amgf Documentation

Release 0.7

Alexander Raichev

August 09, 2012

CONTENTS

CHAPTER ONE

THE AMGF MODULE

Let $F(x) = \sum_{\nu \in \mathbf{N}^d} F_\nu x^\nu$ be a multivariate power series with complex coefficients that converges in a neighborhood of the origin. Assume that $F = G/H$ for some functions G and H holomorphic in a neighborhood of the origin. Assume also that H is a polynomial.

This Python module for use within Sage computes asymptotics for the coefficients $F_{r\alpha}$ as $r \to \infty$ with $r\alpha \in \mathbf{N}^d$ for α in a permissible subset of d-tuples of positive reals. More specifically, it computes arbitrary terms of the asymptotic expansion for $F_{r\alpha}$ when the asymptotics are controlled by a multiple point of the alegbraic variety $H = 0$.

The algorithms and formulas implemented here come from [RaWi2008a] and [RaWi2012].

... [DeLo2006] Wolfram Decker and Christoph Lossen, "Computing in algebraic geometry", Chapter 7.1, Springer-Verlag, 2006.

... [DiEm2005] Alicia Dickenstein and Ioannis Z. Emiris (editors), "Solving polynomial equations", Chapter 9.0, Springer-Verlag, 2005.

... [PeWi2008] Robin Pemantle and Mark C. Wilson, "Twenty combinatorial examples of asymptotics derived from multivariate generating functions", SIAM Rev. (2008) vol. 50 (2) pp. 199-272.

AUTHORS:

- Alexander Raichev (2008-10-01): Initial version
- Alexander Raichev (2010-09-28): Corrected many functions
- Alexander Raichev (2010-12-15): Updated documentation
- Alexander Raichev (2011-03-09): Fixed a division by zero bug in relative_error()
- Alexander Raichev (2011-04-26): Rewrote in object-oreinted style
- Alexander Raichev (2011-05-06): Fixed bug in cohomologous_integrand() and fixed _crit_cone_combo() to work in SR
- Alexander Raichev (2012-08-06): Major rewrite. Created class FFPD and moved functions around.

EXAMPLES:

A smooth point example (Example 5.4 of [RaWi2008a]):

```
sage: R.<x,y> = PolynomialRing(QQ)
sage: q = 1/2
sage: qq = q.denominator()
sage: H = 1 - q*x + q*x*y - x^2*y
sage: Hfac = H.factor()
sage: G = (1 - q*x)/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: alpha = list(qq*vector([2, 1 - q]))
sage: print alpha
[4, 1]
sage: I = F.smooth_critical_ideal(alpha)
sage: print I
Ideal (y^2 - 2*y + 1, x + 1/4*y - 5/4) of Multivariate Polynomial Ring
in x, y over Rational Field
sage: s = solve(I.gens(), [SR(x) for x in R.gens()], solution_dict=true)
sage: print s
[{y: 1, x: 1}]
sage: p = s[0]
sage: asy = F.asymptotics(p, alpha, 1)
Creating auxiliary functions...
Computing derivatives of auxiallary functions...
Computing derivatives of more auxiliary functions...
Computing second order differential operator actions...
sage: print asy
(1/12*2^(2/3)*sqrt(3)*gamma(1/3)/(pi*r^(1/3)), 1,
1/12*2^(2/3)*sqrt(3)*gamma(1/3)/(pi*r^(1/3)))
sage: print F.relative_error(asy[0], alpha, [1, 2, 4, 8, 16], asy[1])
Calculating errors table in the form
exponent, scaled Maclaurin coefficient, scaled asymptotic values,
relative errors...
[((4, 1), 0.1875000000, [0.1953794675], [-0.04202382689]), ((8, 2),
0.1523437500, [0.1550727862], [-0.01791367323]), ((16, 4), 0.1221771240,
[0.1230813519], [-0.007400959228]), ((32, 8), 0.09739671811,
[0.09768973377], [-0.003008475766]), ((64, 16), 0.07744253816,
[0.07753639308], [-0.001211929722])]
```

A multiple point example (Example 6.5 of [RaWi2012]):

```
sage: R.<x,y>= PolynomialRing(QQ)
sage: H = (1 - 2*x - y)**2 * (1 - x - 2*y)**2
sage: Hfac = H.factor()
sage: G = 1/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: print F
(1, [(x + 2*y - 1, 2), (2*x + y - 1, 2)])
```

```
sage: I = F.singular_ideal()
sage: print I
Ideal (x - 1/3, y - 1/3) of Multivariate Polynomial Ring in x, y over
Rational Field
sage: p = {x: 1/3, y: 1/3}
sage: print F.is_convenient_multiple_point(p)
(True, 'convenient in variables [x, y]')
sage: alpha = (var('a'), var('b'))
sage: print F.asymptotic_decomposition(alpha)
[(0, []), (-1/9*(2*a^2*y^2 - 5*a*b*x*y + 2*b^2*x^2)*r^2/(x^2*y^2) +
1/9*(5*(a + b)*x*y - 6*a*y^2 - 6*b*x^2)*r/(x^2*y^2) - 1/9*(4*x^2 - 5*x*y
+ 4*y^2)/(x^2*y^2), [(x + 2*y - 1, 1), (2*x + y - 1, 1)])]
sage: print F.asymptotics(p, alpha, 2)
(-3*((2*a^2 - 5*a*b + 2*b^2)*r^2 + (a + b)*r +
3)*((1/3)^(-b)*(1/3)^(-a))^r, (1/3)^(-b)*(1/3)^(-a), -3*(2*a^2 - 5*a*b +
2*b^2)*r^2 - 3*(a + b)*r - 9)
sage: alpha = [4, 3]
sage: asy = F.asymptotics(p, alpha, 2)
sage: print asy
(3*(10*r^2 - 7*r - 3)*2187^r, 2187, 30*r^2 - 21*r - 9)
sage: print F.relative_error(asy[0], alpha, [1, 2, 4, 8], asy[1])
Calculating errors table in the form
exponent, scaled Maclaurin coefficient, scaled asymptotic values,
relative errors...
[((4, 3), 30.72702332, [0.0000000000], [1.000000000]), ((8, 6),
111.9315678, [69.00000000], [0.3835519207]), ((16, 12), 442.7813138,
[387.0000000], [0.1259793763]), ((32, 24), 1799.879232, [1743.000000],
[0.03160169385])]
```

class `amgf.`**`FFPD`**(*numerator=None*, *denominator_factored=None*, *quotient=None*, *reduce_=True*)

Bases: `object`

Represents a fraction with factored polynomial denominator (FFPD) $p/(q_1^{e_1}\cdots q_n^{e_n})$ by storing the parts p and $[(q_1, e_1), \ldots, (q_n, e_n)]$. Here $q_1, \ldots, q_n$ are elements of a 0- or multivariate factorial polynomial ring R , $q_1, \ldots, q_n$ are distinct irreducible elements of R , $e_1, \ldots, e_n$ are positive integers, and p is a function of the indeterminates of R (a Sage Symbolic Expression). An element r with no polynomial denominator is represented as $[r, (,)]$.

AUTHORS:

- Alexander Raichev (2012-07-26)

`algebraic_dependence_certificate`()

Return the ideal J of annihilating polynomials for the set of polynomials `[q**e for (q, e) in self.denominator_factored()]`, which could be the zero ideal. The ideal J lies in a polynomial ring over the field `self.ring().base_ring()` that has `m =`

`len(self.denominator_factored())` indeterminates. Return None if `self.ring()` is None.

EXAMPLES:

```
sage: R.<x, y> = PolynomialRing(QQ)
sage: f = 1/(x^2 * (x*y + 1) * y^3)
sage: ff = FFPD(quotient =f)
sage: J = ff.algebraic_dependence_certificate()
sage: print J
Ideal (1 - 6*T2 + 15*T2^2 - 20*T2^3 + 15*T2^4 - T0^2*T1^3 -
6*T2^5  + T2^6) of Multivariate Polynomial Ring in
T0, T1, T2 over Rational Field
sage: g = J.gens()[0]
sage: df = ff.denominator_factored()
sage: g(*(q**e for q, e in df)) == 0
True

sage: R.<x, y> = PolynomialRing(QQ)
sage: G = exp(x + y)
sage: H = x^2 * (x*y + 1) * y^3
sage: ff = FFPD(G, H.factor())
sage: J = ff.algebraic_dependence_certificate()
sage: print J
Ideal (1 - 6*T2 + 15*T2^2 - 20*T2^3 + 15*T2^4 - T0^2*T1^3 -
6*T2^5 + T2^6) of Multivariate Polynomial Ring in
T0, T1, T2 over Rational Field
sage: g = J.gens()[0]
sage: df = ff.denominator_factored()
sage: g(*(q**e for q, e in df)) == 0
True

sage: f = 1/(x^3 * y^2)
sage: J = FFPD(quotient =f).algebraic_dependence_certificate()
sage: print J
Ideal (0) of Multivariate Polynomial Ring in T0, T1 over
Rational Field

sage: f = sin(1)/(x^3 * y^2)
sage: J = FFPD(quotient =f).algebraic_dependence_certificate()
sage: print J
None
```

`algebraic_dependence_decomposition`(*whole_and_parts=True*)

Return an algebraic dependence decomposition of `self` as a FFPDSum instance. Recursive.

EXAMPLES:

```
sage: R.<x, y> = PolynomialRing(QQ)
sage: f = 1/(x^2 * (x*y + 1) * y^3)
sage: ff = FFPD(quotient =f)
sage: decomp = ff.algebraic_dependence_decomposition()
sage: print decomp
[(0, []), (-x, [(x*y + 1, 1)]), (x^2*y^2 - x*y + 1,
[(y, 3), (x, 2)])]
sage: print decomp.sum().quotient() == f
True
sage: for r in decomp:
...       J = r.algebraic_dependence_certificate()
...       J is None or J == J.ring().ideal()  # The zero ideal
...
True
True
True

sage: R.<x, y> = PolynomialRing(QQ)
sage: G = sin(x)
sage: H = x^2 * (x*y + 1) * y^3
sage: f = FFPD(G, H.factor())
sage: decomp = f.algebraic_dependence_decomposition()
sage: print decomp
[(0, []), (x^4*y^3*sin(x), [(x*y + 1, 1)]),
(-(x^5*y^5 - x^4*y^4 + x^3*y^3 - x^2*y^2 + x*y - 1)*sin(x),
[(y, 3), (x, 2)])]
sage: if decomp.sum().quotient() == G/H:
...       print 'yep'
...
yep
sage: for r in decomp:
...       J = r.algebraic_dependence_certificate()
...       J is None or J == J.ring().ideal()
...
True
True
True
```

NOTE:

Let $f = p/q$ where q lies in a d -variate polynomial ring $K[X]$ for some field K. Let $q_1^{e_1} \cdots q_n^{e_n}$ be the unique factorization of q in $K[X]$ into irreducible factors and let V_i be the algebraic variety $\{x \in L^d : q_i(x) = 0\}$ of q_i over the algebraic closure L of K. By [Raic2012], f can be written as

(*) $\sum p_A / \prod_{i \in A} q_i^{b_i}$,

where the b_i are positive integers, each p_A is a products of p and an element in $K[X]$, and the sum is taken over all subsets $A \subseteq \{1, \ldots, m\}$ such that $|A| \leq d$ and $\{q_i : i \in A\}$ is algebraically independent.

I call (*) an *algebraic dependence decomposition* of f. Algebraic dependence decompositions are not unique.

The algorithm used comes from [Raic2012].

asymptotic_decomposition (*alpha*, *asy_var=None*)

Return a FFPDSum that has the same asymptotic expansion as `self` in the direction `alpha` but each of whose FFPDs has a denominator factorization of the form $[(q_1, 1), \ldots, (q_n, 1)]$, where `n` is at most `d = self.dimension()`. The output results from a Leinartas decomposition followed by a cohomology decomposition.

INPUT:

- `alpha` - A d-tuple of positive integers or symbolic variables.
- `asy_var` - (Optional; default=None) A symbolic variable with respect to which to compute asymptotics. If None is given the set `asy_var = var('r')`.

EXAMPLES:

```
sage: R.<x, y>= PolynomialRing(QQ)
sage: H = (1 - 2*x -y)*(1 - x -2*y)**2
sage: Hfac = H.factor()
sage: G = 1/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: alpha = var('a, b')
sage: r = var('r')
sage: print F.asymptotic_decomposition(alpha, r)
[(0, []), (-1/3*(a*y - 2*b*x)*r/(x*y) + 1/3*(2*x - y)/(x*y),
[(x + 2*y - 1, 1), (2*x + y - 1, 1)])]
```

AUTHORS:

- Alexander Raichev (2012-08-01)

asymptotics (*p*, *alpha*, *N*, *asy_var=None*, *numerical=0*)

Return the first N terms (some of which could be zero) of the asymptotic expansion of the Maclaurin ray coefficients $F_{r\alpha}$ of the function F represented by `self` as $r \to \infty$, where r = `asy_var` and `alpha` is a tuple of positive integers of length `d = self.dimension()`. Assume that F is holomorphic in a neighborhood of the origin, that the denominator factorization of `self` is also the unique factorization of the denominator of F in the local analytic ring at p (not just in the polynomial ring `self.ring()`), that p is a convenient strictly minimal smooth or multiple point with all nonzero coordinates that is critical and nondegenerate for `alpha`.

INPUT:

- `p` - A dictionary with keys that can be coerced to equal `self.ring().gens()`.

- `alpha` - A tuple of length `self.dimension()` of positive integers or, if p is a smooth point, possibly of symbolic variables.
- `N` - A positive integer.
- `numerical` - (Optional; default =0) A natural number. If numerical > 0, then return a numerical approximation of $F_{r alpha}$ with `numerical` digits of precision. Otherwise return exact values.
- `asy_var` - (Optional; default=None) A symbolic variable. The variable of the asymptotic expansion. If none is given, `var('r')` will be assigned.

OUTPUT:

The tuple (asy, exp_scale, subexp_part). Here asy is the sum of the first N terms (some of which might be 0) of the asymptotic expansion of $F_{r\alpha}$ as $r \to \infty$; exp_scale**r is the exponential factor of asy; subexp_part is the subexponential factor of asy.

EXAMPLES:

A smooth point example:

```
sage: R.<x,y>= PolynomialRing(QQ)
sage: H = (1 - x - y - x*y)**2
sage: Hfac = H.factor()
sage: G = 1/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: print F
(1, [(x*y + x + y - 1, 2)])
sage: alpha = [4, 3]
sage: p = {y: 1/3, x: 1/2}
sage: asy = F.asymptotics(p, alpha, 2)
Creating auxiliary functions...
Computing derivatives of auxiallary functions...
Computing derivatives of more auxiliary functions...
Computing second order differential operator actions...
sage: print asy
(1/6000*(3600*sqrt(2)*sqrt(3)*sqrt(5)*sqrt(r)/sqrt(pi) +
463*sqrt(2)*sqrt(3)*sqrt(5)/(sqrt(pi)*sqrt(r)))*432^r, 432,
1/6000*(3600*sqrt(5)*r +
463*sqrt(5))*sqrt(2)*sqrt(3)/(sqrt(pi)*sqrt(r)))
sage: print F.relative_error(asy[0], alpha, [1, 2, 4, 8, 16], asy[1])
Calculating errors table in the form
    exponent, scaled Maclaurin coefficient, scaled asymptotic
    values, relative errors...
    [((4, 3), 2.083333333, [2.092576110], [-0.004436533009]),
    ((8, 6), 2.787374614, [2.790732875], [-0.001204811281]),
    ((16, 12), 3.826259447, [3.827462310], [-0.0003143703383]),
    ((32, 24), 5.328112821, [5.328540787], [-0.00008032229296]),
    ((64, 48), 7.475927885, [7.476079664], [-0.00002030233658])]
```

A multiple point example:

```
sage: R.<x,y,z>= PolynomialRing(QQ)
sage: H = (4 - 2*x - y - z)**2*(4 - x - 2*y - z)
sage: Hfac = H.factor()
sage: G = 16/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: print F
(-16, [(x + 2*y + z - 4, 1), (2*x + y + z - 4, 2)])
sage: alpha = [3, 3, 2]
sage: p = {x: 1, y: 1, z: 1}
sage: asy = F.asymptotics(p, alpha, 2)
Creating auxiliary functions...
Computing derivatives of auxiliary functions...
Computing derivatives of more auxiliary functions...
Computing second-order differential operator actions...
sage: print asy
(4/3*sqrt(3)*sqrt(r)/sqrt(pi) +
47/216*sqrt(3)/(sqrt(pi)*sqrt(r)), 1,
1/216*(288*sqrt(3)*r + 47*sqrt(3))/(sqrt(pi)*sqrt(r)))
sage: print F.relative_error(asy[0], alpha, [1, 2, 4, 8], asy[1])
Calculating errors table in the form
exponent, scaled Maclaurin coefficient, scaled asymptotic values,
relative errors...
[((3, 3, 2), 0.9812164307, [1.515572606], [-0.5445854340]),
((6, 6, 4),
1.576181132, [1.992989399], [-0.2644418580]),
((12, 12, 8), 2.485286378,
[2.712196351], [-0.09130133851]), ((24, 24, 16), 3.700576827,
[3.760447895], [-0.01617884750])]
```

NOTES:

The algorithms used here come from [RaWi2008a] and [RaWi2012].

AUTHORS:

- Alexander Raichev (2008-10-01, 2010-09-28, 2011-04-27, 2012-08-03)

asymptotics_multiple (*p*, *alpha*, *N*, *asy_var*, *coordinate=None*, *numerical=0*)

Does what asymptotics() does but only in the case of a convenient multiple point nondegenerate for alpha and assuming that that the number of distinct irreducible factors of the denominator of `self` is at most `self.dimension()` and that no factors are repeated. Assume also that `p.values()` are not symbolic variables.

INPUT:

- `p` - A dictionary with keys that can be coerced to equal `self.ring().gens()`.
- `alpha` - A tuple of length `d = self.dimension()` of positive integers or, if

p is a smooth point, possibly of symbolic variables.

- `N` - A positive integer.
- `asy_var` - (Optional; default=None) A symbolic variable. The variable of the asymptotic expansion. If none is given, `var('r')` will be assigned.
- `coordinate`- (Optional; default=None) An integer in {0, ..., d-1} indicating a convenient coordinate to base the asymptotic calculations on. If None is assigned, then choose `coordinate =d-1`.
- `numerical` - (Optional; default =0) A natural number. If numerical > 0, then return a numerical approximation of the Maclaurin ray coefficients of `self` with `numerical` digits of precision. Otherwise return exact values.

NOTES:

The formulas used for computing the asymptotic expansion are Theorem 3.4 and Theorem 3.7 of [RaWi2012].

EXAMPLES:

```
sage: R.<x, y, z>= PolynomialRing(QQ)
sage: H = (4 - 2*x - y - z)*(4 - x -2*y - z)
sage: Hfac = H.factor()
sage: G = 16/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: print F
(16, [(x + 2*y + z - 4, 1), (2*x + y + z - 4, 1)])
sage: p = {x: 1, y: 1, z: 1}
sage: alpha = [3, 3, 2]
sage: F.asymptotics_multiple(p, alpha, 2, var('r'))
Creating auxiliary functions...
Computing derivatives of auxiliary functions...
Computing derivatives of more auxiliary functions...
Computing second-order differential operator actions...
(4/3*sqrt(3)/(sqrt(pi)*sqrt(r)) -
25/216*sqrt(3)/(sqrt(pi)*r^(3/2)), 1,
4/3*sqrt(3)/(sqrt(pi)*sqrt(r)) - 25/216*sqrt(3)/(sqrt(pi)*r^(3/2)))

sage: R.<x, y, z>= PolynomialRing(QQ)
sage: H = (1 - x*(1 + y))*(1 - z*x**2*(1 + 2*y))
sage: Hfac = H.factor()
sage: G = 1/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: print F
(1, [(x*y + x - 1, 1), (2*x^2*y*z + x^2*z - 1, 1)])
sage: p = {x: 1/2, z: 4/3, y: 1}
sage: alpha = [8, 3, 3]
sage: F.asymptotics_multiple(p, alpha, 2, var('r'), coordinate =1)
```

```
Creating auxiliary functions...
Computing derivatives of auxiliary functions...
Computing derivatives of more auxiliary functions...
Computing second-order differential operator actions...
(1/172872*(24696*sqrt(3)*sqrt(7)/(sqrt(pi)*sqrt(r)) -
1231*sqrt(3)*sqrt(7)/(sqrt(pi)*r^(3/2)))*108^r, 108,
1/7*sqrt(3)*sqrt(7)/(sqrt(pi)*sqrt(r)) -
1231/172872*sqrt(3)*sqrt(7)/(sqrt(pi)*r^(3/2)))

sage: R.<x, y>= PolynomialRing(QQ)
sage: H = (1 - 2*x - y) * (1 - x - 2*y)
sage: Hfac = H.factor()
sage: G = exp(x + y)/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: print F
(e^(x + y), [(x + 2*y - 1, 1), (2*x + y - 1, 1)])
sage: p = {x: 1/3, y: 1/3}
sage: alpha = (var('a'), var('b'))
sage: F.asymptotics_multiple(p, alpha, 2, var('r'))
(3*((1/3)^(-b)*(1/3)^(-a))^r*e^(2/3), (1/3)^(-b)*(1/3)^(-a),
3*e^(2/3))
```

AUTHORS:

- Alexander Raichev (2008-10-01, 2010-09-28, 2012-08-02)

asymptotics_smooth (*p*, *alpha*, *N*, *asy_var*, *coordinate=None*, *numerical=0*)

Does what asymptotics() does but only in the case of a convenient smooth point and assuming that the denominator of `self` contains no repeated factors.

INPUT:

- `p` - A dictionary with keys that can be coerced to equal `self.ring().gens()`.
- `alpha` - A tuple of length `d = self.dimension()` of positive integers or, if p is a smooth point, possibly of symbolic variables.
- `N` - A positive integer.
- `asy_var` - (Optional; default=None) A symbolic variable. The variable of the asymptotic expansion. If none is given, `var('r')` will be assigned.
- `coordinate`- (Optional; default=None) An integer in {0, ..., d-1} indicating a convenient coordinate to base the asymptotic calculations on. If None is assigned, then choose `coordinate =d-1`.
- `numerical` - (Optional; default =0) A natural number. If numerical > 0, then return a numerical approximation of the Maclaurin ray coefficients of `self` with `numerical` digits of precision. Otherwise return exact values.

NOTES:

The formulas used for computing the asymptotic expansions are Theorems 3.2 and 3.3 [RaWi2008a] with the exponent of H equal to 1. Theorem 3.2 is a specialization of Theorem 3.4 of [RaWi2012] with $n = 1$.

EXAMPLES:

```
sage: R.<x, y>= PolynomialRing(QQ)
sage: H = 1-x-y-x*y
sage: Hfac = H.factor()
sage: G = 1/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: alpha = [3, 2]
sage: p = {y: 1/2*sqrt(13) - 3/2, x: 1/3*sqrt(13) - 2/3}
sage: F.asymptotics_smooth(p, alpha, 2, var('r'), numerical =3)
Creating auxiliary functions...
Computing derivatives of auxiallary functions...
Computing derivatives of more auxiliary functions...
Computing second order differential operator actions...
((0.369/sqrt(r) - 0.0186/r^(3/2))*71.2^r, 71.2,
0.369/sqrt(r) - 0.0186/r^(3/2))

sage: R.<x, y> = PolynomialRing(QQ)
sage: q = 1/2
sage: qq = q.denominator()
sage: H = 1 - q*x + q*x*y - x^2*y
sage: Hfac = H.factor()
sage: G = (1 - q*x)/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: alpha = list(qq*vector([2, 1 - q]))
sage: print alpha
[4, 1]
sage: p = {x: 1, y: 1}
sage: F.asymptotics_smooth(p, alpha, 5, var('r'))
Creating auxiliary functions...
Computing derivatives of auxiallary functions...
Computing derivatives of more auxiliary functions...
Computing second order differential operator actions...
(1/12*2^(2/3)*sqrt(3)*gamma(1/3)/(pi*r^(1/3)) -
1/96*2^(1/3)*sqrt(3)*gamma(2/3)/(pi*r^(5/3)), 1,
1/12*2^(2/3)*sqrt(3)*gamma(1/3)/(pi*r^(1/3)) -
1/96*2^(1/3)*sqrt(3)*gamma(2/3)/(pi*r^(5/3)))
```

AUTHORS:

- Alexander Raichev (2008-10-01, 2010-09-28, 2012-08-01)

static coerce_point (*R*, *p*)

Coerce the keys of the dictionary `p` into the ring `R`. Assume this is possible.

AUTHORS:

- Alexander Raichev (2009-05-18, 2011-04-18, 2012-08-03)

cohomology_decomposition()

Let $p/(q_1^{e_1} \cdots q_n^{e_n})$ be the fraction represented by `self` and let $K[x_1, \ldots, x_d]$ be the polynomial ring in which the q_i lie. Assume that $n \leq d$ and that the gradients of the q_i are linearly independent at all points in the intersection $V_1 \cap \ldots \cap V_n$ of the algebraic varieties $V_i = \{x \in L^d : q_i(x) = 0\}$, where L is the algebraic closure of the field K. Return a FFPDSum f such that the differential form $f dx_1 \wedge \cdots \wedge dx_d$ is de Rham cohomologous to the differential form $p/(q_1^{e_1} \cdots q_n^{e_n}) dx_1 \wedge \cdots \wedge dx_d$ and such that the denominator of each summand of f contains no repeated irreducible factors.

EXAMPLES:

```
sage: R.<x, y>= PolynomialRing(QQ)
sage: print FFPD(1, [(x, 1), (y, 2)]).cohomology_decomposition()
[(0, [])]
```

```
sage: R.<x, y>= PolynomialRing(QQ)
sage: p = 1
sage: qs = [(x*y - 1, 1), (x**2 + y**2 - 1, 2)]
sage: f = FFPD(p, qs)
sage: print f.cohomology_decomposition()
[(0, []), (4/3*x*y + 4/3, [(x^2 + y^2 - 1, 1)]),
(1/3, [(x*y - 1, 1), (x^2 + y^2 - 1, 1)])]
```

NOTES:

The algorithm used here comes from the proof of Theorem 17.4 of [AiYu1983].

AUTHORS:

- Alexander Raichev (2008-10-01, 2011-01-15, 2012-07-31)

crit_cone_combo(*p*, *alpha*, *coordinate=None*)

Return an auxiliary point associated to the multiple point `p` of the factors `self`. For internal use by asymptotics_multiple().

INPUT:

- `p` - A dictionary with keys that can be coerced to equal `self.ring().gens()`.
- `alpha` - A list of rationals.

OUTPUT:

A solution of the matrix equation $y\Gamma = \alpha'$ for y , where Γ is the matrix given by `[FFPD.direction(v) for v in self.log_grads(p)]` and α' is `FFPD.direction(alpha)`

EXAMPLES:

```
sage: R.<x, y>= PolynomialRing(QQ)
sage: p = exp(x)
sage: df = [(1 - 2*x - y, 1), (1 - x - 2*y, 1)]
sage: f = FFPD(p, df)
sage: p = {x: 1/3, y: 1/3}
sage: alpha = (var('a'), var('b'))
sage: print f.crit_cone_combo(p, alpha)
[1/3*(2*a - b)/b, -2/3*(a - 2*b)/b]
```

NOTES:

Use this function only when Γ is well-defined and there is a unique solution to the matrix equation $y\Gamma = \alpha'$. Fails otherwise.

AUTHORS:

- **Alexander Raichev (2008-10-01, 2008-11-25, 2009-03-04, 2010-09-08,** 2010-12-02, 2012-08-02)

critical_cone (*p*, *coordinate=None*)

Return the critical cone of the convenient multiple point `p`.

INPUT:

- `p` - A dictionary with keys that can be coerced to equal `self.ring().gens()` and values in a field.
- `coordinate` - (Optional; default=None) A natural number.

OUTPUT:

A list of vectors that generate the critical cone of `p` and the cone itself, which is None if the values of `p` don't lie in QQ. Divide logarithmic gradients by their component `coordinate` entries. If `coordinate=None`, then search from d-1 down to 0 for the first index j such that for all i `self.log_grads()[i][j] != 0` and set `coordinate =j`.

EXAMPLES:

```
sage: R.<x, y, z>= PolynomialRing(QQ)
sage: G = 1
sage: H = (1 - x*(1 + y))*(1 - z*x**2*(1 + 2*y))
sage: Hfac = H.factor()
sage: G = 1/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: p = {x: 1/2, y: 1, z: 4/3}
sage: print F.critical_cone(p)
([(2, 1, 0), (3, 1, 3/2)], 2-d cone in 3-d lattice N)
```

AUTHORS:

- Alexander Raichev (2010-08-25, 2012-08-02)

denominator()

Return the denominator of `self`.

EXAMPLES:

```
sage: R.<x,y>= PolynomialRing(QQ)
sage: H = (1 - x - y - x*y)**2*(1-x)
sage: Hfac = H.factor()
sage: G = exp(y)/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: print F.denominator()
x^3*y^2 + 2*x^3*y + x^2*y^2 + x^3 - 2*x^2*y - x*y^2 - 3*x^2 - 2*x*y
- y^2 + 3*x + 2*y - 1
```

denominator_factored()

Return the factorization in `self.ring()` of the denominator of `self` but without the unit part.

EXAMPLES:

```
sage: R.<x,y>= PolynomialRing(QQ)
sage: H = (1 - x - y - x*y)**2*(1-x)
sage: Hfac = H.factor()
sage: G = exp(y)/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: print F.denominator_factored()
[(x - 1, 1), (x*y + x + y - 1, 2)]
```

static **diff_all**(*f*, *V*, *n*, *ending=[]*, *sub=None*, *sub_final=None*, *zero_order=0*, *rekey=None*)

Return a dictionary of representative mixed partial derivatives of f from order 1 up to order n with respect to the variables in V. The default is to key the dictionary by all nondecreasing sequences in V of length 1 up to length n. For internal use.

INPUT:

- `f` - An individual or list of $\mathcal{C}^{n+1}$ functions.
- `V` - A list of variables occurring in f.
- `n` - A natural number.
- `ending` - A list of variables in V.
- `sub` - An individual or list of dictionaries.
- `sub_final` - An individual or list of dictionaries.
- `rekey` - A callable symbolic function in V or list thereof.
- `zero_order` - A natural number.

OUTPUT:

The dictionary $s_1 : deriv_1, ..., s_r : deriv_r$. Here $s_1, \ldots, s_r$ is a listing of all nondecreasing sequences of length 1 up to length n over the alphabet V, where $w > v$ in X iff $str(w) > str(v)$, and $deriv_j$ is the derivative of f with respect to the derivative sequence s_j and simplified with respect to the substitutions in sub and evaluated at sub_final. Moreover, all derivatives with respect to sequences of length less than $zero_order$ (derivatives of order less than $zero_order$) will be made zero.

If $rekey$ is nonempty, then $s_1, \ldots, s_r$ will be replaced by the symbolic derivatives of the functions in $rekey$.

If $ending$ is nonempty, then every derivative sequence s_j will be suffixed by $ending$.

EXAMPLES:

I'd like to print the entire dictionaries, but that doesn't yield consistent output order for doctesting. Order of keys changes.:

```
sage: f = function('f', x)
sage: dd = FFPD.diff_all(f, [x], 3)
sage: dd[(x, x, x)]
D[0, 0, 0](f)(x)

sage: d1 = {diff(f, x): 4*x^3}
sage: dd = FFPD.diff_all(f,[x], 3, sub =d1)
sage: dd[(x, x, x)]
24*x

sage: dd = FFPD.diff_all(f,[x], 3, sub=d1, rekey=f)
sage: dd[diff(f, x, 3)]
24*x

sage: a = {x:1}
sage: dd = FFPD.diff_all(f,[x], 3, sub=d1, rekey=f, sub_final=a)
sage: dd[diff(f, x, 3)]
24

sage: X = var('x, y, z')
sage: f = function('f',*X)
sage: dd = FFPD.diff_all(f, X, 2, ending=[y, y, y])
sage: dd[(z, y, y, y)]
D[1, 1, 1, 2](f)(x, y, z)

sage: g = function('g',*X)
sage: dd = FFPD.diff_all([f, g], X, 2)
sage: dd[(0, y, z)]
D[1, 2](f)(x, y, z)

sage: dd[(1, z, z)]
D[2, 2](g)(x, y, z)
```

```
sage: f = exp(x*y*z)
sage: ff = function('ff',*X)
sage: dd = FFPD.diff_all(f, X, 2, rekey=ff)
sage: dd[diff(ff, x, z)]
x*y^2*z*e^(x*y*z) + y*e^(x*y*z)
```

AUTHORS:

- Alexander Raichev (2008-10-01, 2009-04-01, 2010-02-01)

static diff_op (*A, B, AB_derivs, V, M, r, N*)

Return the derivatives $DD^(l+k)(A[j]B^l)$ evaluated at a point p for various natural numbers j, k, l which depend on r and N. Here DD is a specific second-order linear differential operator that depends on M , A is a list of symbolic functions, B is symbolic function, and AB_derivs contains all the derivatives of A and B evaluated at p that are necessary for the computation. For internal use by the functions asymptotics_smooth() and asymptotics_multiple().

INPUT:

- `A` - A single or length `r` list of symbolic functions in the variables `V`.
- `B` - A symbolic function in the variables `V`.
- `AB_derivs` - A dictionary whose keys are the (symbolic) derivatives of `A[0], ..., A[r-1]` up to order `2*N-2` and the (symbolic) derivatives of `B` up to order `2*N`. The values of the dictionary are complex numbers that are the keys evaluated at a common point p.
- `V` - The variables of the `A[j]` and `B`.
- `M` - A symmetric $l \times l$ matrix, where l is the length of `V`.
- `r, N` - Natural numbers.

OUTPUT:

A dictionary whose keys are natural number tuples of the form (j, k, l), where $l \leq 2k$, $j \leq r-1$, and $j+k \leq N-1$, and whose values are $DD^(l+k)(A[j]B^l)$ evaluated at a point p, where DD is the linear second-order differential operator $-\sum_{i=0}^{l-1}\sum_{j=0}^{l-1} M[i][j]\partial^2/(\partial V[j]\partial V[i])$.

EXAMPLES:

```
sage: T = var('x, y')
sage: A = function('A',*tuple(T))
sage: B = function('B',*tuple(T))
sage: AB_derivs = {}
sage: M = matrix([[1, 2],[2, 1]])
sage: DD = FFPD.diff_op(A, B, AB_derivs, T, M, 1, 2)
sage: DD.keys()
```

```
[(0, 1, 2), (0, 1, 1), (0, 1, 0), (0, 0, 0)]
sage: len(DD[(0, 1, 2)])
246
```

AUTHORS:

- Alexander Raichev (2008-10-01, 2010-01-12)

static diff_op_simple (*A, B, AB_derivs, x, v, a, N*)

Return $DD^(ek+vl)(AB^l)$ evaluated at a point p for various natural numbers e, k, l that depend on v and N. Here DD is a specific linear differential operator that depends on a and v , A and B are symbolic functions, and AB_derivs contains all the derivatives of A and B evaluated at p that are necessary for the computation. For internal use by the function asymptotics_smooth().

INPUT:

- `A, B` - Symbolic functions in the variable `x`.
- `AB_derivs` - A dictionary whose keys are the (symbolic) derivatives of `A` up to order `2*N` if `v` is even or `N` if `v` is odd and the (symbolic) derivatives of `B` up to order `2*N + v` if `v` is even or `N + v` if `v` is odd. The values of the dictionary are complex numbers that are the keys evaluated at a common point p.
- `x` - Symbolic variable.
- `a` - A complex number.
- `v, N` - Natural numbers.

OUTPUT:

A dictionary whose keys are natural number pairs of the form (k, l), where $k < N$ and $l \leq 2k$ and whose values are $DD^(ek + vl)(AB^l)$ evaluated at a point p. Here $e = 2$ if v is even, $e = 1$ if v is odd, and DD is the linear differential operator $(a^{-1/v} d/dt)$ if v is even and $(|a|^{-1/v} i \text{sgn}(a) d/dt)$ if v is odd.

EXAMPLES:

```
sage: A = function('A', x)
sage: B = function('B', x)
sage: AB_derivs = {}
sage: FFPD.diff_op_simple(A, B, AB_derivs, x, 3, 2, 2)
{(1, 0): 1/2*I*2^(2/3)*D[0](A)(x), (0, 0): A(x), (1, 1):
1/4*(A(x)*D[0, 0, 0, 0](B)(x) + B(x)*D[0, 0, 0, 0](A)(x) +
4*D[0](A)(x)*D[0, 0, 0](B)(x) + 4*D[0](B)(x)*D[0, 0, 0](A)(x) +
6*D[0, 0](A)(x)*D[0, 0](B)(x))*2^(2/3)}
```

AUTHORS:

- Alexander Raichev (2010-01-15)

static diff_prod (*f_derivs*, *u*, *g*, *X*, *interval*, *end*, *uderivs*, *atc*)

Take various derivatives of the equation $f = ug$, evaluate them at a point c, and solve for the derivatives of u. For internal use by the function asymptotics_multiple().

INPUT:

- `f_derivs` - A dictionary whose keys are all tuples of the form `s + end`, where `s` is a sequence of variables from `X` whose length lies in `interval`, and whose values are the derivatives of a function f evaluated at c.
- `u` - A callable symbolic function.
- `g` - An expression or callable symbolic function.
- `X` - A list of symbolic variables.
- `interval` - A list of positive integers. Call the first and last values n and nn, respectively.
- `end` - A possibly empty list of repetitions of the variable `z`, where `z` is the last element of `X`.
- `uderivs` - A dictionary whose keys are the symbolic derivatives of order 0 to order $n-1$ of `u` evaluated at c and whose values are the corresponding derivatives evaluated at c.
- `atc` - A dictionary whose keys are the keys of c and all the symbolic derivatives of order 0 to order nn of `g` evaluated c and whose values are the corresponding derivatives evaluated at c.

OUTPUT:

A dictionary whose keys are the derivatives of `u` up to order nn and whose values are those derivatives evaluated at c.

EXAMPLES:

I'd like to print out the entire dictionary, but that does not give consistent output for doctesting. Order of keys changes

```
sage: u = function('u', x)
sage: g = function('g', x)
sage: fd = {(x,):1,(x, x):1}
sage: ud = {u(x=2): 1}
sage: atc = {x: 2, g(x=2): 3, diff(g, x)(x=2): 5}
sage: atc[diff(g, x, x)(x=2)] = 7
sage: dd = FFPD.diff_prod(fd, u, g, [x], [1, 2], [], ud, atc)
sage: dd[diff(u, x, 2)(x=2)]
22/9
```

NOTES:

This function works by differentiating the equation $f = ug$ with respect to the variable sequence `s + end`, for all tuples `s` of `X` of lengths in `interval`, evaluating at the point c , and solving for the remaining derivatives of `u`. This function assumes that `u` never appears in the differentiations of $f = ug$ after evaluating at c.

AUTHORS:

- Alexander Raichev (2009-05-14, 2010-01-21)

static diff_seq(*V*, *s*)

Given a list `s` of tuples of natural numbers, return the list of elements of `V` with indices the elements of the elements of `s`. This function is for internal use by the function diff_op().

INPUT:

- `V` - A list.
- `s` - A list of tuples of natural numbers in the interval `range(len(V))`.

OUTPUT:

The tuple `tuple([V[tt] for tt in sorted(t)])`, where `t` is the list of elements of the elements of `s`.

EXAMPLES:

```
sage: V = list(var('x, t, z'))
sage: FFPD.diff_seq(V,([0, 1],[0, 2, 1],[0, 0]))
(x, x, x, x, t, t, z)
```

AUTHORS:

- Alexander Raichev (2009-05-19)

dimension()

Return the number of indeterminates of `self.ring()`.

EXAMPLES:

```
sage: R.<x,y>= PolynomialRing(QQ)
sage: H = (1 - x - y - x*y)**2*(1-x)
sage: Hfac = H.factor()
sage: G = exp(y)/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: print F.dimension()
2
```

static direction(*v*, *coordinate=None*)

Returns `[vv/v[coordinate] for vv in v]` where `coordinate` is the last index of v if not specified otherwise.

INPUT:

- v - A vector.
- coordinate - (Optional; default=None) An index for v.

EXAMPLES:

```
sage: FFPD.direction([2, 3, 5])
(2/5, 3/5, 1)
sage: FFPD.direction([2, 3, 5], 0)
(1, 3/2, 5/2)
```

AUTHORS:

- Alexander Raichev (2010-08-25)

grads (*p*)

Return a list of the gradients of the polynomials `[q for (q, e) in self.denominator_factored()]` evalutated at p.

INPUT:

- p - (Optional: default=None) A dictionary whose keys are the generators of `self.ring()`.

EXAMPLES:

```
sage: R.<x, y>= PolynomialRing(QQ)
sage: p = exp(x)
sage: df = [(x**3 + 3*y^2, 5), (x*y, 2), (y, 1)]
sage: f = FFPD(p, df)
sage: print f
(e^x, [(y, 1), (x*y, 2), (x^3 + 3*y^2, 5)])
sage: print R.gens()
(x, y)
sage: p = None
sage: print f.grads(p)
[(0, 1), (y, x), (3*x^2, 6*y)]

sage: p = {x: sqrt(2), y: var('a')}
sage: print f.grads(p)
[(0, 1), (a, sqrt(2)), (6, 6*a)]
```

AUTHORS:

- Alexander Raichev (2009-03-06)

is_convenient_multiple_point (*p*)

Return True if p is a convenient multiple point of self and False otherwise. Also return a short comment.

INPUT:

- p - A dictionary with keys that can be coerced to equal `self.ring().gens()`.

OUTPUT:

A pair (verdict, comment). In case `p` is a convenient multiple point, verdict =True and comment ='No problem'. In case `p` is not, verdict =False and comment is string explaining why `p` fails to be a convenient multiple point.

EXAMPLES:

```
sage: R.<x, y, z>= PolynomialRing(QQ)
sage: H = (1 - x*(1 + y))*(1 - z*x**2*(1 + 2*y))
sage: df = H.factor()
sage: G = 1/df.unit()
sage: F = FFPD(G, df)
sage: p1 = {x: 1/2, y: 1, z: 4/3}
sage: p2 = {x: 1, y: 2, z: 1/2}
sage: print F.is_convenient_multiple_point(p1)
(True, 'convenient in variables [x, y]')
sage: print F.is_convenient_multiple_point(p2)
(False, 'not a singular point')
```

NOTES:

See [RaWi2012] for more details.

AUTHORS:

- Alexander Raichev (2011-04-18, 2012-08-02)

leinartas_decomposition()

Return a Leinartas decomposition of `self` as a FFPDSum instance.

EXAMPLES:

```
sage: R.<x, y> = PolynomialRing(QQ)
sage: f = 1/x + 1/y + 1/(x*y + 1)
sage: decomp = FFPD(quotient =f).leinartas_decomposition()
sage: print decomp
[(0, []), (1, [(x*y + 1, 1)]), (x + y, [(y, 1), (x, 1)])]
sage: print decomp.sum().quotient() == f
True
sage: for r in decomp:
...       L = r.nullstellensatz_certificate()
...       print L is None
...       J = r.algebraic_dependence_certificate()
...       print J is None or J == J.ring().ideal()
...
True
True
True
True
```

```
True
True

sage: R.<x, y> = PolynomialRing(QQ)
sage: f = sin(x)/x + 1/y + 1/(x*y + 1)
sage: G = f.numerator()
sage: H = R(f.denominator())
sage: ff = FFPD(G, H.factor())
sage: decomp = ff.leinartas_decomposition()
sage: print decomp
[(0, []), (-(x*y^2*sin(x) + x^2*y + x*y + y*sin(x) + x)*y,
[(y, 1)]), ((x*y^2*sin(x) + x^2*y + x*y + y*sin(x) + x)*x*y,
[(x*y + 1, 1)]), (x*y^2*sin(x) + x^2*y + x*y + y*sin(x) + x,
[(y, 1), (x, 1)])]
sage: if decomp.sum().quotient() == f:
...       print 'yep'
...
yep
sage: for r in decomp:
...       L = r.nullstellensatz_certificate()
...       print L is None
...       J = r.algebraic_dependence_certificate()
...       print J is None or J == J.ring().ideal()
...
True
True
True
True
True
True
True
True

sage: R.<x, y, z>= PolynomialRing(GF(2, 'a'))
sage: f = 1/(x * y * z * (x*y + z))
sage: decomp = FFPD(quotient =f).leinartas_decomposition()
sage: print decomp
[(0, []), (1, [(z, 2), (x*y + z, 1)]),
(1, [(z, 2), (y, 1), (x, 1)])]
sage: print decomp.sum().quotient() == f
True
```

NOTE:

Let $f = p/q$ where q lies in a d -variate polynomial ring $K[X]$ for some field K. Let $q_1^{e_1} \cdots q_n^{e_n}$ be the unique factorization of q in $K[X]$ into irreducible factors and let V_i be the algebraic variety $\{x \in L^d : q_i(x) = 0\}$ of q_i over the algebraic closure L of K. By [Raic2012], f can be written as

(*) $\sum p_A / \prod_{i \in A} q_i^{b_i}$,

where the b_i are positive integers, each p_A is a product of p and an element of $K[X]$, and the sum is taken over all subsets $A \subseteq \{1, \ldots, m\}$ such that (1) $|A| \le d$, (2) $\cap_{i \in A} T_i \neq \emptyset$, and (3) $\{q_i : i \in A\}$ is algebraically independent.

In particular, any rational expression in d variables can be represented as a sum of rational expressions whose denominators each contain at most d distinct irreducible factors.

I call (*) a *Leinartas decomposition* of f. Leinartas decompositions are not unique.

The algorithm used comes from [Raic2012].

list()

Convert `self` into a list for printing.

log_grads(*p*)

Return a list of the logarithmic gradients of the polynomials `[q for (q, e) in self.denominator_factored()]` evalutated at `p`.

INPUT:

- `p` - (Optional: default=None) A dictionary whose keys are the generators of `self.ring()`.

NOTE:

The logarithmic gradient of a function f at point p is the vector $(x_1 \partial_1 f(x), \ldots, x_d \partial_d f(x))$ evaluated at p.

EXAMPLES:

```
sage: R.<x, y>= PolynomialRing(QQ)
sage: p = exp(x)
sage: df = [(x**3 + 3*y^2, 5), (x*y, 2), (y, 1)]
sage: f = FFPD(p, df)
sage: print f
(e^x, [(y, 1), (x*y, 2), (x^3 + 3*y^2, 5)])
sage: print R.gens()
(x, y)
sage: p = None
sage: print f.log_grads(p)
[(0, y), (x*y, x*y), (3*x^3, 6*y^2)]

sage: p = {x: sqrt(2), y: var('a')}
sage: print f.log_grads(p)
[(0, a), (sqrt(2)*a, sqrt(2)*a), (6*sqrt(2), 6*a^2)]
```

AUTHORS:

- Alexander Raichev (2009-03-06)

maclaurin_coefficients (*multi_indices*, *numerical=0*)

Returns the Maclaurin coefficients of self that have multi-indices `alpha`, `2*alpha`, ..., `r*alpha`.

INPUT:

- `multi_indices` - A list of tuples of positive integers, where each tuple has length `self.ring().ngens()`.
- `numerical` - (Optional; default =0) A natural number. If positive, return numerical approximations of coefficients with `numerical` digits of accuracy.

OUTPUT:

A dictionary of the form (nu, Maclaurin coefficient of index nu of self).

EXAMPLES:

```
sage: R.<x, y, z> = PolynomialRing(QQ)
sage: H = (4 - 2*x - y - z) * (4 - x - 2*y - z)
sage: Hfac = H.factor()
sage: G = 16/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: alpha = vector([3, 3, 2])
sage: interval = [1, 2, 4]
sage: S = [r*alpha for r in interval]
sage: print F.maclaurin_coefficients(S, numerical =10)
{(6, 6, 4): 0.7005249476, (12, 12, 8): 0.5847732654,
(3, 3, 2): 0.7849731445}
```

NOTES:

Uses iterated univariate Maclaurin expansions. Slow.

AUTHORS:

- Alexander Raichev (2011-04-08, 2012-08-03)

nullstellensatz_certificate ()

Let $[(q_1, e_1), \ldots, (q_n, e_n)]$ be the denominator factorization of `self`. Return a list of polynomials $h_1, \ldots, h_m$ in `self.ring()` that satisfies $h_1 q_1 + \cdots + h_m q_n = 1$ if it exists. Otherwise return None. Only works for multivariate `self`.

EXAMPLES:

```
sage: R.<x, y> = PolynomialRing(QQ)
sage: G = sin(x)
sage: H = x^2 * (x*y + 1)
sage: f = FFPD(G, H.factor())
sage: L = f.nullstellensatz_certificate()
sage: print L
[y^2, -x*y + 1]
```

```
sage: df = f.denominator_factored()
sage: sum([L[i]*df[i][0]**df[i][1] for i in xrange(len(df))]) == 1
True

sage: f = 1/(x*y)
sage: L = FFPD(quotient =f).nullstellensatz_certificate()
sage: L is None
True
```

nullstellensatz_decomposition()

Return a Nullstellensatz decomposition of `self` as a FFPDSum instance. Recursive. Only works for multivariate `self`.

EXAMPLES:

```
sage: R.<x, y> = PolynomialRing(QQ)
sage: f = 1/(x*(x*y + 1))
sage: decomp = FFPD(quotient =f).nullstellensatz_decomposition()
sage: print decomp
[(0, []), (1, [(x, 1)]), (-y, [(x*y + 1, 1)])]
sage: decomp.sum().quotient() == f
True
sage: for r in decomp:
...         L = r.nullstellensatz_certificate()
...         L is None
...
True
True
True

sage: R.<x, y> = PolynomialRing(QQ)
sage: G = sin(y)
sage: H = x*(x*y + 1)
sage: f = FFPD(G, H.factor())
sage: decomp = f.nullstellensatz_decomposition()
sage: print decomp
[(0, []), (sin(y), [(x, 1)]), (-y*sin(y), [(x*y + 1, 1)])]
sage: if decomp.sum().quotient() == G/H:
...         print 'yep'
...
yep
sage: for r in decomp:
...         L = r.nullstellensatz_certificate()
...         L is None
...
True
True
True
```

NOTE:

Let $f = p/q$ where q lies in a d -variate polynomial ring $K[X]$ for some field K and $d \geq 1$. Let $q_1^{e_1} \cdots q_n^{e_n}$ be the unique factorization of q in $K[X]$ into irreducible factors and let V_i be the algebraic variety $\{x \in L^d : q_i(x) = 0\}$ of q_i over the algebraic closure L of K. By [Raic2012], f can be written as

(*) $\sum p_A / \prod_{i \in A} q_i^{e_i}$,

where the p_A are products of p and elements in $K[X]$ and the sum is taken over all subsets $A \subseteq \{1, \ldots, m\}$ such that $\cap_{i \in A} T_i \neq \emptyset$.

I call (*) a *Nullstellensatz decomposition* of f. Nullstellensatz decompositions are not unique.

The algorithm used comes from [Raic2012].

numerator()

Return the numerator of `self`.

EXAMPLES:

```
sage: R.<x,y>= PolynomialRing(QQ)
sage: H = (1 - x - y - x*y)**2*(1-x)
sage: Hfac = H.factor()
sage: G = exp(y)/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: print F.numerator()
-e^y
```

static **permutation_sign**(*s*, *u*)

This function returns the sign of the permutation on `1, ..., len(u)` that is induced by the sublist `s` of `u`. For internal use by cohomology_decomposition().

INPUT:

- `s` - A sublist of `u`.
- `u` - A list.

OUTPUT:

The sign of the permutation obtained by taking indices within `u` of the list `s + sc`, where `sc` is `u` with the elements of `s` removed.

EXAMPLES:

```
sage: u = ['a','b','c','d','e']
sage: s = ['b','d']
sage: FFPD.permutation_sign(s, u)
-1
sage: s = ['d','b']
```

```
sage: FFPD.permutation_sign(s, u)
1
```

AUTHORS:

- Alexander Raichev (2008-10-01, 2012-07-31)

quotient()

Convert `self` into a quotient.

EXAMPLES:

```
sage: R.<x,y>= PolynomialRing(QQ)
sage: H = (1 - x - y - x*y)**2*(1-x)
sage: Hfac = H.factor()
sage: G = exp(y)/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: print F
(-e^y, [(x - 1, 1), (x*y + x + y - 1, 2)])
sage: print F.quotient()
-e^y/(x^3*y^2 + 2*x^3*y + x^2*y^2 + x^3 - 2*x^2*y - x*y^2 - 3*x^2 -
2*x*y - y^2 + 3*x + 2*y - 1)
```

relative_error(*approx*, *alpha*, *interval*, *exp_scale=1*, *digits=10*)

Returns the relative error between the values of the Maclaurin coefficients of `self` with multi-indices `r alpha` for `r` in `interval` and the values of the functions (of the variable `r`) in `approx`.

INPUT:

- `approx` - An individual or list of symbolic expressions in one variable.
- `alpha` - A list of positive integers of length `self.ring().ngens()`
- `interval` - A list of positive integers.
- `exp_scale` - (Optional; default =1) A number.

OUTPUT:

A list whose entries are of the form `[r*alpha, a_r, b_r, err_r]` for `r` in `interval`. Here `r*alpha` is a tuple; `a_r` is the `r*alpha` (multi-index) coefficient of the Maclaurin series for `self` divided by `exp_scale**r`; `b_r` is a list of the values of the functions in `approx` evaluated at `r` and divided by `exp_scale**m`; `err_r` is the list of relative errors `(a_r - f)/a_r` for `f` in `b_r`. All outputs are decimal approximations.

EXAMPLES:

```
sage: R.<x, y>= PolynomialRing(QQ)
sage: H = 1 - x - y - x*y
sage: Hfac = H.factor()
```

```
sage: G = 1/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: alpha = [1, 1]
sage: r = var('r')
sage: a1 = (0.573/sqrt(r))*5.83^r
sage: a2 = (0.573/sqrt(r) - 0.0674/r^(3/2))*5.83^r
sage: es = 5.83
sage: F.relative_error([a1, a2], alpha, [1, 2, 4, 8], es)
Calculating errors table in the form
exponent, scaled Maclaurin coefficient, scaled asymptotic values,
relative errors...
[((1, 1), 0.5145797599, [0.5730000000, 0.5056000000],
[-0.1135300000, 0.01745066667]), ((2, 2), 0.3824778089,
[0.4051721856, 0.3813426871], [-0.05933514614, 0.002967810973]),
((4, 4), 0.2778630595, [0.2865000000, 0.2780750000],
[-0.03108344267, -0.0007627515584]), ((8, 8), 0.1991088276,
[0.2025860928, 0.1996074055], [-0.01746414394, -0.002504047242])]
```

AUTHORS:

- Alexander Raichev (2009-05-18, 2011-04-18, 2012-08-03)

ring()

Return the ring of the denominator of `self`, which is None in the case where `self` doesn't have a denominator.

EXAMPLES:

```
sage: R.<x,y>= PolynomialRing(QQ)
sage: H = (1 - x - y - x*y)**2*(1-x)
sage: Hfac = H.factor()
sage: G = exp(y)/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: print F.ring()
Multivariate Polynomial Ring in x, y over Rational Field
sage: F = FFPD(quotient=G/H)
sage: print F
(e^y/(x^3*y^2 + 2*x^3*y + x^2*y^2 + x^3 - 2*x^2*y - x*y^2 - 3*x^2 -
2*x*y - y^2 + 3*x + 2*y - 1), [])
sage: print F.ring()
None
```

singular_ideal()

Let R be the ring of `self` and H its denominator. Let $Hred$ be the reduction (square-free part) of H. Return the ideal in R generated by $Hred$ and its partial derivatives. If the coefficient field of R is algebraically closed, then the output is the ideal of the singular locus (which is a variety) of the variety of H.

EXAMPLES:

```
sage: R.<x, y, z>= PolynomialRing(QQ)
sage: H = (1 - x*(1 + y))**3*(1 - z*x**2*(1 + 2*y))
sage: df = H.factor()
sage: G = 1/df.unit()
sage: F = FFPD(G, df)
sage: F.singular_ideal()
Ideal (x*y + x - 1, y^2 - 2*y*z + 2*y - z + 1, x*z + y - 2*z + 1)
of Multivariate Polynomial Ring in x, y, z over Rational Field
```

AUTHORS:

- **Alexander Raichev (2008-10-01, 2008-11-20, 2010-12-03, 2011-04-18,** 2012-08-03)

smooth_critical_ideal(*alpha*)

Let R be the ring of `self` and H its denominator. Return the ideal in R of smooth critical points of the variety of H for the direction `alpha`. If the variety V of H has no smooth points, then return the ideal in R of V.

INPUT:

- `alpha` - A d-tuple of positive integers and/or symbolic entries, where d = `self.ring().ngens()`.

EXAMPLES:

```
sage: R.<x, y> = PolynomialRing(QQ)
sage: H = (1-x-y-x*y)^2
sage: Hfac = H.factor()
sage: G = 1/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: alpha = var('a1, a2')
sage: F.smooth_critical_ideal(alpha)
Ideal (y^2 + 2*a1/a2*y - 1, x + (a2/(-a1))*y + (-a2 + a1)/(-a1))
of Multivariate Polynomial Ring in x, y over Fraction Field of
Multivariate Polynomial Ring in a2, a1 over Rational Field
```

```
sage: R.<x, y> = PolynomialRing(QQ)
sage: H = (1-x-y-x*y)^2
sage: Hfac = H.factor()
sage: G = 1/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: alpha = [7/3, var('a')]
sage: F.smooth_critical_ideal(alpha)
Ideal (y^2 + (-14/(-3*a))*y - 1, x + (-3/7*a)*y + 3/7*a - 1) of
Multivariate Polynomial Ring in x, y over Fraction Field of
Univariate Polynomial Ring in a over Rational Field
```

NOTES:

See [RaWi2012] for more details.

AUTHORS:

- **Alexander Raichev (2008-10-01, 2008-11-20, 2009-03-09, 2010-12-02,** 2011-04-18, 2012-08-03)

static subs_all (*f*, *sub*, *simplify=False*)

Return the items of f substituted by the dictionaries of sub in order of their appearance in sub.

INPUT:

- `f` - An individual or list of symbolic expressions or dictionaries
- `sub` - An individual or list of dictionaries.
- `simplify` - Boolean (default: False).

OUTPUT:

The items of f substituted by the dictionaries of sub in order of their appearance in sub. The subs() command is used. If simplify is True, then simplify() is used after substitution.

EXAMPLES:

```
sage: var('x, y, z')
(x, y, z)
sage: a = {x:1}
sage: b = {y:2}
sage: c = {z:3}
sage: FFPD.subs_all(x + y + z, a)
y + z + 1
sage: FFPD.subs_all(x + y + z, [c, a])
y + 4
sage: FFPD.subs_all([x + y + z, y^2], b)
[x + z + 2, 4]
sage: FFPD.subs_all([x + y + z, y^2], [b, c])
[x + 5, 4]
```

```
sage: var('x, y')
(x, y)
sage: a = {'foo': x**2 + y**2, 'bar': x - y}
sage: b = {x: 1 , y: 2}
sage: FFPD.subs_all(a, b)
{'foo': 5, 'bar': -1}
```

AUTHORS:

- Alexander Raichev (2009-05-05)

univariate_decomposition()

Return the usual univariate partial fraction decomposition of `self` as a FFPDSum instance. Assume that `self` lies in the field of fractions of a univariate factorial polynomial ring.

EXAMPLES:

One variable:

```
sage: R.<x> = PolynomialRing(QQ)
sage: f = 5*x^3 + 1/x + 1/(x-1) + 1/(3*x^2 + 1)
sage: print f
(15*x^7 - 15*x^6 + 5*x^5 - 5*x^4 + 6*x^3 - 2*x^2 + x - 1)/(3*x^4 -
3*x^3 + x^2 - x)
sage: decomp = FFPD(quotient =f).univariate_decomposition()
sage: print decomp
[(5*x^3, []), (1, [(x - 1, 1)]), (1, [(x, 1)]),
(1/3, [(x^2 + 1/3, 1)])]
sage: print decomp.sum().quotient() == f
True
```

One variable with numerator in symbolic ring:

```
sage: R.<x> = PolynomialRing(QQ)
sage: f = 5*x^3 + 1/x + 1/(x-1) + exp(x)/(3*x^2 + 1)
sage: print f
e^x/(3*x^2 + 1) + ((5*(x - 1)*x^3 + 2)*x - 1)/((x - 1)*x)
sage: decomp = FFPD(quotient =f).univariate_decomposition()
sage: print decomp
[(e^x/(3*x^2 + 1) + ((5*(x - 1)*x^3 + 2)*x - 1)/((x - 1)*x), [])]
```

One variable over a finite field:

```
sage: R.<x> = PolynomialRing(GF(2))
sage: f = 5*x^3 + 1/x + 1/(x-1) + 1/(3*x^2 + 1)
sage: print f
(x^6 + x^4 + 1)/(x^3 + x)
sage: decomp = FFPD(quotient =f).univariate_decomposition()
sage: print decomp
[(x^3, []), (1, [(x, 1)]), (x, [(x + 1, 2)])]
sage: print decomp.sum().quotient() == f
True
```

One variable over an inexact field:

```
sage: R.<x> = PolynomialRing(CC)
sage: f = 5*x^3 + 1/x + 1/(x-1) + 1/(3*x^2 + 1)
sage: print f
(15.0000000000000*x^7 - 15.0000000000000*x^6 + 5.00000000000000*x^5
- 5.00000000000000*x^4 + 6.00000000000000*x^3 -
```

```
2.00000000000000*x^2 + x - 1.00000000000000)/(3.00000000000000*x^4
- 3.00000000000000*x^3 + x^2 - x)
sage: decomp = FFPD(quotient =f).univariate_decomposition()
sage: print decomp
[(5.00000000000000*x^3, []), (1.00000000000000,
[(x - 1.00000000000000, 1)]), (-0.288675134594813*I,
[(x - 0.577350269189626*I, 1)]), (1.00000000000000, [(x, 1)]),
(0.288675134594813*I, [(x + 0.577350269189626*I, 1)])]
sage: print decomp.sum().quotient() == f # Rounding error coming
False
```

NOTE:

Let $f = p/q$ be a rational expression where p and q lie in a univariate factorial polynomial ring R. Let $q_1^{e_1} \cdots q_n^{e_n}$ be the unique factorization of q in R into irreducible factors. Then f can be written uniquely as

(*) $p_0 + \sum_{i=1}^m p_i/q_i^{e_i}$,

for some $p_j \in R$. I call (*) the *usual partial fraction decomposition* of f.

AUTHORS:

- Robert Bradshaw (2007-05-31)
- Alexander Raichev (2012-06-25)

class amgf.**FFPDSum**

Bases: list

A list representing the sum of FFPD objects with distinct denominator factorizations.

AUTHORS:

- Alexander Raichev (2012-06-25)

combine_like_terms()

Combine terms in self with the same denominator. Only useful for multivariate decompositions.

EXAMPLES:

```
sage: R.<x, y>= PolynomialRing(QQ)
sage: f = FFPD(quotient =1/(x * y * (x*y + 1)))
sage: g = FFPD(quotient =x/(x * y * (x*y + 1)))
sage: s = FFPDSum([f, g, f])
sage: t = s.combine_like_terms()
sage: print s
[(1, [(y, 1), (x, 1), (x*y + 1, 1)]), (1, [(y, 1), (x*y + 1, 1)]),
(1, [(y, 1), (x, 1), (x*y + 1, 1)])]
sage: print t
[(1, [(y, 1), (x*y + 1, 1)]), (2, [(y, 1), (x, 1), (x*y + 1, 1)])]
```

```
sage: R.<x, y>= PolynomialRing(QQ)
sage: H = x * y * (x*y + 1)
sage: f = FFPD(1, H.factor())
sage: g = FFPD(exp(x + y), H.factor())
sage: s = FFPDSum([f, g])
sage: print s
[(1, [(y, 1), (x, 1), (x*y + 1, 1)]), (e^(x + y), [(y, 1), (x, 1),
(x*y + 1, 1)])]
sage: t = s.combine_like_terms()
sage: print t
[(e^(x + y) + 1, [(y, 1), (x, 1), (x*y + 1, 1)])]
```

ring()
Return the polynomial ring of the denominators of `self`. If `self` doesn't have any denominators, then return None.

sum()
Return the sum of the FFPDs in `self` as a FFPD.

EXAMPLES:

```
sage: R.<x, y> = PolynomialRing(QQ)
sage: df = (x, 1), (y, 1), (x*y + 1, 1)
sage: f = FFPD(2, df)
sage: g = FFPD(2*x*y, df)
sage: print FFPDSum([f, g])
[(2, [(y, 1), (x, 1), (x*y + 1, 1)]), (2, [(x*y + 1, 1)])]
sage: print FFPDSum([f, g]).sum()
(2, [(y, 1), (x, 1)])

sage: R.<x, y> = PolynomialRing(QQ)
sage: f = FFPD(cos(x), [(x, 2)])
sage: g = FFPD(cos(y), [(x, 1), (y, 2)])
sage: print FFPDSum([f, g])
[(cos(x), [(x, 2)]), (cos(y), [(y, 2), (x, 1)])]
sage: print FFPDSum([f, g]).sum()
(y^2*cos(x) + x*cos(y), [(y, 2), (x, 2)])
```

whole_and_parts()
Rewrite `self` as a FFPDSum of a (possibly zero) polynomial FFPD followed by reduced rational expression FFPDs. Only useful for multivariate decompositions.

EXAMPLES:

```
sage: R.<x, y> = PolynomialRing(QQ, 'x, y')
sage: f = x**2 + 3*y + 1/x + 1/y
sage: f = FFPD(quotient =f)
sage: print f
(x^3*y + 3*x*y^2 + x + y, [(y, 1), (x, 1)])
```

```
sage: print FFPDSum([f]).whole_and_parts()
[(x^2 + 3*y, []), (x + y, [(y, 1), (x, 1)])]

sage: R.<x, y> = PolynomialRing(QQ)
sage: f = cos(x)**2 + 3*y + 1/x + 1/y
sage: print f
1/x + 1/y + cos(x)^2 + 3*y
sage: G = f.numerator()
sage: H = R(f.denominator())
sage: f = FFPD(G, H.factor())
sage: print f
(x*y*cos(x)^2 + 3*x*y^2 + x + y, [(y, 1), (x, 1)])
sage: print FFPDSum([f]).whole_and_parts()
[(0, []), (x*y*cos(x)^2 + 3*x*y^2 + x + y, [(y, 1), (x, 1)])]
```

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

BIBLIOGRAPHY

[AiYu1983] I.A. Aizenberg and A.P. Yuzhakov, "Integral representations and residues in multidimensional complex analysis", Translations of Mathematical Monographs, 58. American Mathematical Society, Providence, RI, 1983. x+283 pp. ISBN: 0-8218-4511-X.

[Raic2012] Alexander Raichev, "Leinartas's partial fraction decomposition", http://arxiv.org/abs/1206.4740.

[RaWi2008a] Alexander Raichev and Mark C. Wilson, "Asymptotics of coefficients of multivariate generating functions: improvements for smooth points", Electronic Journal of Combinatorics, Vol. 15 (2008), R89, http://arxiv.org/pdf/0803.2914.pdf.

[RaWi2012] Alexander Raichev and Mark C. Wilson, "Asymptotics of coefficients of multivariate generating functions: improvements for smooth points", To appear in 2012 in the Online Journal of Analytic Combinatorics, http://arxiv.org/pdf/1009.5715.pdf.

PYTHON MODULE INDEX

a

INDEX

R

S

U

W