# Sage Reference Manual: Diophantine approximation

*Release 7.5*

**The Sage Development Team**

**Jan 12, 2017**

CONTENTS

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.

CHAPTER

# ONE

# CONTINUED FRACTIONS

A continued fraction is a representation of a real number in terms of a sequence of integers denoted $[a_0; a_1, a_2, \ldots]$. 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:

$$[a_0; a_1, a_2, \ldots] = a_0 + \frac{1}{[a_1; a_2, \ldots]} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots}}}$$

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

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

$$\pi = 3 + \cfrac{1}{7 + \cfrac{1}{15 + \cfrac{1}{1 + \cfrac{1}{292 + \cfrac{1}{\ddots}}}}}$$

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.<cbrt5> = NumberField(x^3 - 5, embedding=1.709)
sage: continued_fraction(cbrt5)
[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: 12212112211212212112212211221121122112122 1...
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 1...
sage: v = cf.value()
sage: v
1.387954587967143?
sage: v.n(digits=100)
1.
↪3879545879671423369193138598731854778781524524985322718949172898264185776226489321698852370342429
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 range(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 `contfrac` (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_generic` 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)
125534149269984145152881172257540108158836388648008943184302610393086333722107674 8
```

**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 with `prec` bits (or decimal `digits` ) of precision.

INPUT:

- `prec` – precision in bits

- `digits` – precision in decimal digits (only used if `prec` is not given)
- `algorithm` – ignored for continued fractions

If neither `prec` nor `digits` is given, the default precision is 53 bits (roughly 16 digits).

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)
1.2810251332955698155529303809759 0
sage: cf.n(digits=33)
1.2810251332955698155529303809759 0
```

**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)
39437716112122669627437388126007482131572665965887449517273934974469212453530052 83
```

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

Return a numerical approximation of this continued fraction with `prec` bits (or decimal `digits` ) of precision.

INPUT:

- `prec` – precision in bits
- `digits` – precision in decimal digits (only used if `prec` is not given)
- `algorithm` – ignored for continued fractions

If neither `prec` nor `digits` is given, the default precision is 53 bits (roughly 16 digits).

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)
1.2810251332955698155529 3038097590
sage: cf.n(digits=33)
1.2810251332955698155529 3038097590
```

**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
sage: c.numerator(152)
39437716112122669627437388126007482131572665965887449517273934974469212453530052 83
```

**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)
12553414926998414515288117225754010815883638864800894318430261039308633372210767 48
```

**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 the object returned is a `lazy_list_generic` 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
```

```
                                                2 + ---------
                                                    1 + ...
sage: print(continued_fraction(pi).str(nterms=1))
3 + ...
sage: print(continued_fraction(pi).str(nterms=2))
        1
3 + ---------
     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.
↪42238873688278548834154711602456582530687910899171182931189245291645674727256588331245541296
```

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 1.718281828459046? in Real Interval Field with 53 bits of␣
↪precision.
```

**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: 1511515115115151151151151151151151151151151...
```

**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,.
↪..
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...]
sage: cf.value()
e - 1

sage: w = words.FibonacciWord([2,5])
sage: cf = continued_fraction(w)
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 xsrange(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 range(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 lits 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, 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, 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 range(10):
....:     x = RR.random_element()
....:     cff = continued_fraction(x)
....:     cfe = QQ(x).continued_fraction()
....:     assert cff == cfe, "%s %s %s"%(x,cff,cfe)
```

TESTS:

Fixed trac ticket #18901. For RealLiteral, continued_fraction calls continued_fraction_list:

```
sage: continued_fraction(1.575709393346379)
[1; 1, 1, 2, 1, 4, 18, 1, 5, 2, 25037802, 7, 1, 3, 1, 28, 1, 8, 2]
```

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 $a_1$, ... 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 lits 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, 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 20 terms
[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 20 terms
[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, 100000000000000000000, 2688]
sage: continued_fraction_list(1 + 10^-20 - e^-100, nterms=5)
[1, 100000000000000000000, 2688, 8, 1]
sage: continued_fraction_list(1 + 10^-20 - e^-100, nterms=5)
[1, 100000000000000000000, 2688, 8, 1]
```

Fixed trac ticket #18901:

```
sage: a = 1.575709393346379
sage: type(a)
<type 'sage.rings.real_mpfr.RealLiteral'>
sage: continued_fraction_list(a)
[1, 1, 1, 2, 1, 4, 18, 1, 5, 2, 25037802, 7, 1, 3, 1, 28, 1, 8, 2]
```

Check that this works for arb elements (trac ticket #20069):

```
sage: continued_fraction(RBF(e))
[2; 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 1, 1, 12]
```

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 range(10,54):
....:     R = RealIntervalField(20)
....:     for _ in range(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
```

CHAPTER

TWO

# INDICES AND TABLES

- Index
- Module Index
- Search Page

## r

## A



## C



## D



## F



## H



## I



## L

## M



## N



## P



## Q



## R



## S



## V