two real numbers x and y have the same tail (up to a shift) in their continued fraction expansion if and only if there are integers a, b, c, d with $|ad - bc| = 1$ and such that $y = (ax + b)/(cx + d)$.

Moreover, the rational numbers obtained by truncation of the expansion of a real number gives its so-called best approximations. For more informations on continued fractions, you may have a look at [Wikipedia article Continued fraction](#).

EXAMPLES:

If you want to create the continued fraction of some real number you may either use its method `continued_fraction` (if it exists) or call `continued_fraction()`:

```
sage: (13/27).continued_fraction()
[0; 2, 13]
sage: 0 + 1/(2 + 1/13)
13/27

sage: continued_fraction(22/45)
[0; 2, 22]
sage: 0 + 1/(2 + 1/22)
22/45

sage: continued_fraction(pi)
[3; 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, ...]
sage: continued_fraction_list(pi, nterms=5)
[3, 7, 15, 1, 292]

sage: K.<cbirt5> = NumberField(x^3 - 5, embedding=1.709)
sage: continued_fraction(cbirt5)
[1; 1, 2, 2, 4, 3, 3, 1, 5, 1, 1, 4, 10, 17, 1, 14, 1, 1, 3052, 1, ...]
```

It is also possible to create a continued fraction from a list of partial quotients:

```
sage: continued_fraction([-3,1,2,3,4,1,2])
[-3; 1, 2, 3, 4, 1, 2]
```

Even infinite:

```
sage: w = words.ThueMorseWord([1,2])
sage: w
word: 1221211221121221211212211221211221121221...
sage: continued_fraction(w)
[1; 2, 2, 1, 2, 1, 1, 2, 2, 1...]
```

To go back and forth between the value (as a real number) and the partial quotients (seen as a finite or infinite list) you can use the methods `quotients` and `value`:

```
sage: cf = (13/27).continued_fraction()
sage: cf.quotients()
[0, 2, 13]
sage: cf.value()
13/27
```

```
sage: cf = continued_fraction(pi)
sage: cf.quotients()
lazy list [3, 7, 15, ...]
sage: cf.value()
pi
```

```
sage: w = words.FibonacciWord([1,2])
sage: cf = continued_fraction(w)
sage: cf.quotients()
word: 12112121121121121121121121121121121121121...
sage: v = cf.value()
sage: v
1.387954587967143?
sage: v.n(digits=100)
1.38795458796714233691931385987318547787815245249853227189491728982641857762264893216988523703424296...
sage: v.continued_fraction()
[1; 2, 1, 1, 2, 1, 2, 1, 1, 2, 1, 1, 2, 1, 2, 1, 1, 2, 1, 2...]
```


Recall that quadratic numbers correspond to ultimately periodic continued fractions. For them special methods give access to preperiod and period:

```
sage: K.<sqrt2> = QuadraticField(2)
sage: cf = continued_fraction(sqrt2); cf
[1; (2)*]
sage: cf.value()
sqrt2
sage: cf.preperiod()
(1,)
sage: cf.period()
(2,)

sage: cf = (3*sqrt2 + 1/2).continued_fraction(); cf
[4; (1, 2, 1, 7)*]

sage: cf = continued_fraction([(1,2,3),(1,4)]); cf
[1; 2, 3, (1, 4)*]
sage: cf.value()
-2/23*sqrt2 + 36/23
```

On the following we can remark how the tail may change even in the same quadratic field:

```
sage: for i in xrange(20): print continued_fraction(i*sqrt2)
[0]
[1; (2)*]
[2; (1, 4)*]
[4; (4, 8)*]
[5; (1, 1, 1, 10)*]
[7; (14)*]
...
[24; (24, 48)*]
[25; (2, 5, 6, 5, 2, 50)*]
[26; (1, 6, 1, 2, 3, 2, 26, 2, 3, 2, 1, 6, 1, 52)*]
```

Nevertheless, the tail is preserved under invertible integer homographies:

```
sage: apply_homography = lambda m,z: (m[0,0]*z + m[0,1]) / (m[1,0]*z + m[1,1])
sage: m1 = SL2Z([60,13,83,18])
sage: m2 = SL2Z([27,80,28,83])
sage: a = sqrt2/3
sage: a.continued_fraction()
[0; 2, (8, 4)*]
sage: b = apply_homography(m1, a)
sage: b.continued_fraction()
[0; 1, 2, 1, 1, 1, 1, 6, (8, 4)*]
sage: c = apply_homography(m2, a)
sage: c.continued_fraction()
[0; 1, 26, 1, 2, 2, (8, 4)*]
sage: d = apply_homography(m1**2*m2**3, a)
sage: d.continued_fraction()
[0; 1, 2, 1, 1, 1, 1, 5, 2, 1, 1, 1, 1, 5, 26, 1, 2, 1, 26, 1, 2, 1, 26, 1, 2, 2, (8, 4)*]
```

Todo

- Gosper's algorithm to compute the continued fraction of $(ax + b)/(cx + d)$ knowing the one of x (see Gosper (1972, <http://www.inwap.com/pdp10/hbaker/hakmem/cf.html>), Knuth (1998, TAOCP vol 2, Exercise 4.5.3.15), Fowler (1999). See also Liardet, P. and Stambul, P. "Algebraic Computation with Continued Fractions." J.

Number Th. 73, 92-121, 1998.

- Improve numerical approximation (the method `_mpfr_()` is quite slow compared to the same method for an element of a number field)
 - Make a class for generalized continued fractions of the form $a_0 + b_0/(a_1 + b_1/(...))$ (the standard continued fractions are when all $b_n = 1$ while the Hirzebruch-Jung continued fractions are the one for which $b_n = -1$ for all n). See [Wikipedia article Generalized_continued_fraction](#).
 - look at the function `ContinuedFractionApproximationOfRoot` in GAP
-

AUTHORS:

- Vincent Delecroix (2014): cleaning, refactorisation, documentation from the old implementation in `confrac` ([trac ticket #14567](#)).

class `sage.rings.continued_fraction.ContinuedFraction_base`

Bases: `sage.structure.sage_object.SageObject`

Base class for (standard) continued fractions.

If you want to implement your own continued fraction, simply derived from this class and implement the following methods:

- `def quotient(self, n):` return the n -th quotient of `self` as a Sage integer
- `def length(self):` the number of partial quotients of `self` as a Sage integer or Infinity.

and optionally:

- `def value(self):` return the value of `self` (an exact real number)

This base class will provide:

- computation of convergents in `convergent()`, `numerator()` and `denominator()`
- comparison with other continued fractions (see `__cmp__()`)
- elementary arithmetic function `floor()`, `ceil()`, `sign()`
- accurate numerical approximations `_mpfr_()`

All other methods, in particular the ones involving binary operations like sum or product, rely on the optional method `value()` (and not on convergents) and may fail at execution if it is not implemented.

additive_order()

Return the additive order of this continued fraction, which we defined to be the additive order of its value.

EXAMPLES:

```
sage: continued_fraction(-1).additive_order()
+Infinity
sage: continued_fraction(0).additive_order()
1
```

ceil()

Return the ceil of `self`.

EXAMPLES:

```
sage: cf = continued_fraction([2,1,3,4])
sage: cf.ceil()
3
```

convergent (*n*)

Return the *n*-th partial convergent to self.

EXAMPLES:

```
sage: a = continued_fraction(pi); a
[3; 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, ...]
sage: a.convergent(3)
355/113
sage: a.convergent(15)
411557987/131002976
```

convergents ()

Return the list of partial convergents of self.

If self is an infinite continued fraction, then the object returned is a `lazy_list` which behave like an infinite list.

EXAMPLES:

```
sage: a = continued_fraction(23/157); a
[0; 6, 1, 4, 1, 3]
sage: a.convergents()
[0, 1/6, 1/7, 5/34, 6/41, 23/157]

sage: #TODO: example with infinite list
```

denominator (*n*)

Return the denominator of the *n*-th partial convergent of self.

EXAMPLES:

```
sage: c = continued_fraction(pi); c
[3; 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, ...]
sage: c.denominator(0)
1
sage: c.denominator(12)
25510582
sage: c.denominator(152)
1255341492699841451528811722575401081588363886480089431843026103930863337221076748
```

floor ()

Return the floor of self.

EXAMPLES:

```
sage: cf = continued_fraction([2,1,2,3])
sage: cf.floor()
2
```

is_minus_one ()

Test whether self is minus one.

EXAMPLES:

```
sage: continued_fraction(-1).is_minus_one()
True
sage: continued_fraction(1).is_minus_one()
False
sage: continued_fraction(0).is_minus_one()
False
sage: continued_fraction(-2).is_minus_one()
False
```

```
sage: continued_fraction([-1,1]).is_minus_one()
False
```

is_one()

Test whether self is one.

EXAMPLES:

```
sage: continued_fraction(1).is_one()
True
sage: continued_fraction(5/4).is_one()
False
sage: continued_fraction(0).is_one()
False
sage: continued_fraction(pi).is_one()
False
```

is_zero()

Test whether self is zero.

EXAMPLES:

```
sage: continued_fraction(0).is_zero()
True
sage: continued_fraction((0,1)).is_zero()
False
sage: continued_fraction(-1/2).is_zero()
False
sage: continued_fraction(pi).is_zero()
False
```

multiplicative_order()

Return the multiplicative order of this continued fraction, which we defined to be the multiplicative order of its value.

EXAMPLES:

```
sage: continued_fraction(-1).multiplicative_order()
2
sage: continued_fraction(1).multiplicative_order()
1
sage: continued_fraction(pi).multiplicative_order()
+Infinity
```

n (prec=None, digits=None, algorithm=None)

Return a numerical approximation of this continued fraction.

INPUT:

- prec - the precision
- digits - the number of digits
- algorithm - the algorithm to use

See `sage.misc.functional.numerical_approx()` for more information on the input.

EXAMPLES:

```
sage: w = words.FibonacciWord([1,3])
sage: cf = continued_fraction(w)
sage: cf
[1; 3, 1, 1, 3, 1, 3, 1, 1, 3, 1, 1, 3, 1, 1, 3, 1, 3, 1, 1, 3, 1, 3...]
```

```
sage: cf.numerical_approx(prec=53)
1.28102513329557
```

The method n is a shortcut to this one:

```
sage: cf.n(digits=25)
1.281025133295569815552930
sage: cf.n(digits=33)
1.28102513329556981555293038097590
```

numerator(n)

Return the numerator of the n -th partial convergent of self.

EXAMPLES:

```
sage: c = continued_fraction(pi); c
[3; 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, ...]
sage: c.numerator(0)
3
sage: c.numerator(12)
80143857
sage: c.numerator(152)
3943771611212266962743738812600748213157266596588744951727393497446921245353005283
```

numerical_approx($prec=None$, $digits=None$, $algorithm=None$)

Return a numerical approximation of this continued fraction.

INPUT:

- $prec$ - the precision
- $digits$ - the number of digits
- $algorithm$ - the algorithm to use

See `sage.misc.functional.numerical_approx()` for more information on the input.

EXAMPLES:

```
sage: w = words.FibonacciWord([1,3])
sage: cf = continued_fraction(w)
sage: cf
[1; 3, 1, 1, 3, 1, 3, 1, 1, 3, 1, 1, 3, 1, 3, 1, 1, 3, 1, 3...]
sage: cf.numerical_approx(prec=53)
1.28102513329557
```

The method n is a shortcut to this one:

```
sage: cf.n(digits=25)
1.281025133295569815552930
sage: cf.n(digits=33)
1.28102513329556981555293038097590
```

p(n)

Return the numerator of the n -th partial convergent of self.

EXAMPLES:

```
sage: c = continued_fraction(pi); c
[3; 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, ...]
sage: c.numerator(0)
3
sage: c.numerator(12)
80143857
```

```
80143857
sage: c.numerator(152)
3943771611212266962743738812600748213157266596588744951727393497446921245353005283
```

pn(*n*)

Return the numerator of the *n*-th partial convergent of *self*.

This method is deprecated since [trac ticket #14567](#) and `numerator()` should be used instead.

EXAMPLES:

```
sage: continued_fraction([1,2,3,5,4]).pn(3)
doctest:...: DeprecationWarning: pn is deprecated. Use the methods p or numerator instead.
See http://trac.sagemath.org/14567 for details.
53
```

q(*n*)

Return the denominator of the *n*-th partial convergent of *self*.

EXAMPLES:

```
sage: c = continued_fraction(pi); c
[3; 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, ...]
sage: c.denominator(0)
1
sage: c.denominator(12)
25510582
sage: c.denominator(152)
1255341492699841451528811722575401081588363886480089431843026103930863337221076748
```

qn(*n*)

Return the denominator of the *n*-th partial convergent of *self*.

This method is deprecated since [trac ticket #14567](#). Use `denominator()` instead.

EXAMPLES:

```
sage: continued_fraction([1,2,3,12,1]).qn(3)
doctest:...: DeprecationWarning: qn is deprecated. Use the methods q or denominator instead.
See http://trac.sagemath.org/14567 for details.
93
```

quotients()

Return the list of partial quotients of *self*.

If *self* is an infinite continued fraction, the object returned is a `:class:~sage.misc.lazy_list.lazy_list` which behave like an infinite list.

EXAMPLES:

```
sage: a = continued_fraction(23/157); a
[0; 6, 1, 4, 1, 3]
sage: a.quotients()
[0, 6, 1, 4, 1, 3]

sage: #TODO: example with infinite list
```

sign()

Returns the sign of *self* as an Integer.

The sign is defined to be 0 if *self* is 0, 1 if *self* is positive and -1 if *self* is negative.

EXAMPLES:

```
sage: continued_fraction(tan(pi/7)).sign()
1
sage: continued_fraction(-34/2115).sign()
-1
sage: continued_fraction([0]).sign()
0
```

str (*nterms=10, unicode=False, join=True*)

Return a string representing this continued fraction.

INPUT:

- *nterms* – the maximum number of terms to use
- *unicode* – (default `False`) whether to use unicode character
- *join* – (default `True`) if `False` instead of returning a string return a list of string, each of them representing a line

EXAMPLES:

```
sage: print continued_fraction(pi).str()
1
3 + -----
      1
7 + -----
      1
15 + -----
      1
1 + -----
      1
292 + -----
      1
1 + -----
      1
1 + -----
      1
1 + -----
      1
1 + -----
      1
2 + -----
      1
1 + ...
```

```
sage: print continued_fraction(pi).str(nterms=1)
```

```
3 + ...
```

```
sage: print continued_fraction(pi).str(nterms=2)
```

```
1
3 + -----
      1
7 + ...
```

```
sage: print continued_fraction(243/354).str()
```

```
1
-----
1
1 + -----
      1
2 + -----
      1
5 + -----
      1
3 + ---
```

```

                2
sage: continued_fraction(243/354).str(join=False)
['          1          ',
 '-----',
 '          1          ',
 '1 + -----',
 '          1          ',
 '          2 + -----',
 '          1          ',
 '          5 + -----',
 '          1          ',
 '          3 + ----',
 '          2 ' ]

sage: print continued_fraction(243/354).str(unicode=True)
          1
-----
          1
1 + -----
          1
          2 + -----
          1
          5 + -----
          1
          3 + ----
          2

```

```

class sage.rings.continued_fraction.ContinuedFraction_infinite(w, value=None,
                                                                check=True)
Bases: sage.rings.continued_fraction.ContinuedFraction_base

```

A continued fraction defined by an infinite sequence of partial quotients.

EXAMPLES:

```

sage: t = continued_fraction(words.ThueMorseWord([1,2])); t
[1; 2, 2, 1, 2, 1, 1, 2, 2, 1...]
sage: t.n(digits=100)
1.4223887368827854883415471160245658253068791089917118293118924529164567472725658833124554129620

```

We check that comparisons work well:

```

sage: t > continued_fraction(1) and t < continued_fraction(3/2)
True
sage: t < continued_fraction(1) or t > continued_fraction(2)
False

```

Can also be called with a value option:

```

sage: def f(n):
....:     if n % 3 == 2: return 2*(n+1)//3
....:     return 1
sage: w = Word(f, alphabet=NN)
sage: w
word: 1,1,2,1,1,4,1,1,6,1,1,8,1,1,10,1,1,12,1,1,14,1,1,16,1,1,18,1,1,20,1,1,22,1,1,24,1,1,26,1,...
sage: cf = continued_fraction(w, value=e-1)
sage: cf
[1; 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 1, 1, 12, 1, 1...]

```

In that case a small check is done on the input:


```

sage: cf = continued_fraction(w, value=pi)
Traceback (most recent call last):
...
ValueError: value evaluates to 3.141592653589794? while the continued fraction evaluates to 1.71

```

length()

Returns infinity.

EXAMPLES:

```

sage: w = words.FibonacciWord([3,13])
sage: cf = continued_fraction(w)
sage: cf.length()
+Infinity

```

quotient(n)

The n-th partial quotient of self.

EXAMPLES:

```

sage: w = words.FibonacciWord([1,3])
sage: cf = continued_fraction(w)
sage: cf.quotient(0)
1
sage: cf.quotient(1)
3
sage: cf.quotient(2)
1

```

quotients()

Return the infinite list from which this continued fraction was built.

EXAMPLES:

```

sage: w = words.FibonacciWord([1,5])
sage: cf = continued_fraction(w)
sage: cf.quotients()
word: 15115151151151151151151151151151151151151...

```

value()

The value of self.

If this value was provided on initialization, just return this value otherwise return an element of the real lazy field.

EXAMPLES:

```

sage: def f(n):
....:     if n % 3 == 2: return 2*(n+1)//3
....:     return 1
sage: w = Word(f, alphabet=NN)
sage: w
word: 1,1,2,1,1,4,1,1,6,1,1,8,1,1,10,1,1,12,1,1,14,1,1,16,1,1,18,1,1,20,1,1,22,1,1,24,1,1,26,1,1,28,1,1,30,1,1,32,1,1,34,1,1,36,1,1,38,1,1,40,1,1,42,1,1,44,1,1,46,1,1,48,1,1,50,1,1,52,1,1,54,1,1,56,1,1,58,1,1,60,1,1,62,1,1,64,1,1,66,1,1,68,1,1,70,1,1,72,1,1,74,1,1,76,1,1,78,1,1,80,1,1,82,1,1,84,1,1,86,1,1,88,1,1,90,1,1,92,1,1,94,1,1,96,1,1,98,1,1,100,1,1,102,1,1,104,1,1,106,1,1,108,1,1,110,1,1,112,1,1,114,1,1,116,1,1,118,1,1,120,1,1,122,1,1,124,1,1,126,1,1,128,1,1,130,1,1,132,1,1,134,1,1,136,1,1,138,1,1,140,1,1,142,1,1,144,1,1,146,1,1,148,1,1,150,1,1,152,1,1,154,1,1,156,1,1,158,1,1,160,1,1,162,1,1,164,1,1,166,1,1,168,1,1,170,1,1,172,1,1,174,1,1,176,1,1,178,1,1,180,1,1,182,1,1,184,1,1,186,1,1,188,1,1,190,1,1,192,1,1,194,1,1,196,1,1,198,1,1,200,1,1,202,1,1,204,1,1,206,1,1,208,1,1,210,1,1,212,1,1,214,1,1,216,1,1,218,1,1,220,1,1,222,1,1,224,1,1,226,1,1,228,1,1,230,1,1,232,1,1,234,1,1,236,1,1,238,1,1,240,1,1,242,1,1,244,1,1,246,1,1,248,1,1,250,1,1,252,1,1,254,1,1,256,1,1,258,1,1,260,1,1,262,1,1,264,1,1,266,1,1,268,1,1,270,1,1,272,1,1,274,1,1,276,1,1,278,1,1,280,1,1,282,1,1,284,1,1,286,1,1,288,1,1,290,1,1,292,1,1,294,1,1,296,1,1,298,1,1,300,1,1,302,1,1,304,1,1,306,1,1,308,1,1,310,1,1,312,1,1,314,1,1,316,1,1,318,1,1,320,1,1,322,1,1,324,1,1,326,1,1,328,1,1,330,1,1,332,1,1,334,1,1,336,1,1,338,1,1,340,1,1,342,1,1,344,1,1,346,1,1,348,1,1,350,1,1,352,1,1,354,1,1,356,1,1,358,1,1,360,1,1,362,1,1,364,1,1,366,1,1,368,1,1,370,1,1,372,1,1,374,1,1,376,1,1,378,1,1,380,1,1,382,1,1,384,1,1,386,1,1,388,1,1,390,1,1,392,1,1,394,1,1,396,1,1,398,1,1,400,1,1,402,1,1,404,1,1,406,1,1,408,1,1,410,1,1,412,1,1,414,1,1,416,1,1,418,1,1,420,1,1,422,1,1,424,1,1,426,1,1,428,1,1,430,1,1,432,1,1,434,1,1,436,1,1,438,1,1,440,1,1,442,1,1,444,1,1,446,1,1,448,1,1,450,1,1,452,1,1,454,1,1,456,1,1,458,1,1,460,1,1,462,1,1,464,1,1,466,1,1,468,1,1,470,1,1,472,1,1,474,1,1,476,1,1,478,1,1,480,1,1,482,1,1,484,1,1,486,1,1,488,1,1,490,1,1,492,1,1,494,1,1,496,1,1,498,1,1,500,1,1,502,1,1,504,1,1,506,1,1,508,1,1,510,1,1,512,1,1,514,1,1,516,1,1,518,1,1,520,1,1,522,1,1,524,1,1,526,1,1,528,1,1,530,1,1,532,1,1,534,1,1,536,1,1,538,1,1,540,1,1,542,1,1,544,1,1,546,1,1,548,1,1,550,1,1,552,1,1,554,1,1,556,1,1,558,1,1,560,1,1,562,1,1,564,1,1,566,1,1,568,1,1,570,1,1,572,1,1,574,1,1,576,1,1,578,1,1,580,1,1,582,1,1,584,1,1,586,1,1,588,1,1,590,1,1,592,1,1,594,1,1,596,1,1,598,1,1,600,1,1,602,1,1,604,1,1,606,1,1,608,1,1,610,1,1,612,1,1,614,1,1,616,1,1,618,1,1,620,1,1,622,1,1,624,1,1,626,1,1,628,1,1,630,1,1,632,1,1,634,1,1,636,1,1,638,1,1,640,1,1,642,1,1,644,1,1,646,1,1,648,1,1,650,1,1,652,1,1,654,1,1,656,1,1,658,1,1,660,1,1,662,1,1,664,1,1,666,1,1,668,1,1,670,1,1,672,1,1,674,1,1,676,1,1,678,1,1,680,1,1,682,1,1,684,1,1,686,1,1,688,1,1,690,1,1,692,1,1,694,1,1,696,1,1,698,1,1,700,1,1,702,1,1,704,1,1,706,1,1,708,1,1,710,1,1,712,1,1,714,1,1,716,1,1,718,1,1,720,1,1,722,1,1,724,1,1,726,1,1,728,1,1,730,1,1,732,1,1,734,1,1,736,1,1,738,1,1,740,1,1,742,1,1,744,1,1,746,1,1,748,1,1,750,1,1,752,1,1,754,1,1,756,1,1,758,1,1,760,1,1,762,1,1,764,1,1,766,1,1,768,1,1,770,1,1,772,1,1,774,1,1,776,1,1,778,1,1,780,1,1,782,1,1,784,1,1,786,1,1,788,1,1,790,1,1,792,1,1,794,1,1,796,1,1,798,1,1,800,1,1,802,1,1,804,1,1,806,1,1,808,1,1,810,1,1,812,1,1,814,1,1,816,1,1,818,1,1,820,1,1,822,1,1,824,1,1,826,1,1,828,1,1,830,1,1,832,1,1,834,1,1,836,1,1,838,1,1,840,1,1,842,1,1,844,1,1,846,1,1,848,1,1,850,1,1,852,1,1,854,1,1,856,1,1,858,1,1,860,1,1,862,1,1,864,1,1,866,1,1,868,1,1,870,1,1,872,1,1,874,1,1,876,1,1,878,1,1,880,1,1,882,1,1,884,1,1,886,1,1,888,1,1,890,1,1,892,1,1,894,1,1,896,1,1,898,1,1,900,1,1,902,1,1,904,1,1,906,1,1,908,1,1,910,1,1,912,1,1,914,1,1,916,1,1,918,1,1,920,1,1,922,1,1,924,1,1,926,1,1,928,1,1,930,1,1,932,1,1,934,1,1,936,1,1,938,1,1,940,1,1,942,1,1,944,1,1,946,1,1,948,1,1,950,1,1,952,1,1,954,1,1,956,1,1,958,1,1,960,1,1,962,1,1,964,1,1,966,1,1,968,1,1,970,1,1,972,1,1,974,1,1,976,1,1,978,1,1,980,1,1,982,1,1,984,1,1,986,1,1,988,1,1,990,1,1,992,1,1,994,1,1,996,1,1,998,1,1,1000,1,1,1002,1,1,1004,1,1,1006,1,1,1008,1,1,1010,1,1,1012,1,1,1014,1,1,1016,1,1,1018,1,1,1020,1,1,1022,1,1,1024,1,1,1026,1,1,1028,1,1,1030,1,1,1032,1,1,1034,1,1,1036,1,1,1038,1,1,1040,1,1,1042,1,1,1044,1,1,1046,1,1,1048,1,1,1050,1,1,1052,1,1,1054,1,1,1056,1,1,1058,1,1,1060,1,1,1062,1,1,1064,1,1,1066,1,1,1068,1,1,1070,1,1,1072,1,1,1074,1,1,1076,1,1,1078,1,1,1080,1,1,1082,1,1,1084,1,1,1086,1,1,1088,1,1,1090,1,1,1092,1,1,1094,1,1,1096,1,1,1098,1,1,1100,1,1,1102,1,1,1104,1,1,1106,1,1,1108,1,1,1110,1,1,1112,1,1,1114,1,1,1116,1,1,1118,1,1,1120,1,1,1122,1,1,1124,1,1,1126,1,1,1128,1,1,1130,1,1,1132,1,1,1134,1,1,1136,1,1,1138,1,1,1140,1,1,1142,1,1,1144,1,1,1146,1,1,1148,1,1,1150,1,1,1152,1,1,1154,1,1,1156,1,1,1158,1,1,1160,1,1,1162,1,1,1164,1,1,1166,1,1,1168,1,1,1170,1,1,1172,1,1,1174,1,1,1176,1,1,1178,1,1,1180,1,1,1182,1,1,1184,1,1,1186,1,1,1188,1,1,1190,1,1,1192,1,1,1194,1,1,1196,1,1,1198,1,1,1200,1,1,1202,1,1,1204,1,1,1206,1,1,1208,1,1,1210,1,1,1212,1,1,1214,1,1,1216,1,1,1218,1,1,1220,1,1,1222,1,1,1224,1,1,1226,1,1,1228,1,1,1230,1,1,1232,1,1,1234,1,1,1236,1,1,1238,1,1,1240,1,1,1242,1,1,1244,1,1,1246,1,1,1248,1,1,1250,1,1,1252,1,1,1254,1,1,1256,1,1,1258,1,1,1260,1,1,1262,1,1,1264,1,1,1266,1,1,1268,1,1,1270,1,1,1272,1,1,1274,1,1,1276,1,1,1278,1,1,1280,1,1,1282,1,1,1284,1,1,1286,1,1,1288,1,1,1290,1,1,1292,1,1,1294,1,1,1296,1,1,1298,1,1,1300,1,1,1302,1,1,1304,1,1,1306,1,1,1308,1,1,1310,1,1,1312,1,1,1314,1,1,1316,1,1,1318,1,1,1320,1,1,1322,1,1,1324,1,1,1326,1,1,1328,1,1,1330,1,1,1332,1,1,1334,1,1,1336,1,1,1338,1,1,1340,1,1,1342,1,1,1344,1,1,1346,1,1,1348,1,1,1350,1,1,1352,1,1,1354,1,1,1356,1,1,1358,1,1,1360,1,1,1362,1,1,1364,1,1,1366,1,1,1368,1,1,1370,1,1,1372,1,1,1374,1,1,1376,1,1,1378,1,1,1380,1,1,1382,1,1,1384,1,1,1386,1,1,1388,1,1,1390,1,1,1392,1,1,1394,1,1,1396,1,1,1398,1,1,1400,1,1,1402,1,1,1404,1,1,1406,1,1,1408,1,1,1410,1,1,1412,1,1,1414,1,1,1416,1,1,1418,1,1,1420,1,1,1422,1,1,1424,1,1,1426,1,1,1428,1,1,1430,1,1,1432,1,1,1434,1,1,1436,1,1,1438,1,1,1440,1,1,1442,1,1,1444,1,1,1446,1,1,1448,1,1,1450,1,1,1452,1,1,1454,1,1,1456,1,1,1458,1,1,1460,1,1,1462,1,1,1464,1,1,1466,1,1,1468,1,1,1470,1,1,1472,1,1,1474,1,1,1476,1,1,1478,1,1,1480,1,1,1482,1,1,1484,1,1,1486,1,1,1488,1,1,1490,1,1,1492,1,1,1494,1,1,1496,1,1,1498,1,1,1500,1,1,1502,1,1,1504,1,1,1506,1,1,1508,1,1,1510,1,1,1512,1,1,1514,1,1,1516,1,1,1518,1,1,1520,1,1,1522,1,1,1524,1,1,1526,1,1,1528,1,1,1530,1,1,1532,1,1,1534,1,1,1536,1,1,1538,1,1,1540,1,1,1542,1,1,1544,1,1,1546,1,1,1548,1,1,1550,1,1,1552,1,1,1554,1,1,1556,1,1,1558,1,1,1560,1,1,1562,1,1,1564,1,1,1566,1,1,1568,1,1,1570,1,1,1572,1,1,1574,1,1,1576,1,1,1578,1,1,1580,1,1,1582,1,1,1584,1,1,1586,1,1,1588,1,1,1590,1,1,1592,1,1,1594,1,1,1596,1,1,1598,1,1,1600,1,1,1602,1,1,1604,1,1,1606,1,1,1608,1,1,1610,1,1,1612,1,1,1614,1,1,1616,1,1,1618,1,1,1620,1,1,1622,1,1,1624,1,1,1626,1,1,1628,1,1,1630,1,1,1632,1,1,1634,1,1,1636,1,1,1638,1,1,1640,1,1,1642,1,1,1644,1,1,1646,1,1,1648,1,1,1650,1,1,1652,1,1,1654,1,1,1656,1,1,1658,1,1,1660,1,1,1662,1,1,1664,1,1,1666,1,1,1668,1,1,1670,1,1,1672,1,1,1674,1,1,1676,1,1,1678,1,1,1680,1,1,1682,1,1,1684,1,1,1686,1,1,1688,1,1,1690,1,1,1692,1,1,1694,1,1,1696,1,1,1698,1,1,1700,1,1,1702,1,1,1704,1,1,1706,1,1,1708,1,1,1710,1,1,1712,1,1,1714,1,1,1716,1,1,1718,1,1,1720,1,1,1722,1,1,1724,1,1,1726,1,1,1728,1,1,1730,1,1,1732,1,1,1734,1,1,1736,1,1,1738,1,1,1740,1,1,1742,1,1,1744,1,1,1746,1,1,1748,1,1,1750,1,1,1752,1,1,1754,1,1,1756,1,1,1758,1,1,1760,1,1,1762,1,1,1764,1,1,1766,1,1,1768,1,1,1770,1,1,1772,1,1,1774,1,1,1776,1,1,1778,1,1,1780,1,1,1782,1,1,1784,1,1,1786,1,1,1788,1,1,1790,1,1,1792,1,1,1794,1,1,1796,1,1,1798,1,1,1800,1,1,1802,1,1,1804,1,1,1806,1,1,1808,1,1,1810,1,1,1812,1,1,1814,1,1,1816,1,1,1818,1,1,1820,1,1,1822,1,1,1824,1,1,1826,1,1,1828,1,1,1830,1,1,1832,1,1,1834,1,1,1836,1,1,1838,1,1,1840,1,1,1842,1,1,1844,1,1,1846,1,1,1848,1,1,1850,1,1,1852,1,1,1854,1,1,1856,1,1,1858,1,1,1860,1,1,1862,1,1,1864,1,1,1866,1,1,1868,1,1,1870,1,1,1872,1,1,1874,1,1,1876,1,1,1878,1,1,1880,1,1,1882,1,1,1884,1,1,1886,1,1,1888,1,1,1890,1,1,1892,1,1,1894,1,1,1896,1,1,1898,1,1,1900,1,1,1902,1,1,1904,1,1,1906,1,1,1908,1,1,1910,1,1,1912,1,1,1914,1,1,1916,1,1,1918,1,1,1920,1,1,1922,1,1,1924,1,1,1926,1,1,1928,1,1,1930,1,1,1932,1,1,1934,1,1,1936,1,1,1938,1,1,1940,1,1,1942,1,1,1944,1,1,1946,1,1,1948,1,1,1950,1,1,1952,1,1,1954,1,1,1956,1,1,1958,1,1,1960,1,1,1962,1,1,1964,1,1,1966,1,1,1968,1,1,1970,1,1,1972,1,1,1974,1,1,1976,1,1,1978,1,1,1980,1,1,1982,1,1,1984,1,1,1986,1,1,1988,1,1,1990,1,1,1992,1,1,1994,1,1,1996,1,1,1998,1,1,2000,1,1,2002,1,1,2004,1,1,2006,1,1,2008,1,1,2010,1,1,2012,1,1,2014,1,1,2016,1,1,2018,1,1,2020,1,1,2022,1,1,2024,1,1,2026,1,1,2028,1,1,2030,1,1,2032,1,1,2034,1,1,2036,1,1,2038,1,1,2040,1,1,2042,1,1,2044,1,1,2046,1,1,2048,1,1,2050,1,1,2052,1,1,2054,1,1,2056,1,1,2058,1,1,2060,1,1,2062,1,1,2064,1,1,2066,1,1,2068,1,1,2070,1,1,2072,1,1,2074,1,1,2076,1,1,2078,1,1,2080,1,1,2082,1,1,2084,1,1,2086,1,1,2088,1,1,2090,1,1,2092,1,1,2094,1,1,2096,1,1,2098,1,1,2100,1,1,2102,1,1,2104,1,1,2106,1,1,2108,1,1,2110,1,1,2112,1,1,2114,1,1,2116,1,1,2118,1,1,2120,1,1,2122,1,1,2124,1,1,2126,1,1,2128,1,1,2130,1,1,2132,1,1,2134,1,1,2136,1,1,2138,1,1,2140,1,1,2142,1,1,2144,1,1,2146,1,1,2148,1,1,2150,1,1,2152,1,1,2154,1,1,2156,1,1,2158,1,1,2160,1,1,2162,1,1,2164,1,1,2166,1,1,2168,1,1,2170,1,1,2172,1,1,2174,1,1,2176,1,1,2178,1,1,2180,1,1,2182,1,1,2184,1,1,2186,1,1,2188,1,1,2190,1,1,2192,1,1,2194,1,1,2196,1,1,2198,1,1,2200,1,1,2202,1,1,2204,1,1,2206,1,1,2208,1,1,2210,1,1,2212,1,1,2214,1,1,2216,1,1,2218,1,1,2220,1,1,2222,1,1,2224,1,1,2226,1,1,2228,1,1,2230,1,1,2232,1,1,2234,1,1,2236,1,1,2238,1,1,2240,1,1,2242,1,1,2244,1,1,2246,1,1,2248,1,1,2250,1,1,2252,1,1,2254,1,1,2256,1,1,2258,1,1,2260,1,1,2262,1,1,2264,1,1,2266,1,1,2268,1,1,2270,1,1,2272,1,1,2274,1,1,2276,1,1,2278,1,1,2280,1,1,2282,1,1,2284,1,1,2286,1,1,2288,1,1,2290,1,1,2292,1,1,2294,1,1,2296,1,1,2298,1,1,2300,1,1,2302,1,1,2304,1,1,2306,1,1,2308,1,1,2310,1,1,2312,1,1,2314,1,1,2316,1,1,2318,1,1,2320,1,1,2322,1,1,2324,1,1,2326,1,1,2328,1,1,2330,1,1,2332,1,1,2334,1,1,2336,1,1,2338,1,1,2340,1,1,2342,1,1,2344,1,1,2346,1,1,2348,1,1,2350,1,1,2352,1,1,2354,1,1,2356,1,1,2358,1,1,2360,1,1,2362,1,1,2364,1,1,2366,1,1,2368,1,1,2370,1,1,2372,1,1,2374,1,1,2376,1,1,2378,1,1,2380,1,1,2382,1,1,2384,1,1,2386,1,1,2388,1,1,2390,1,1,2392,1,1,2394,1,1,2396,1,1,2398,1,1,2400,1,1,2402,1,1,2404,1,1,2406,1,1,2408,1,1,2410,1,1,2412,1,1,2414,1,1,2416,1,1,2418,1,1,2420,1,1,2422,1,1,2424,1,1,2426,1,1,2428,1,1,2430,1,1,2432,1,1,2434,1,1,2436,1,1,2438,1,1,2440,1,1,2442,1,1,2444,1,1,2446,1,1,2448,1,1,2450,1,1,2452,1,1,2454,1,1,2456,1,1,2458,1,1,2460,1,1,2462,1,1,2464,1,1,2466,1,1,2468,1,1,2470,1,1,2472,1,1,2474,1,1,2476,1,1,2478,1,1,2480,1,1,2482,1,1,2484,1,1,2486,1,1,2488,1,1,2490,1,1,2492,1,1,2494,1,1,2496,1,1,2498,1,1,2500,1,1,2502,1,1,2504,1,1,2506,1,1,2508,1,1,2510,1,1,2512,1,1,2514,1,1,2516,1,1,2518,1,1,2520,1,1,2522,1,1,2524,1,1,2526,1,1,2528,1,1,2530,1,1,2532,1,1,2534,1,1,2536,1,1,2538,1,1,2540,1,1,2542,1,1,2544,1,1,2546,1,1,2548,1,1,2550,1,1,2552,1,1,2554,1,1,2556,1,1,2558,1,1,2560,1,1,2562,1,1,2564,1,1,2566,1,1,2568,1,1,2570,1,1,2572,1,1,2574,1,1,2576,1,1,2578,1,1,2580,1,1,2582,1,1,2584,1,1,2586,1,1,2588,1,1,2590,1,1,2592,1,1,2594,1,1,2596,1,1,2598,1,1,2600,1,1,2602,1,1,2604,1,1,2606,1,1,2608,1,1,2610,1,1,2612,1,1,2614,1,1,2616,1,1,2618,1,1,2620,1,1,2622,1,1,2624,1,1,2626,1,1,2628,1,1,2630,1,1,2632,1,1,2634
```

```
sage: cf
[2; 5, 2, 2, 5, 2, 5, 2, 2, 5, 2, 2, 5, 2, 5, 2, 2, 5, 2, 5...]
sage: cf.value()
2.184951302409338?
```

```
class sage.rings.continued_fraction.ContinuedFraction_periodic(x1, x2=None,
                                                                check=True)
Bases: sage.rings.continued_fraction.ContinuedFraction_base
```

Continued fraction associated with rational or quadratic number.

A rational number has a finite continued fraction expansion (or ultimately 0). The one of a quadratic number, ie a number of the form $a + b\sqrt{D}$ with a and b rational, is ultimately periodic.

Note: This class stores a tuple `_x1` for the preperiod and a tuple `_x2` for the period. In the purely periodic case `_x1` is empty while in the rational case `_x2` is the tuple `(0,)`.

length()

Returns the number of partial quotients of `self`.

EXAMPLES:

```
sage: continued_fraction(2/5).length()
3
sage: cf = continued_fraction([(0,1),(2,)]); cf
[0; 1, (2)*]
sage: cf.length()
+Infinity
```

period()

Return the periodic part of `self`.

EXAMPLES:

```
sage: K.<sqrt3> = QuadraticField(3)
sage: cf = continued_fraction(sqrt3); cf
[1; (1, 2)*]
sage: cf.period()
(1, 2)

sage: for k in xrange(2,40):
....:     if not k.is_square():
....:         s = QuadraticField(k).gen()
....:         cf = continued_fraction(s)
....:         print '%2d %d %s'%(k, len(cf.period()), cf)
2 1 [1; (2)*]
3 2 [1; (1, 2)*]
5 1 [2; (4)*]
6 2 [2; (2, 4)*]
7 4 [2; (1, 1, 1, 4)*]
8 2 [2; (1, 4)*]
10 1 [3; (6)*]
11 2 [3; (3, 6)*]
12 2 [3; (2, 6)*]
13 5 [3; (1, 1, 1, 1, 6)*]
14 4 [3; (1, 2, 1, 6)*]
...
35 2 [5; (1, 10)*]
37 1 [6; (12)*]
```

```
38 2 [6; (6, 12)*]
39 2 [6; (4, 12)*]
```

preperiod()

Return the preperiodic part of self.

EXAMPLES:

```
sage: K.<sqrt3> = QuadraticField(3)
sage: cf = continued_fraction(sqrt3); cf
[1; (1, 2)*]
sage: cf.preperiod()
(1,)

sage: cf = continued_fraction(sqrt3/7); cf
[0; 4, (24, 8)*]
sage: cf.preperiod()
(0, 4)
```

quotient(n)

Return the n-th partial quotient of self.

EXAMPLES:

```
sage: cf = continued_fraction([(12,5), (1,3)])
sage: [cf.quotient(i) for i in xrange(10)]
[12, 5, 1, 3, 1, 3, 1, 3, 1, 3]
```

value()

Return the value of self as a quadratic number (with square free discriminant).

EXAMPLES:

Some purely periodic examples:

```
sage: cf = continued_fraction([( ), (2, )]); cf
[(2)*]
sage: v = cf.value(); v
sqrt2 + 1
sage: v.continued_fraction()
[(2)*]

sage: cf = continued_fraction([( ), (1, 2)]); cf
[(1, 2)*]
sage: v = cf.value(); v
1/2*sqrt3 + 1/2
sage: v.continued_fraction()
[(1, 2)*]
```

The number `sqrt3` that appear above is actually internal to the continued fraction. In order to be access it from the console:

```
sage: cf.value().parent().inject_variables()
Defining sqrt3
sage: sqrt3
sqrt3
sage: ((sqrt3+1)/2).continued_fraction()
[(1, 2)*]
```

Some ultimately periodic but non periodic examples:

```

sage: cf = continued_fraction([(1,),(2,)]); cf
[1; (2)*]
sage: v = cf.value(); v
sqrt2
sage: v.continued_fraction()
[1; (2)*]

sage: cf = continued_fraction([(1,3),(1,2)]); cf
[1; 3, (1, 2)*]
sage: v = cf.value(); v
-sqrt3 + 3
sage: v.continued_fraction()
[1; 3, (1, 2)*]

sage: cf = continued_fraction([(-5,18), (1,3,1,5)])
sage: cf.value().continued_fraction() == cf
True
sage: cf = continued_fraction([(-1,),(1,)])
sage: cf.value().continued_fraction() == cf
True

```

TESTS:

```

sage: a1 = ((0,1),(2,3))
sage: a2 = ((-12,1,1),(2,3,2,4))
sage: a3 = ((1,),(1,2))
sage: a4 = ((-2,2),(1,124,13))
sage: a5 = ((0,),(1,))
sage: for a in a1,a2,a3,a4,a5:
....:     cf = continued_fraction(a)
....:     assert cf.value().continued_fraction() == cf

```

class sage.rings.continued_fraction.**ContinuedFraction_real**(*x*)
 Bases: sage.rings.continued_fraction.ContinuedFraction_base

Continued fraction of a real (exact) number.

This class simply wraps a real number into an attribute (that can be accessed through the method `value()`). The number is assumed to be irrational.

EXAMPLES:

```

sage: cf = continued_fraction(pi)
sage: cf
[3; 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, ...]
sage: cf.value()
pi

sage: cf = continued_fraction(e)
sage: cf
[2; 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 1, 1, 12, 1, 1, ...]
sage: cf.value()
e

```

length()

Return infinity

EXAMPLES:

```
sage: continued_fraction(pi).length()
+Infinity
```

quotient(*n*)

Returns the *n*-th quotient of self.

EXAMPLES:

```
sage: cf = continued_fraction(pi)
sage: cf.quotient(27)
13
sage: cf.quotient(2552)
152
sage: cf.quotient(10000)    # long time
5
```

The algorithm is not efficient with element of the symbolic ring and, if possible, one can always prefer number fields elements. The reason is that, given a symbolic element *x*, there is no automatic way to evaluate in RIF an expression of the form $(a*x+b)/(c*x+d)$ where both the numerator and the denominator are extremely small:

```
sage: a1 = pi
sage: c1 = continued_fraction(a1)
sage: p0 = c1.numerator(12); q0 = c1.denominator(12)
sage: p1 = c1.numerator(13); q1 = c1.denominator(13)
sage: num = (q0*a1 - p0); num.n()
1.49011611938477e-8
sage: den = (q1*a1 - p1); den.n()
-2.98023223876953e-8
sage: a1 = -num/den
sage: RIF(a1)
[-infinity .. +infinity]
```

The same computation with an element of a number field instead of *pi* gives a very satisfactory answer:

```
sage: K.<a2> = NumberField(x^3 - 2, embedding=1.25)
sage: c2 = continued_fraction(a2)
sage: p0 = c2.numerator(111); q0 = c2.denominator(111)
sage: p1 = c2.numerator(112); q1 = c2.denominator(112)
sage: num = (q0*a2 - p0); num.n()
-4.56719261665907e46
sage: den = (q1*a2 - p1); den.n()
-3.65375409332726e47
sage: a2 = -num/den
sage: b2 = RIF(a2); b2
1.002685823312715?
sage: b2.absolute_diameter()
8.88178419700125e-16
```

The consequence is that the precision needed with *c1* grows when we compute larger and larger partial quotients:

```
sage: c1.quotient(100)
2
sage: c1._xa.parent()
Real Interval Field with 353 bits of precision
sage: c1.quotient(200)
3
sage: c1._xa.parent()
Real Interval Field with 753 bits of precision
```

```

sage: c1.quotient(300)
5
sage: c1._xa.parent()
Real Interval Field with 1053 bits of precision

sage: c2.quotient(200)
6
sage: c2._xa.parent()
Real Interval Field with 53 bits of precision
sage: c2.quotient(500)
1
sage: c2._xa.parent()
Real Interval Field with 53 bits of precision
sage: c2.quotient(1000)
1
sage: c2._xa.parent()
Real Interval Field with 53 bits of precision

```

value()

Return the value of self (the number from which it was built).

EXAMPLES:

```

sage: cf = continued_fraction(e)
sage: cf.value()
e

```

```

sage.rings.continued_fraction.Hirzebruch_Jung_continued_fraction_list(x,
                                                                    bits=None,
                                                                    nterms=None)

```

Return the Hirzebruch-Jung continued fraction of x as a list.

This function is deprecated since [trac ticket #14567](#). See `continued_fraction_list()` and the documentation therein.

INPUT:

- x – exact rational or something that can be numerically evaluated. The number to compute the continued fraction of.
- $bits$ – integer (default: the precision of x). the precision of the real interval field that is used internally. This is only used if x is not an exact fraction.
- $nterms$ – integer (default: None). The upper bound on the number of terms in the continued fraction expansion to return. A list of integers, the coefficients in the Hirzebruch-Jung continued fraction expansion of x .

EXAMPLES:

```

sage: Hirzebruch_Jung_continued_fraction_list(17/11)
doctest:...: DeprecationWarning: Hirzebruch_Jung_continued_fraction_list(x) is replaced by
continued_fraction_list(x,type="hj")
or for rationals
x.continued_fraction_list(type="hj")
See http://trac.sagemath.org/14567 for details.
[2, 3, 2, 2, 2, 2]

```

```

sage.rings.continued_fraction.check_and_reduce_pair(x1, x2=None)

```

There are often two ways to represent a given continued fraction. This function makes it canonical.

In the very special case of the number 0 we return the pair $((0), (0))$.

TESTS:

```

sage: from sage.rings.continued_fraction import check_and_reduce_pair
sage: check_and_reduce_pair([])
((0,), (+Infinity,))
sage: check_and_reduce_pair([-1,1])
((0,), (+Infinity,))
sage: check_and_reduce_pair([1,1,1])
((1, 2), (+Infinity,))
sage: check_and_reduce_pair([1,3],[2,3])
((1,), (3, 2))
sage: check_and_reduce_pair([1,2,3],[2,3,2,3,2,3])
((1,), (2, 3))
sage: check_and_reduce_pair([1,2],[])
((1, 2), (+Infinity,))

```

`sage.rings.continued_fraction.continued_fraction(x, value=None)`

Return the continued fraction of x .

INPUT:

- x – a number or a list of partial quotients (for finite development) or two list of partial quotients (preperiod and period for ultimately periodic development)

EXAMPLES:

A finite continued fraction may be initialized by a number or by its list of partial quotients:

```

sage: continued_fraction(12/571)
[0; 47, 1, 1, 2, 2]
sage: continued_fraction([3,2,1,4])
[3; 2, 1, 4]

```

It can be called with elements defined from symbolic values, in which case the partial quotients are evaluated in a lazy way:

```

sage: c = continued_fraction(golden_ratio); c
[1; 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...]
sage: c.convergent(12)
377/233
sage: fibonacci(14)/fibonacci(13)
377/233

sage: continued_fraction(pi)
[3; 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, ...]
sage: c = continued_fraction(pi); c
[3; 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, ...]
sage: a = c.convergent(3); a
355/113
sage: a.n()
3.14159292035398
sage: pi.n()
3.14159265358979

sage: continued_fraction(sqrt(2))
[1; 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, ...]
sage: continued_fraction(tan(1))
[1; 1, 1, 3, 1, 5, 1, 7, 1, 9, 1, 11, 1, 13, 1, 15, 1, 17, 1, 19, ...]
sage: continued_fraction(tanh(1))
[0; 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, ...]
sage: continued_fraction(e)

```

```
[2; 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 1, 1, 12, 1, 1, ...]
```

If you want to play with quadratic numbers (such as `golden_ratio` and `sqrt(2)` above), it is much more convenient to use number fields as follows since preperiods and periods are computed:

```
sage: K.<sqrt5> = NumberField(x^2-5, embedding=2.23)
sage: my_golden_ratio = (1 + sqrt5)/2
sage: cf = continued_fraction((1+sqrt5)/2); cf
[(1)*]
sage: cf.convergent(12)
377/233
sage: cf.period()
(1,)
sage: cf = continued_fraction(2/3+sqrt5/5); cf
[1; 8, (1, 3, 1, 1, 3, 9)*]
sage: cf.preperiod()
(1, 8)
sage: cf.period()
(1, 3, 1, 1, 3, 9)

sage: L.<sqrt2> = NumberField(x^2-2, embedding=1.41)
sage: cf = continued_fraction(sqrt2); cf
[1; (2)*]
sage: cf.period()
(2,)
sage: cf = continued_fraction(sqrt2/3); cf
[0; 2, (8, 4)*]
sage: cf.period()
(8, 4)
```

It is also possible to go the other way around, build a ultimately periodic continued fraction from its preperiod and its period and get its value back:

```
sage: cf = continued_fraction([(1,1), (2,8)]); cf
[1; 1, (2, 8)*]
sage: cf.value()
2/11*sqrt5 + 14/11
```

It is possible to deal with higher degree number fields but in that case the continued fraction expansion is known to be aperiodic:

```
sage: K.<a> = NumberField(x^3-2, embedding=1.25)
sage: cf = continued_fraction(a); cf
[1; 3, 1, 5, 1, 1, 4, 1, 1, 8, 1, 14, 1, 10, 2, 1, 4, 12, 2, 3, ...]
```

Note that initial rounding can result in incorrect trailing partial quotients:

```
sage: continued_fraction(RealField(39)(e))
[2; 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 2]
```

Note the value returned for floating point number is the continued fraction associated to the rational number you obtain with a conversion:

```
sage: for _ in xrange(10):
....:     x = RR.random_element()
....:     cff = continued_fraction(x)
....:     cfe = QQ(x).continued_fraction()
....:     assert cff == cfe, "%s %s %s"%(x,cff,cfe)
```



```
sage.rings.continued_fraction.continued_fraction_list(x, type='std', partial_convergents=False, bits=None, nterms=None)
```

Returns the (finite) continued fraction of x as a list.

The continued fraction expansion of x are the coefficients a_i in

$$x = a_0 + 1/(a_1 + 1/(...))$$

with a_0 integer and a_1, \dots positive integers. The Hirzebruch-Jung continued fraction is the one for which the $+$ signs are replaced with $-$ signs

$$x = a_0 - 1/(a_1 - 1/(...))$$

See also:

```
continued_fraction()
```

INPUT:

- x – exact rational or floating-point number. The number to compute the continued fraction of.
- `type` – either “std” (default) for standard continued fractions or “hj” for Hirzebruch-Jung ones.
- `partial_convergents` – boolean. Whether to return the partial convergents.
- `bits` – an optional integer that specify a precision for the real interval field that is used internally.
- `nterms` – integer. The upper bound on the number of terms in the continued fraction expansion to return.

OUTPUT:

A list of integers, the coefficients in the continued fraction expansion of x . If `partial_convergents` is set to `True`, then return a pair containing the coefficient list and the partial convergents list is returned.

EXAMPLES:

```
sage: continued_fraction_list(45/19)
[2, 2, 1, 2, 2]
sage: 2 + 1/(2 + 1/(1 + 1/(2 + 1/2)))
45/19

sage: continued_fraction_list(45/19, type="hj")
[3, 2, 3, 2, 3]
sage: 3 - 1/(2 - 1/(3 - 1/(2 - 1/3)))
45/19
```

Specifying `bits` or `nterms` modify the length of the output:

```
sage: continued_fraction_list(e, bits=20)
[2, 1, 2, 1, 1, 4, 2]
sage: continued_fraction_list(sqrt(2)+sqrt(3), bits=30)
[3, 6, 1, 5, 7, 2]
sage: continued_fraction_list(pi, bits=53)
[3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14]

sage: continued_fraction_list(log(3/2), nterms=15)
[0, 2, 2, 6, 1, 11, 2, 1, 2, 2, 1, 4, 3, 1, 1]
sage: continued_fraction_list(tan(sqrt(pi)), nterms=20)
[-5, 9, 4, 1, 1, 1, 1, 1, 5, 1, 1, 1, 1, 1, 2, 4, 3, 1, 63]
```

When the continued fraction is infinite (ie x is an irrational number) and the parameters `bits` and `nterms` are not specified then a warning is raised:

```
sage: continued_fraction_list(sqrt(2))
doctest:...: UserWarning: the continued fraction of sqrt(2) seems infinite, return only the first
[1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
sage: continued_fraction_list(sqrt(4/19))
doctest:...: UserWarning: the continued fraction of 2*sqrt(1/19) seems infinite, return only the first
[0, 2, 5, 1, 1, 2, 1, 16, 1, 2, 1, 1, 5, 4, 5, 1, 1, 2, 1, 16]
```

An examples with the list of partial convergents:

```
sage: continued_fraction_list(RR(pi), partial_convergents=True)
([3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 3],
 [(3, 1),
  (22, 7),
  (333, 106),
  (355, 113),
  (103993, 33102),
  (104348, 33215),
  (208341, 66317),
  (312689, 99532),
  (833719, 265381),
  (1146408, 364913),
  (4272943, 1360120),
  (5419351, 1725033),
  (80143857, 25510582),
  (245850922, 78256779)])
```

TESTS:

```
sage: continued_fraction_list(1 + 10^-10, nterms=3)
[1, 10000000000]
sage: continued_fraction_list(1 + 10^-20 - e^-100, nterms=3)
[1, 10000000000000000000000, 2688]
sage: continued_fraction_list(1 + 10^-20 - e^-100, nterms=5)
[1, 10000000000000000000000, 2688, 8, 1]
sage: continued_fraction_list(1 + 10^-20 - e^-100, nterms=5)
[1, 10000000000000000000000, 2688, 8, 1]
```

`sage.rings.continued_fraction.convergents(x)`

Return the (partial) convergents of the number x .

EXAMPLES:

```
sage: convergents(143/255)
[0, 1, 1/2, 4/7, 5/9, 9/16, 14/25, 23/41, 60/107, 143/255]
```

`sage.rings.continued_fraction.farey(v , lim)`

Return the Farey sequence associated to the floating point number v .

INPUT:

- v - float (automatically converted to a float)
- lim - maximum denominator.

OUTPUT: Results are (numerator, denominator); (1, 0) is “infinity”.

EXAMPLES:

```

sage: farey(2.0, 100)
doctest:...: DeprecationWarning: farey is deprecated.
See http://trac.sagemath.org/14567 for details.
(2, 1)
sage: farey(2.0, 1000)
(2, 1)
sage: farey(2.1, 1000)
(21, 10)
sage: farey(2.1, 100000)
(21, 10)
sage: farey(pi, 100000)
(312689, 99532)

```

AUTHORS:

•Scott David Daniels: Python Cookbook, 2nd Ed., Recipe 18.13

`sage.rings.continued_fraction.last_two_convergents(x)`

Given the list `x` that consists of numbers, return the two last convergents $p_{n-1}, q_{n-1}, p_n, q_n$.

This function is principally used to compute the value of a ultimately periodic continued fraction.

OUTPUT: a 4-tuple of Sage integers

EXAMPLES:

```

sage: from sage.rings.continued_fraction import last_two_convergents
sage: last_two_convergents([])
(0, 1, 1, 0)
sage: last_two_convergents([0])
(1, 0, 0, 1)
sage: last_two_convergents([-1, 1, 3, 2])
(-1, 4, -2, 9)

```

TESTS:

```

sage: all(type(x) is Integer for x in last_two_convergents([]))
True

```

`sage.rings.continued_fraction.rat_interval_cf_list(r1, r2)`

Return the common prefix of the rationals `r1` and `r2` seen as continued fractions.

OUTPUT: a list of Sage integers.

EXAMPLES:

```

sage: from sage.rings.continued_fraction import rat_interval_cf_list
sage: rat_interval_cf_list(257/113, 5224/2297)
[2, 3, 1, 1, 1, 4]
sage: for prec in xrange(10, 54):
....:     R = RealIntervalField(20)
....:     for _ in xrange(100):
....:         x = R.random_element() * R.random_element() + R.random_element() / 100
....:         l = x.lower().exact_rational()
....:         u = x.upper().exact_rational()
....:         cf = rat_interval_cf_list(l, u)
....:         a = continued_fraction(cf).value()
....:         b = continued_fraction(cf+[1]).value()
....:         if a > b:
....:             a, b = b, a
....:         assert a <= l
....:         assert b >= u

```


INDICES AND TABLES

- [Index](#)
- [Module Index](#)
- [Search Page](#)

r

`sage.rings.continued_fraction`, [3](#)

A

`additive_order()` (`sage.rings.continued_fraction.ContinuedFraction_base` method), 6

C

`ceil()` (`sage.rings.continued_fraction.ContinuedFraction_base` method), 6

`check_and_reduce_pair()` (in module `sage.rings.continued_fraction`), 18

`continued_fraction()` (in module `sage.rings.continued_fraction`), 19

`continued_fraction_list()` (in module `sage.rings.continued_fraction`), 20

`ContinuedFraction_base` (class in `sage.rings.continued_fraction`), 6

`ContinuedFraction_infinite` (class in `sage.rings.continued_fraction`), 12

`ContinuedFraction_periodic` (class in `sage.rings.continued_fraction`), 14

`ContinuedFraction_real` (class in `sage.rings.continued_fraction`), 16

`convergent()` (`sage.rings.continued_fraction.ContinuedFraction_base` method), 6

`convergents()` (in module `sage.rings.continued_fraction`), 22

`convergents()` (`sage.rings.continued_fraction.ContinuedFraction_base` method), 7

D

`denominator()` (`sage.rings.continued_fraction.ContinuedFraction_base` method), 7

F

`farey()` (in module `sage.rings.continued_fraction`), 22

`floor()` (`sage.rings.continued_fraction.ContinuedFraction_base` method), 7

H

`Hirzebruch_Jung_continued_fraction_list()` (in module `sage.rings.continued_fraction`), 18

I

`is_minus_one()` (`sage.rings.continued_fraction.ContinuedFraction_base` method), 7

`is_one()` (`sage.rings.continued_fraction.ContinuedFraction_base` method), 8

`is_zero()` (`sage.rings.continued_fraction.ContinuedFraction_base` method), 8

L

`last_two_convergents()` (in module `sage.rings.continued_fraction`), 23

`length()` (`sage.rings.continued_fraction.ContinuedFraction_infinite` method), 13

`length()` (`sage.rings.continued_fraction.ContinuedFraction_periodic` method), 14

`length()` (`sage.rings.continued_fraction.ContinuedFraction_real` method), 16

M

`multiplicative_order()` (`sage.rings.continued_fraction.ContinuedFraction_base` method), 8

N

`n()` (`sage.rings.continued_fraction.ContinuedFraction_base` method), 8

`numerator()` (`sage.rings.continued_fraction.ContinuedFraction_base` method), 9

`numerical_approx()` (`sage.rings.continued_fraction.ContinuedFraction_base` method), 9

P

`p()` (`sage.rings.continued_fraction.ContinuedFraction_base` method), 9

`period()` (`sage.rings.continued_fraction.ContinuedFraction_periodic` method), 14

`pn()` (`sage.rings.continued_fraction.ContinuedFraction_base` method), 10

`preperiod()` (`sage.rings.continued_fraction.ContinuedFraction_periodic` method), 15

Q

`q()` (`sage.rings.continued_fraction.ContinuedFraction_base` method), 10

`qn()` (`sage.rings.continued_fraction.ContinuedFraction_base` method), 10

`quotient()` (`sage.rings.continued_fraction.ContinuedFraction_infinite` method), 13

`quotient()` (`sage.rings.continued_fraction.ContinuedFraction_periodic` method), 15

`quotient()` (`sage.rings.continued_fraction.ContinuedFraction_real` method), 17

`quotients()` (`sage.rings.continued_fraction.ContinuedFraction_base` method), 10

`quotients()` (`sage.rings.continued_fraction.ContinuedFraction_infinite` method), 13

R

`rat_interval_cf_list()` (in module `sage.rings.continued_fraction`), 23

S

`sage.rings.continued_fraction` (module), 3

`sign()` (`sage.rings.continued_fraction.ContinuedFraction_base` method), 10

`str()` (`sage.rings.continued_fraction.ContinuedFraction_base` method), 11

V

`value()` (`sage.rings.continued_fraction.ContinuedFraction_infinite` method), 13

`value()` (`sage.rings.continued_fraction.ContinuedFraction_periodic` method), 15

`value()` (`sage.rings.continued_fraction.ContinuedFraction_real` method), 18