Integration of Rational Functions

Alkis Akritas
ECE/UTh/Volos

Abstract

In these notes we examine the integration of rational functions. This is a very important process, because other cases — such as integration of transcendental functions — are reduced to it.

We present Ostrogradsky's[1] method and Hermite's reduction method for the computation of the rational part of the integral and two methods for the computation of the logarithmic part.

Table of contents

1 Introduction

Throughout these notes we will use the computer algebra system `Sympy`, which can be initialized as follows:

```
>>> from sympy import *
>>> var('a:z')

   (a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z)

Python]
```

We begin our discussion with some basic definitions we will need.

1. For more information see: http://en.wikipedia.org/wiki/Mikhail_Ostrogradsky

1.1 Basic Notions and Definitions

Consider the polynomials $p(x)$, $q(x)$ with integer coefficients, i.e. $p(x)$, $q(x) \in \mathbb{Z}[x]$; the expression $\frac{p(x)}{q(x)}$ is called a *rational function*, and $p(x)$, $q(x)$ are called the *numerator*, *denominator* polynomial, respectively. The polynomials $p(x)$, $q(x)$ are not uniquely determined, since $\frac{1}{x+1} = \frac{7}{7x+7} = \frac{x^2+1}{(x+1)(x^2+1)}$.

Given the rational function $f(x) = \frac{p(x)}{q(x)}$ — and assuming that the polynomials $p(x)$, $q(x)$ have no common factors (cancel them if they do) — the roots of $p(x)$ are called the *zeros* of $f(x)$, whereas the roots of $q(x)$ are called the *poles* of $f(x)$. The root ϱ of $q(x)$ is a pole of $f(x)$ if $\lim_{x\to\varrho} |f(x)| = \infty$. The multiplicity of a zero (or pole) ϱ of $f(x)$ is the multiplicity of the root ϱ of $p(x)$ (or $q(x)$).

The *factored* or *pole-zero* form of $f(x) = \frac{p(x)}{q(x)}$ is

$$f(x) = \frac{p(x)}{q(x)} = k \cdot \frac{(x-r_1)\cdots(x-r_m)}{(x-s_1)\cdots(x-s_n)},$$

where

i. $m = \deg(p), n = \deg(q)$ are the degrees of $p(x), q(x)$, respectively,

ii. $k = \frac{\mathrm{lc}(p)}{\mathrm{lc}(q)}$ is the leading coefficient of $p(x)$ divided by the leading coefficient of $q(x)$,

iii. $r_1, ..., r_m$ are the zeros of $f(x)$ — that is, the roots of $p(x)$, and

iv. $s_1, ..., s_n$ are the poles of $f(x)$ — that is, the roots of $q(x)$.

Assuming that $f(x) = \frac{p(x)}{q(x)}$ has no repeated poles (i.e. all poles of $f(x)$ have multiplicity 1) and that $m = \deg(p) < n = \deg(q)$, then we can have the *partial fraction expansion* of $f(x)$ as

$$f(x) = \frac{p(x)}{q(x)} = \frac{f_1}{x-s_1} + \cdots + \frac{f_n}{x-s_n},$$

where

i. $s_1, ..., s_n$ are the poles of $f(x)$

ii. $f_1, ..., f_n$ are called the *residues*,

iii. when $s_k = \bar{s_l}$, the complex conjugate of s_l, then $f_k = \bar{f_l}$.

The word residue is used in a number of different contexts in mathematics. Two of the most common uses are the complex residue of a pole — used here — and the remainder of a congruence. The residue in our case is the coefficient a_{-1} in the Laurent series

$$f(z) = \sum_{n=-\infty}^{\infty} a_n (z - z_0)^n$$

of $f(z)$ about the point z_0.

For example, as can be easily verified with the functions `apart`, `together` and `residue` of `sympy`, we have

$$\frac{x^2-2}{x^3+3x^2+2x} = \frac{-1}{x} + \frac{1}{x+1} + \frac{1}{x+2}$$

```
>>> f = (x**2 - 2) / (x**3 + 3*x**2 + 2*x)
>>> fr = apart( f )
>>> fr
   1/(x + 2) + 1/(x + 1) - 1/x
>>> together( fr )
   (x*(x + 1) + x*(x + 2) - (x + 1)*(x + 2))/(x*(x + 1)*(x + 2))
>>> residue( f, x, 0)
   -1
>>> series(f, x, x0=0)
```

```
   -1/x + 3/2 - 5*x/4 + 9*x**2/8 - 17*x**3/16 + 33*x**4/32 - 65*x**5/64 + O(x**6)
>>> residue( f, x, -1)
   1
>>> series(f, x, x0=-1)
   1/(x + 1) + 2 + 2*(x + 1)**2 + 2*(x + 1)**4 + O((x + 1)**6, (x, -1))
>>> residue( f, x, -2)
   1
>>> series(f, x, x0=-2)
   1/(x + 2) - 2 - 7*(x + 2)**2/8 - 15*(x + 2)**3/16 - 31*(x + 2)**4/32 - 63*(x +
   2)**5/64 - 3*x/4 + O((x + 2)**6, (x, -2))
Python]
```

1.2 Computing the Partial Fraction Expansion

From the above discussion we see that in order to compute the partial fraction expansion of a rational function $f(x) = \frac{p(x)}{q(x)}$ we need to:

i. compute the poles $s_1, ..., s_n$ of $f(x)$ — i.e. compute the roots of $q(x)$, and

ii. compute the residues $f_1, ..., f_n$.

To compute the poles of $f(x)$ we can compute the roots of $q(x)$ either with the function `factor` or with the function `solve`.

Once we have the poles, the residues $f_1, ..., f_n$ can be computed with several methods. We demonstrate them using the following example

$$f(x) = \frac{a_2 \cdot x^2 + a_1 \cdot x + a_0}{(x - s_1)(x - s_2)(x - s_3)} = \frac{f_1}{x - s_1} + \frac{f_2}{x - s_2} + \frac{f_3}{x - s_3}. \tag{1}$$

1.2.1 Method 1: Computation of Residues by Solving a System of Linear Equations

With this method we clear the denominators in (1) to obtain

$$a_2 \cdot x^2 + a_1 \cdot x + a_0 = f_1 \cdot (x - s_2) \cdot (x - s_3) + f_2 \cdot (x - s_1) \cdot (x - s_3) + f_3 \cdot (x - s_1) \cdot (x - s_2);$$

next equate coefficients

i. $a_0 = (s_2 \cdot s_3) \cdot f_1 + (s_1 \cdot s_3) \cdot f_2 + (s_1 \cdot s_2) \cdot f_3$

ii. $a_1 = (-s_2 - s_3) \cdot f_1 + (-s_1 - s_3) \cdot f_2 + (-s_1 - s_2) \cdot f_3$

iii. $a_2 = f_1 + f_2 + f_3$

and solve the system of linear equations for f_1, f_2, f_3.

1.2.2 Method 2: Computation of Residues by Evaluating $(x - s_k) \cdot f(x)$ at the Poles s_k

With this method, to compute $f_i, 1 \leqslant i \leqslant 3$, we do the following:

i. multiply both sides of (1) by $(x - s_i)$,

ii. cancel $(x - s_i)$ from numerator and denominator, and

iii. evaluate at $x = s_i$.

In other words, we have the general formula:

$$f_k = (x - s_k) \cdot f(x)|_{x=s_k}, \tag{2}$$

which means do the last three steps (i), (ii), (iii).

So, to compute f_1 in our example we first multiply both sides of (1) by $(x - s_1)$ to obtain — after cancellations:

$$\begin{aligned}\frac{(x-s_1)(a_2\cdot x^2+a_1\cdot x+a_0)}{(x-s_1)(x-s_2)(x-s_3)} &= \frac{f_1\cdot(x-s_1)}{x-s_1}+\frac{f_2\cdot(x-s_1)}{x-s_2}+\frac{f_3\cdot(x-s_1)}{x-s_3}\\ \frac{(a_2\cdot x^2+a_1\cdot x+a_0)}{(x-s_2)(x-s_3)} &= f_1+\frac{f_2\cdot(x-s_1)}{x-s_2}+\frac{f_3\cdot(x-s_1)}{x-s_3};\end{aligned}$$

then replacing x by s_1 we have the explicit formula

$$\frac{a_2\cdot s_1^2+a_1\cdot s_1+a_0}{(s_1-s_2)(s_1-s_3)}=f_1.$$

We can compute f_2, f_3 the same way.

1.2.3 Method 3: Computation of Residues by Evaluating $\frac{p(x)}{q'(x)}$ at the Poles s_k

With this method, we have $f_i=\frac{p(s_i)}{q'(s_i)}, 1\leqslant i\leqslant 3$. To see this, note that from formula (2) we have

$$f_k=\lim_{x\to s_k}\frac{(x-s_k)\cdot p(x)}{q(x)}=\lim_{x\to s_k}\frac{p(x)+p'(x)(x-s_k)}{q'(x)}=\frac{p(s_k)}{q'(s_k)},$$

where l'Hopital's rule was used.

1.2.4 Repeated Poles

Suppose now that we have

$$f(x)=\frac{p(x)}{q(x)}=\frac{p(x)}{(x-s_1)^{k_1}\cdots(x-s_l)^{k_l}}, \tag{3}$$

where:

i. $m=\deg(p)<n=\deg(q)$, and

ii. the poles s_i are distinct — that is, $s_i\neq s_j$ for $i\neq j$ and have multiplicity k_i.

Then the partial fraction expansion of (3) has the form

$$\begin{aligned}f(x) &= \frac{f_{1,k_1}}{(x-s_1)^{k_1}}+\frac{f_{1,k_1-1}}{(x-s_1)^{k_1-1}}+\cdots+\frac{f_{1,1}}{(x-s_1)}\\ &+ \frac{f_{2,k_2}}{(x-s_2)^{k_2}}+\frac{f_{2,k_2-1}}{(x-s_2)^{k_2-1}}+\cdots+\frac{f_{2,1}}{(x-s_2)}\\ &\vdots\ \cdots\\ &+ \frac{f_{l,k_l}}{(x-s_l)^{k_l}}\frac{f_{l,k_l-1}}{(x-s_{2l})^{k_l-1}}+\cdots+\frac{f_{l,1}}{(x-s_l)}.\end{aligned}$$

Here, we have n residues, just as before and the terms involve higher powers of $\frac{1}{(x-s)}$. Consequently, formula (2) above becomes

$$f_{i,k_i}=(x-s_i)^{k_i}f(x)|_{x=s_i} \tag{4}$$

from which we obtain the residue f_{i,k_i}. To obtain the other residues, f_{i,k_i-1}, ..., $f_{i,1}$ we use the following extension

$$f_{i,k_i-j}=\frac{1}{j!}\frac{d^j}{dx^j}((x-s_i)^{k_i}\cdot f(x))|_{x=s_i}. \tag{5}$$

Example 1. The rational function $f(x)=\frac{1}{x^2(x+1)}$ has the partial fraction expansion

$$f(x)=\frac{f_1}{x^2}+\frac{f_2}{x}+\frac{f_3}{x+1}=\frac{1}{x^2}-\frac{1}{x}+\frac{1}{x+1}.$$

With method 2, the residues f_1, f_3 are computed using formulae (4) and (2), respectively, whereas residue f_2 is computed using formula (5). So we have:

$$\begin{aligned} f_1 &= x^2 f(x)|_{x=0} = \frac{1}{x+1}|_{x=0} = 1 \\ f_2 &= \frac{d}{dx}(x^2 \cdot f(x))|_{x=0} = \frac{-1}{(x+1)^2}|_{x=0} = -1 \\ f_3 &= (x+1) \cdot f(x)|_{x=-1} = \frac{1}{x^2}|_{x=-1} = 1 \end{aligned}$$

Note that `sympy`'s function `residue` returns the residues $f_{i,1}$, and, therefore, using it we obtain f_2, f_3.

```
Python] f = 1 / ( x**2 * (x + 1) )
Python] residue(f, x, 0)
    -1
Python] residue(f, x, -1)
    1
Python] apart(f)
    1/(x + 1) - 1/x + x**(-2)
Python]
```

1.3 The Road Ahead

In the sequel we will compute — in several stages — the integral $\int \frac{p(x)}{q(x)} \mathrm{dx}$, where $p(x) = 4x^7 + 4x^6 + 16x^5 + 12x^4 + 8x^3$ and $q(x) = x^6 + 2x^5 + 3x^4 + 4x^3 + 3x^2 + 2x + 1$.

```
Python] p =  4*x**7 + 4*x**6 + 16*x**5 + 12*x**4 + 8*x**3
Python] q = x**6 + 2*x**5 + 3*x**4 + 4*x**3 + 3*x**2 + 2*x + 1
Python]
Python]
```

Before we present the algorithms for the various stages we reveal the answer which is obtained with the function `integrate`:

```
Python] integrate( p / q, x )
    2*x**2 - 4*x + (4*x**2 + 3*x + 5)/(x**3 + x**2 + x + 1) + 9*log(x + 1) + 3*log(x**2 +
   1)/2 - 3*atan(x)
Python]
```

Notice that the above expression contains *three* parts:

i. the *polynomial part*, which in our case is

$$2x^2 - 4x, \tag{6}$$

ii. the *rational part*, which in our case is

$$\frac{4x^2 + 3x + 5}{x^3 + x^2 + x + 1}, \tag{7}$$

iii. and the *logarithmic* (or *transcendental*) part, which in our case is

$$9 \cdot \log(x+1) + \frac{3 \cdot \log(x^2+1)}{2} - 3 \cdot \operatorname{atan}(x). \tag{8}$$

In the sequel we elaborate on each part individually.

2 Polynomial Part of the Integral

This is the easiest part to compute. It appears only when the degree of the numerator polynomial $p(x)$ is greater than the degree of the denominator polynomial $q(x)$ — as is the case with our polynomials presented in the Introduction. In such cases, all we have to do is take the quotient and remainder of $p(x)$ and $q(x)$.

Integrating the quotient of $p(x)$ and $q(x)$ — quite an easy task to perform — we obtain the polynomial part of the integral we are after. In our case, to obtain the result in (6) we have:

```
Python] pp = quo(p, q, x)
Python] pp

    4*x - 4

Python] integrate( pp, x )

    2*x**2 - 4*x

Python]
```

On the other hand, the remainder provides us with a *new* polynomial $p(x)$, which will be the numerator of a new rational function having the same denominator $q(x)$. However, this time the degree of the numerator is smaller than that of the denomintor.

```
Python] rem(p, q, x)

    12*x**5 + 8*x**4 + 12*x**3 + 4*x**2 + 4*x + 4

Python]
```

Below is the algorithm to compute the polynomial part:

```
Python] def int_poly_part(p, q, x):
            """
            Input: Polynomials p, q with deg(p) <=> deg(q).
            Output: If deg(p) > deg(q) computes the polynomial part of the integral
            ∫ p/q and a new polynomial, p, deg(p) < deg(q), which will be the numerator
            of a new rational function with the same denominator q.
            """
            poly_part = quo(p, q, x)
            r = rem(p, q, x)
            return [integrate(poly_part, x), r]
Python]
```

For the polynomials $p(x)$, $q(x)$ given in the Introduction we obtain the polynomial part (6) of the integral and a new polynomial $p(x)$, $\deg(p) < \deg(q)$, to be used in the following stages.

```
Python] [poly_part, p] = int_poly_part(p, q, x)
Python] poly_part

    2*x**2 - 4*x

Python] p
```

```
    12*x**5 + 8*x**4 + 12*x**3 + 4*x**2 + 4*x + 4
Python]
```

3 Rational Part of the Integral

At this point we have the new pair of polynomials $p(x), q(x)$, where $p(x)$ changed, whereas $q(x)$ remained the same and $\deg(p) < \deg(q)$. That is, we have

```
Python] p =  12*x**5 + 8*x**4 + 12*x**3 + 4*x**2 + 4*x + 4
Python] q =  x**6 + 2*x**5 + 3*x**4 + 4*x**3 + 3*x**2 + 2*x + 1
Python]
```

Below we go through the steps that lead to (7), the rational part of the integral computed in the Introduction. In other words, we will compute the integral $\int \frac{p(x)}{q(x)} \mathrm{dx}$.

We present two methods for the computation of the rational part: Ostrogradsky's method of 1845 and Hermite's reduction method of 1872.

3.1 Ostrogradsky's Method of 1845 for the Rational Part

Our discussion in this section is taken from the paper "How to Integrate Rational Functions" by T. N. Subramaniam and D. E. G. Malm.

Ostrogradsky's method is based on his theorem, which states the following:

Theorem 2. ***(Ostrogradsky, 1845)*** *Let $\frac{p}{q}$ be a rational function. Let $q = \prod_{i=1}^{n} h_i^{\alpha_i}$ be the factorization of q into linear and irreducible quadratic factors, and let $q_1 = \prod_{i=1}^{n} h_i^{\alpha_i - 1}$ and $q_2 = \prod_{i=1}^{n} h_i$. Then there are polynomials p_1, p_2 such that*

$$\int \frac{p(x)}{q(x)} \mathrm{dx} = \frac{p_1(x)}{q_1(x)} + \int \frac{p_2(x)}{q_2(x)} \mathrm{dx}. \tag{9}$$

Proof. *Here $q(x), q_1(x), q_2(x)$ are known polynomials of degrees $\deg(q), \deg(q_1), \deg(q_2)$, respectively; $p(x)$ is also a known polynomial of degree not greater than $\deg(q) - 1$. By contrast, $p_1(x)$, $p_2(x)$ are unknown polynomials — of degrees not greater than $\deg(q_1) - 1, \deg(q_2) - 1$ respectively — and (their coefficients) have to be computed.*

The proof can be found in the paper by Subramaniam and Malm and hence it is omitted here. □

What is really nice about formula (9) is the fact that we can compute $p_1(x)$, $p_2(x)$, $q_1(x)$, $q_2(x)$ ***without actually factoring $q(x)$***! Here is how:

From a theorem we stated when discussing square-free factorization we see that $q_1(x) = \gcd\,(q(x), q'(x))$, from which it easily follows that $q_2(x) = \frac{q(x)}{q_1(x)}$. That is, for our example, the denominator polynomials $q_1(x)$, $q_2(x)$ in (9) are:

```
Python] q1 = gcd( q, diff(q, x, 1) )
Python] q1
    x**3 + x**2 + x + 1
```

```
Python] q2 = quo( q, q1 )
Python] q2
    x**3 + x**2 + x + 1
Python]
```

To compute $p_1(x)$, $p_2(x)$ in (9), both of which are of degree $\leqslant 2$, note that because of the above relations, $q_1(x)$ divides $q'(x)$. However, since $q(x) = q_1(x)q_2(x)$, $q_1(x)$ also divides the product $q_1'(x)q_2(x)$ and, therefore, $s(x) = \frac{q_1'(x)q_2(x)}{q_1(x)}$ is a polynomial. Differentiating both sides of (9) we obtain

$$\frac{p(x)}{q(x)} = \frac{q_1(x)p_1'(x) - p_1(x)q_1'(x)}{q_1^2(x)} + \frac{p_2(x)}{q_2(x)} = \frac{p_1'(x) - p_1(x)\frac{q_1'(x)}{q_1(x)}}{q_1(x)} + \frac{p_2(x)}{q_2(x)}.$$

Clearing the denominators and keeping in mind that $q(x) = q_1(x)q_2(x)$, we have

$$p(x) = p_1'(x)q_2(x) - p_1(x)s(x) + p_2(x)q_1(x). \tag{10}$$

Hence, since $p(x), q_1(x), q_2(x)$ and $s(x)$ are known polynomials, we solve for $p_1(x), p_2(x)$ with the method of undertermined coefficients.

Since the polynomial $p_1(x), p_2(x)$, in (9), are both of degree $\leqslant 2$, suppose that $p_1(x) = w_2 \cdot x^2 + w_1 \cdot x + w_0$ and $p_2(x) = z_2 \cdot x^2 + z_1 \cdot x + z_0$. In `sympy` these polynomials are formed with the help of the functions `numbered_symbols`, `take` and `sum`.

```
Python] coeffs_p1 = take( numbered_symbols(w), 3 )
Python] coeffs_p1
    [w0, w1, w2]
Python] p1 = sum( [coeffs_p1[i]*x**i for i in range(3)] )
Python] p1
    w0 + w1*x + w2*x**2
Python] coeffs_p2 = take( numbered_symbols(z), 3 )
Python] coeffs_p2
    [z0, z1, z2]
Python] p2 = sum( [coeffs_p2[i]*x**i for i in range(3)] )
Python] p2
    x**2*z2 + x*z1 + z0
Python]
```

Compute now the polynomial $s(x)$, and substitute it in (10) to obtain the polynomial

$$\begin{aligned} t(x) &= z_2 \cdot x^5 + (-w_2 + z_1 + z_2) \cdot x^4 + (-2 \cdot w_1 + z_0 + z_1 + z_2) \cdot x^3 \\ &+ (-3 \cdot w_0 - w_1 + w_2 + z_0 + z_1 + z_2) \cdot x^2 + (-2 \cdot w_0 + 2 \cdot w_2 + z_0 + z_1) \cdot x \\ &+ (-w_0 + w_1 + z_0). \end{aligned}$$

```
Python] s = diff( q1, x, 1 ) * q2 / q1
Python] s
    3*x**2 + 2*x + 1
```

```
Python] t = (
               diff( p1, x, 1 ) * q2 - p1 * s + p2 * q1

           ).expand().collect(x)
Python] t
    -w0 + w1 + x**5*z2 + x**4*(-w2 + z1 + z2) + x**3*(-2*w1 + z0 + z1 + z2) + x**2*(-3*w0
   - w1 + w2 + z0 + z1 + z2) + x*(-2*w0 + 2*w2 + z0 + z1) + z0
Python]
```

Now, set $t(x) - p(x) = 0$ and solve the following system of undetermined coefficients

$$\begin{aligned} z_2 &= 12 \\ -w_2 + z_2 + z_1 &= 8 \\ -2w_1 + z_2 + z_1 + z_0 &= 12 \\ w_2 - w_1 - 3w_0 + z_2 + z_1 + z_0 &= 4 \\ 2w_2 - 2w_0 + z_1 + z_0 &= 4 \\ w_1 - w_0 + z_0 &= 4 \end{aligned}$$

In `sympy` this is done quite easily by setting $t(x) - p(x) = 0$ (the 0 is omitted in the function `solve_undetermined_coeffs`) and specifying the list of undetermined coefficients:

```
Python] sols = solve_undetermined_coeffs( t - p, coeffs_p1 + coeffs_p2, x )
Python] sols
    {z1: 0, w1: 3, w0: 5, z0: 6, z2: 12, w2: 4}
```

Therefore, the numerator polynomials $p_1(x), p_2(x)$ in (9) are

$$\begin{aligned} p_1(x) &= 4 \cdot x^2 + 3 \cdot x + 4 \\ p_2(x) &= 12 \cdot x^2 + 6, \end{aligned}$$

```
Python] p1 = p1.subs(sols)
Python] p1
    4*x**2 + 3*x + 5
Python] p2 = p2.subs(sols)
Python] p2
    12*x**2 + 6
Python]
```

and the integral becomes:

$$\int \frac{12x^5 + 8x^4 + 12x^3 + 4x^2 + 4x + 4}{x^6 + 2x^5 + 3x^4 + 4x^3 + 3x^2 + 2x + 1} \mathrm{dx} = \frac{4x^2 + 3x + 5}{x^3 + x^2 + x + 1} + \int \frac{12x^2 + 6}{x^3 + x^2 + x + 1} \mathrm{dx}.$$

The rational part $\frac{4x^2 + 3x + 5}{x^3 + x^2 + x + 1}$ is the same as (7), which was computed in the Introduction, whereas the integral $\int \frac{12x^2 + 6}{x^3 + x^2 + x + 1} \mathrm{dx}$ and will give us the logarithmic part.

The above procedure can be made into the function `int_rat_part` presented below.

```
Python] def int_rat_part_0(p, q, x):
            """
            Input: Polynomials p, q with deg(p) < deg(q).
            Output: p1/q1 the rational part of the integral ∫ p/q and p2/q2,
            the integrand of the logarithmic (transcendental) part.
            """
            # form q1, q2 and take their degrees
            q1 = gcd( q, diff(q, x, 1) )
            q2 = quo( q, q1, x )
            dq1 = degree( q1, x )
            dq2 = degree( q2, x )

            # form p1, p2 with undetermined coefficients
            coeffsp1 = take( numbered_symbols(w), dq1 )
            p1 = sum( [coeffsp1[i]*x**i for i in range(dq1)] )
            coeffsp2 = take( numbered_symbols(z), dq2 )
            p2 = sum( [coeffsp2[i]*x**i for i in range(dq2)] )

            # form s, t
            s = quo( diff(q1, x, 1)*q2 , q1, x )
            t = ( diff( p1, x, 1 )*q2 - p1*s + p2*q1
            ).expand().collect(x)

            # solve system to find coefficients of p1, p2
            slts = solve_undetermined_coeffs( t - p, coeffsp1 + coeffsp2, x )

            # form rational part p1 / q1
            if degree(p1, x) >= 0:
                p1 = p1.subs(slts)
            rat_part = p1 / q1

            # form log part p2 / q2
            if degree(p2, x) >= 0:
                p2 = p2.subs(slts)
            log_part = p2 / q2

            ## simplify numerators, denominators
            temp = together( rat_part )
            temp_num = expand( numer( temp ) )
            temp_den = expand( denom( temp ) )
            rat_part = temp_num / temp_den

            temp1 = together( log_part )
            temp1_num = expand( numer( temp1 ) )
            temp1_den = expand( denom( temp1 ) )
            log_part = temp1_num / temp1_den

            return [rat_part, log_part]
Python]
```

The integral of the rational function for the polynomials $p(x), q(x)$ of our example gives both a rational part and a logarithmic part.

```
Python] p = 12*x**5 + 8*x**4 + 12*x**3 + 4*x**2 + 4*x + 4
Python] q = x**6 + 2*x**5 + 3*x**4 + 4*x**3 + 3*x**2 + 2*x + 1
Python] [rat_part, log_part] = int_rat_part_O(p, q, x)
Python] rat_part
    -1/(5*x - 5)
Python] log_part
    -1/(5*x**2 + 15*x - 20)
Python]
```

The computation of the logarithmic part, that is the computation of the integral

$$\int \frac{12x^2+6}{x^3+x^2+x+1}\mathrm{dx} \tag{11}$$

will be discussed in the next section. In the following subsection below we elaborate on the second approach to compute the rational part.

3.2 Hermite's Method of 1872 for the Rational Part

Almost 30 years after Ostrogradsky, Hermite "reinvented" the integration of rational functions.

Hermite's method provides a constructive means for obtaining the rational part of the integral of a rational function and requires only rational operations.

The method has the following two stages:

i. first, we obtain the complete squarefree partial fraction expansion (defined below) of the integrand,

ii. and then to *each* fraction, computed in (i), with the denominator raised to a power $j>1$, we apply a reduction scheme producing two functions: one of these is the rational part of the integral, whereas the integral of the other is the logarithmic (or transcendental) part of the original integral.

Even though the required concepts mentioned above are known, we remind ourselves that:

Definition 3. *Let $p(x)\in\mathbb{Z}[x]$ be a polynomial of positive degree. Then $p(x)$ is said to be **square-free** if it cannot be written in the form $p(x)=q(x)\cdot r^2(x)$, where $r(x)$ is a polynomial of positive degree.*

It follows that a square-free polynomial has only roots of multiplicity one.

Definition 4. *Let $p(x)\in\mathbb{Z}[x]$ and suppose $p(x)=p_0\cdot\prod_{i=1}^{k}p_i^i(x)$, where each $p_i(x)\in\mathbb{Z}[x]$, $p_i(x)$ is primitive and has positive leading coefficient for $1\leqslant i\leqslant k$. Also, $\deg(p_k)>0$, $p_0\in\mathbb{Z}$ and the polynomials $p_i(x)$ are pairwise relatively prime. Then the expression $p_0\cdot\prod_{i=1}^{k}p_i^i(x)$ is called the **square-free factorization** of $p(x)$.*

In `sympy` the functions `sqf` and `sqf_list` compute the square-free factorization of a given polynomial. The output follows *Mathematica* and is slightly different from the output of `Xcas` and `maxima` — but otherwise suits our purposes fine. On this topic see issue #8695 for `sympy`.

Definition 5. *Let $\frac{p(x)}{q(x)}$ be a rational function with $\deg(p)<\deg(q)$ and let $q_0\cdot\prod_{i=1}^{k}q_i^i(x)$ be the square-free factorization of $q(x)$. Suppose, moreover, that there exist polynomials $p_i(x)\in\mathbb{Q}[x]$, $1\leqslant i\leqslant k$, such that*

$\frac{p(x)}{q(x)} = \sum_{i=1}^{k} \frac{p_i(x)}{q_i^i(x)}$, $\deg(p_i) < \deg(q_i^i)$, *or if* $\deg(q_i) = 0$ *then* $p_i(x) = 0$. *Then, this sum is called a* ***square-free partial fraction expansion*** *of* $\frac{p(x)}{q(x)}$.

We will not exactly make use of the above expansion. Instead, we will use the following:

Definition 6. *Let* $\frac{p(x)}{q(x)}$ *be a rational function with* $\deg(p) < \deg(q)$ *and let* $q_0 \cdot \prod_{i=1}^{k} q_i^i(x)$ *be the square-free factorization of* $q(x)$. *Suppose, moreover, that there exist polynomials* $p_{i,j}(x) \in \mathbb{Q}[x]$, $1 \leqslant j \leqslant i, 1 \leqslant i \leqslant k$, *such that* $\frac{p(x)}{q(x)} = \sum_{i=1}^{k} \sum_{j=1}^{i} \frac{p_{i,j}(x)}{q_i^j(x)}$, $\deg(p_{i,j}) < \deg(q_i)$, *or if* $\deg(q_i) = 0$ *then* $p_{i,j}(x) = 0$, *for* $1 \leqslant j \leqslant i$. *Then, this sum is called a* ***complete, square-free partial fraction expansion*** *of* $\frac{p(x)}{q(x)}$.

In other words, the partial fraction expansion discussed in the Introduction is a complete, square-free partial fraction expansion. `Sympy` has two functions for computing complete, square-free partial fraction expansions: the first is `apart`, which returns the complete, square-free partial fraction expansions mentioned above, and the second is `apart_list`, which is useful for later processing of the output.

For example, consider the rational function $\frac{1}{(x^2+1)(x-1)^2(x-2)^3(x-3)^3}$. Its complete, square-free partial fraction expansion is

$$\begin{aligned} \frac{1}{(x^2+1)(x-1)^2(x-2)^3(x-3)^3} &= \frac{-(x+1)}{1000(x^2+1)} + \frac{7}{32(x-1)} + \frac{1}{16(x-1)^2} - \frac{66}{125(x-2)} \\ &- \frac{1}{25(x-2)^2} - \frac{1}{5(x-2)^3} + \frac{1241}{4000(x-3)} - \frac{23}{200(x-3)^2} \\ &+ \frac{1}{40(x-3)^3} \end{aligned} \tag{12}$$

and is obtained from `sympy`'s function `apart`:

```
Python] f = 1 / ( (x**2 + 1) * (x - 1)**2 * (x - 2)**3 * (x - 3)**3)
Python] fr = apart( f )
Python] fr

    -(x + 1)/(1000*(x**2 + 1)) + 7/(32*(x - 1)) + 1/(16*(x - 1)**2) - 66/(125*(x - 2)) -
   1/(25*(x - 2)**2) - 1/(5*(x - 2)**3) + 1241/(4000*(x - 3)) - 23/(200*(x - 3)**2) +
   1/(40*(x - 3)**3)

Python]
```

Note that we can get a list of all the terms in an expansion using the attribute `args`. So for our example we have:

```
Python] fr_terms = fr.args
Python] fr_terms

    (-1/(25*(x - 2)**2), -1/(5*(x - 2)**3), 1/(40*(x - 3)**3), -23/(200*(x - 3)**2),
   1/(16*(x - 1)**2), -66/(125*(x - 2)), 7/(32*(x - 1)), -(x + 1)/(1000*(x**2 + 1)),
   1241/(4000*(x - 3)))

Python]
```

Once we have a list of all the terms of the complete square free partial fraction expansion we apply to those with denominator raised to power $j > 1$, Hermite's reduction scheme, which is described below for the general case.

Suppose we want to compute the integral $\int \frac{p(x)}{q(x)}\mathrm{dx}$, where $p(x)$, $q(x) \in \mathbb{Q}(x)$, with $\mathrm{LC}(q) = 1$, i.e. $q(x)$ is monic, $\gcd(p, q) = 1$ and $\deg(p) < \deg(q)$. Hermite's method proceeds as follows.

We compute the partial fraction expansion of $\frac{p(x)}{q(x)}$ in the form

$$\frac{p(x)}{q(x)} = \sum_{i=1}^{k} \sum_{j=1}^{i} \frac{p_{i,j}(x)}{q_i^j(x)}.$$

Then, the integral of $\frac{p(x)}{q(x)}$ can be expressed in the form

$$\int \frac{p(x)}{q(x)}\mathrm{dx} = \sum_{i=1}^{k} \sum_{j=1}^{i} \int \frac{p_{i,j}(x)}{q_i^j(x)}\mathrm{dx}. \tag{13}$$

Our goal now is to apply reductions on the integrals appearing in the right-hand side of (13) until each integral that remains has a denominator which is simply square-free — rather than a power $j > 1$ of a square-free $q_i(x)$. To achieve this, we will use *integration by parts* — the known formula $\int u\,\mathrm{dv} = u \cdot v - \int v\,\mathrm{du}$ — and the extended Euclidean algorithm.

Consider a particular non-zero integrand $\frac{p_{i,j}(x)}{q_i^j(x)}$, with $j > 1$. Since $q_i(x)$ is square-free $\gcd(q_i, q_i') = 1$, and using the extended Euclidean algorithm we can compute polynomials $s(x), t(x) \in \mathbb{Q}[x]$ such that

$$s(x) \cdot q_i(x) + t(x) \cdot q_i'(x) = p_{i,j}(x), \tag{14}$$

and $\deg(s) < \deg(q_i) - 1 = \deg(q_i')$ and $\deg(t) < \deg(q_i)$; the degree requirement is *very* important. Dividing (14) by $q_i^j(x)$ yields

$$\int \frac{p_{i,j}(x)}{q_i^j(x)}\mathrm{dx} = \int \frac{s(x)}{q_i^{j-1}(x)}\mathrm{dx} + \int \frac{t(x) \cdot q_i'(x)}{q_i^j(x)}\mathrm{dx}.$$

Now we can apply integration by parts to the second integral on the right, where

$$u = t(x), v = \frac{-1}{(j-1)q_i^{j-1}(x)}.$$

The result is

$$\int \frac{t(x) \cdot q_i'(x)}{q_i^j(x)}\mathrm{dx} = \frac{-t(x)}{(j-1)q_i^{j-1}(x)} + \int \frac{t'(x)}{(j-1)q_i^{j-1}(x)}\mathrm{dx},$$

and, hence, we have achieved the reduction

$$\int \frac{p_{i,j}(x)}{q_i^j(x)}\mathrm{dx} = \frac{-t(x)/(j-1)}{q_i^{j-1}(x)} + \int \frac{s(x) + t'(x)/(j-1)}{(j-1)q_i^{j-1}(x)}\mathrm{dx}. \tag{15}$$

Note that this process has produced a rational function — which contributes to the overall rational function of the integral — and another integral where the power of $q_i(x)$ has been reduced by one. It may happen that the numerator of the new integrand is zero, in which case the process terminates. Otherwise, if $j - 1 = 1$ the integral contributes to the logarithmic part, whereas if $j - 1 > 1$ the same process maybe applied again. Note that the new integrand satisfies the degree inequality

$$\deg(s + t'/(j-1)) \leqslant \max\{\deg(s), \deg(t')\} < \deg(q_i) - 1,$$

which is consistent with Definition (6).

By repeated application of the reduction process (15), until the denominators of all remaining integrands are square-free, we obtain the full rational part of the integral. Thus we now have Hermite's method for rational function integration.

```
Python] def int_rat_part_H(p, q, x):
            """
            Input: Polynomials p, q with deg(p) < deg(q).
            Output: p1/q1 the rational part of the integral ∫ p/q and p2/q2,
            the integrand of the logarithmic (transcendental) part.
            """
            rat_part = []
            log_part = []
            # Compute a list of all fractions in the complete, square-free partial
            # fraction expansion of p / q.
            pfe = apart(p / q)
            lpf = Add.make_args( pfe )

            # apply reduction process to each fraction in the list
            for fr in lpf:
                fa = factor_list(denom(fr))
                expo = fa[1][0][1]

                # append it to log_part if expo = 1
                if expo == 1:
                    log_part.append(fr)

                else:
                    n = expo
                    # get the numerator of fr
                    nm = numer(fr) / fa[0]
                    # get square-free factor
                    poly = fa[1][0][0]
                    while n > 1:
                        pp = diff(poly, x)
                        # compute s, t such that deg(s) < deg(pp), deg(t) < deg(poly)
                        [s, t, g] = gcdex(poly, pp)
                        s = s * nm
                        s = rem(s, pp, x)  ## ensure degree requirement
                        t = t * nm
                        t = rem(t, poly, x)  ## ensure degree requirement
                        n = n - 1
                        rat_part.append((-t / n) / poly**n)
                        nm = s + diff(t, x) / n
                    log_part.append(nm / poly)

            ## simplify numerator, denominator
            temp = together( sum(rat_part) )
            temp_num = expand( numer( temp ) )
            temp_den = expand( denom( temp ) )
            rat_part = temp_num / temp_den

            temp1 = together( sum(log_part) )
            temp1_num = expand( numer( temp1 ) )
            temp1_den = expand( denom( temp1 ) )
            log_part = temp1_num / temp1_den

            return [rat_part,  log_part]
```

```
Python]

Python] int_rat_part_H(1, (x**2 + 1) * (x - 1)**2 * (x - 2)**3 * (x - 3)**3, x)

    [(37*x**4 - 227*x**3 + 342*x**2 + 148*x - 400)/(400*x**5 - 4400*x**4 + 18800*x**3 -
   38800*x**2 + 38400*x - 14400), (370*x**3 + 1380*x**2 + 330*x + 1420)/(4000*x**5 -
   24000*x**4 + 48000*x**3 - 48000*x**2 + 44000*x - 24000)]

Python] int_rat_part_O(1, (x**2 + 1) * (x - 1)**2 * (x - 2)**3 * (x - 3)**3, x)

    [(37*x**4 - 227*x**3 + 342*x**2 + 148*x - 400)/(400*x**5 - 4400*x**4 + 18800*x**3 -
   38800*x**2 + 38400*x - 14400), (37*x**3 + 138*x**2 + 33*x + 142)/(400*x**5 - 2400*x**4
   + 4800*x**3 - 4800*x**2 + 4400*x - 2400)]

Python] int_rat_part_H(735*x**4 + 441*x**2 -(S(12446)/3)*x + S(21854)/9, x**6 +
        (S(2)/3)*x**5 - (S(65)/9)*x**4 + (S(20)/9)*x**3 + 15*x**2 - (S(154)/9)*x +
        S(49)/9, x)

    [(-6615*x**3 + 6615*x**2 + 6762*x - 6272)/(9*x**4 - 6*x**3 - 36*x**2 + 54*x - 21), 0]

Python] int_rat_part_O(735*x**4 + 441*x**2 -(S(12446)/3)*x + S(21854)/9, x**6 +
        (S(2)/3)*x**5 - (S(65)/9)*x**4 + (S(20)/9)*x**3 + 15*x**2 - (S(154)/9)*x +
        S(49)/9, x)

    [(-6615*x**3 + 6615*x**2 + 6762*x - 6272)/(9*x**4 - 6*x**3 - 36*x**2 + 54*x - 21), 0]

Python]
```

4 Logarithmic (or Transcendental) Part

As we saw, the integral of the rational function $\int \frac{p(x)}{q(x)}\mathrm{dx}$ presented in the Introduction gave us a polynomial part, a rational part and the logarithmic part $\int \frac{12x^2+6}{x^3+x^2+x+1}\mathrm{dx}$.

To evaluate the integral above we could, of course, use the method of partial fractions. However, instead of that, we use the following theorem mentioned in the paper by T. M. Subramaniam and D. E. G. Malm (p. 766),

$$\int \frac{p(x)}{q(x)}\mathrm{dx} = \sum \frac{p(a)}{q'(a)}\log(x-a), \tag{16}$$

where the sum ranges over all the roots a of $q(x)$ — including the complex ones — and log is the complex logarithm defined in `sympy` by the function `log`.

```
Python] log(-3)

    log(3) + I*pi

Python] log(I)

    I*pi/2

Python]
```

The method based on theorem (16) is superior to the method of partial fractions.

Recall that the fraction $b = \frac{p(a)}{q'(a)}$, for some root a of $q(x)$ is called the residue of $\frac{p(x)}{q(x)}$. Note that

i. b is a residue if and only if $p(a) - b \cdot q'(a) = 0$ for some a such that $q(a) = 0$ and this holds if and only if $p(x) - b \cdot q'(x)$ and $q(x)$ have a common root.

ii. if $\gcd(p(x) - b \cdot q'(x), q(x)) = r(x)$, then the roots of $r(x)$ are precisely the roots of $q(x)$ which have b as their residue.

Collecting together terms with the same coefficient in (16) we obtain

$$\int \frac{p(x)}{q(x)} \mathrm{dx} = \sum_i b_i \cdot \log(\, r_i(x)\,), \tag{17}$$

where the b_i are the complex numbers b such that $p(x) - b \cdot q'(x)$ and $q(x)$ have a common root and $r_i(x) = \gcd(p(x) - b_i \cdot q'(x), q(x))$. Equivalently, $p(x) - b \cdot q'(x)$ and $q(x)$ have a common root if their *resultant* (defined below) is zero.

Therefore, if we can compute the residues b_i, then a gcd calculation — most probably over an extension field — will give us the integral.

Both algorithms discussed below are based on the *resultant* of $p(x) - b \cdot q'(x)$ and $q(x)$, where $p(x), q(x)$ are the numerator and denominator, respectively, of the integrand.

The *resultant* of two polynomials $a(x), b(x)$ is defined as the product of differences of the roots of the two polynomials, i.e.,

$$\mathrm{res}(a, b) = \prod_{i=1}^{n} \prod_{j=1}^{m} (\alpha_i - \beta_j),$$

where $a(x) = (x - \alpha_1) \cdots (x - \alpha_n)$ and $b(x) = (x - \beta_1) \cdots (x - \beta_m)$ are monic polynomials split into linear factors.

Clearly, the resultant of two polynomials is 0 if and only if the two polynomials share a root. An important result states that the resultant of two polynomials can be computed from only their coefficients by taking the determinant of the Sylvester matrix (of 1840) of the two polynomials — to be discussed elsewhere.

4.1 Logarithmic Part with GCD Computations in Extension Fields

For the logarithmic part of our example, the polynomials are:

```
Python] p = 3*x + 3
Python] q = x**3 + x**2 + x + 1
Python] qq = diff(q, x, 1)
Python] qq
    3*x**2 + 2*x + 1
Python]
```

The roots of $q(x)$ are $-1, -i$, and i.

```
Python] roots = solve( q, x )
Python] roots
    [-1, -I, I]
Python]
```

and according to formula (16), the integral is

```
Python] sum( [ simplify( ( p.subs(x, roots[i]) / qq.subs(x, roots[i] ) ) ) * log(x -
        roots[i]) for i in range( len( roots ) ) ] )

    -3*I*log(x - I)/2 + 3*I*log(x + I)/2
```

as can be verified by `sympy` — *after* we rewrite the output.

```
Python] t = integrate(p / q, x)
Python] t

    3*atan(x)

Python] t.rewrite(log)

    3*I*log((-I*x + 1)/(I*x + 1))/2
```

The same result is also obtained with residues and gcd computations.

```
Python] residues = [ simplify( ( p.subs(x, roots[i]) / qq.subs(x, roots[i]) ) ) for i in
        range(len(roots))]
Python] residues

    [0, 3*I/2, -3*I/2]

Python] gcd( p - residues[1]*qq, q, x)

    x + I

Python] gcd( p - residues[2]*qq, q, x)

    x - I

Python]
```

Then, the integral is the sum in (17):

```
Python] residues[1]*log(x + I) + residues[2]*log(x - I)

    -3*I*log(x - I)/2 + 3*I*log(x + I)/2

Python]
```

Another example is the following:

```
Python] p = x
Python] q = x**4 + 1
Python] qq = diff(q, x, 1)
Python] roots = solve( q, x)
Python] roots

    [-sqrt(2)/2 - sqrt(2)*I/2, -sqrt(2)/2 + sqrt(2)*I/2, sqrt(2)/2 - sqrt(2)*I/2,
   sqrt(2)/2 + sqrt(2)*I/2]

Python] residues = [ simplify( quo( p.subs(x, roots[i]), qq.subs(x, roots[i]), x ) ) for i
        in range(len(roots))]
Python] residues
```

```
    [-I/4, I/4, I/4, -I/4]
Python] gcds = [ gcd( p - residues[i]*qq, q, x) for i in range(len(roots)) ]
Python] gcds
    [x**2 - I, x**2 + I, x**2 + I, x**2 - I]
Python] logcombine( sum([ residues[i]*log( gcds[i] ) for i in range( len( roots ) ) ]),
        force=True )
    I*log(sqrt(x**2 + I)/sqrt(x**2 - I))
Python] t = integrate(p / q, x )
Python] t
    atan(x**2)/2
Python] t.rewrite(log)
    I*log((-I*x**2 + 1)/(I*x**2 + 1))/4
Python]
```

The same result can be computed with the resultant — where we introduce the variable z.

```
Python] resultant(p - z*qq, q, x)
    256*z**4 + 32*z**2 + 1
Python] solve( resultant(p - z*qq, q, x), z)
    [-I/4, I/4]
Python] gcd( p - (-I/4)*qq, q, x)
    x**2 - I
Python] gcd( p - (I/4)*qq, q, x)
    x**2 + I
Python] logcombine((-I/4)*log(gcd( p - (-I/4)*qq, q, x)) + (I/4)*log(gcd( p - (I/4)*qq, q,
        x)), force=true)
    I*log((x**2 + I)**(1/4)/(x**2 - I)**(1/4))
Python]
```

So far the roots were computed in closed form. In the following example below, this is not possible and the gcd's have to be computed in a field extended by `RootOf`.

```
Python] p = x**2 - 1
Python] q = x**6 - x + 1
Python] c = solve(q)
Python] c
    [RootOf(x**6 - x + 1, 0), RootOf(x**6 - x + 1, 1), RootOf(x**6 - x + 1, 2),
   RootOf(x**6 - x + 1, 3), RootOf(x**6 - x + 1, 4), RootOf(x**6 - x + 1, 5)]
Python] map(N, c)
    [-0.94540233331126 - 0.611836693781009*I, -0.94540233331126 + 0.611836693781009*I,
   0.154735144496843 - 1.03838075445846*I, 0.790667188814418 - 0.300506920309552*I,
```

```
   0.790667188814418 + 0.300506920309552*I, 0.154735144496843 + 1.03838075445846*I]
Python]
```

This example produces a very long output and will only be treated with `sympy`.

```
Python] integrate( p / q, x )

    RootSum(43531*_t**6 + 363*_t**4 - 1060*_t**3 + 27*_t**2 + 13*_t + 3, Lambda(_t,
   _t*log(19442177907668010*_t**5/10374601222607 - 1538552499493896*_t**4/10374601222607
   - 198788025618967*_t**3/10374601222607 - 429257255345079*_t**2/10374601222607 +
   5578472509755*_t/10374601222607 + x + 8754932267486/10374601222607)))
Python]
```

Here is our first integration algorithm for the logarithmic part. Note how we specially treat the case of quadratic resultants (i.e. of degree 2).

```
Python] def int_log_part(p, q, x):
            """
            Input: Polynomials p, q with deg(p) < deg(q) and q monic
            and square-free.
            Output: The integral ∫ p/q, provided the roots of q can be
            expressed in closed form (without the RootOf function).
            """
            # compute the resultant
            r = resultant(p - y*q.diff(x), q, x)
            r = r / content(r)

            # factor the resultant
            fr = factor_list(r)

            # add log terms
            integral = []
            for i in range(len(fr[1])):
                d = degree(fr[1][i][0], y)
                if d==1:
                    c = solve(fr[1][i][0], y)
                    v = gcd(p - c[0]*q.diff(x), q, x)
                    v = v / LC(v)
                    integral.append(c[0]*log(v))
                elif d==2:
                    # give answer in terms of radicals
                    c = solve(fr[1][i][0], y)
                    # compute the gcd in a field extension!!
                    v = [ simplify( gcd( p - c[j]*q.diff(x), q, x ) ) for j in range(d)]
                    for j in range(d):
                        integral.append(c[j]*log(v[j]))
                else:
                    # answer in terms of rootsum / rootof
                    c = solve(fr[1][i][0])
                    v = [ gcd(p - c[j]*q.diff(x), q, x, extension=c[j]) for j in range(d)]
                    for j in range(d):
                        integral.append(c[j]*log(v[j]))

            return sum(integral)
```

```
Python]
```

For our example from the Introduction we have

```
Python] p =  12*x**2 + 6
Python] q =  x**3 + x**2 + x + 1
Python] integrate(p / q, x)  ## sympy output

    9*log(x + 1) + 3*log(x**2 + 1)/2 - 3*atan(x)

Python] int_log_part(p, q, x)

    9*log(x + 1) + (3/2 + 3*I/2)*log(x - I) + (3/2 - 3*I/2)*log(x + I)

Python]
```

We repeat again that even though the answer above differs from the one obtained by `sympy` and the one in formula (8), stated in the Introduction, they are actually the same — just expressed differently to accomodate students of Calculus courses. To see this, just `rewrite` the term $3\cdot \operatorname{atan}(x)$ in terms of logarithms and `logcombine` the terms $\frac{3}{2}\log(x-i)+\frac{3}{2}\log(x+i)$.

```
Python] t = 3*atan(x).rewrite(log)
Python] t

    3*I*log((-I*x + 1)/(I*x + 1))/2

Python] expand_log(t, force=True)

    3*I*(log(-I*x + 1) - log(I*x + 1))/2

Python] (S(3)/2) * simplify( logcombine( log(x - I) + log(x + I), force=True) )

    3*log(x**2 + 1)/2

Python]
```

In summary, we can get the expressions to look the same, but that requires additional work and we will not follow that path.

4.2 Logarithmic Part with Subresultant PRS

In this section the algorithm presented for the integration of the logarithmic part makes use of subresultant polynomial remainder sequences in order to replace the *direct* computation of the resultant and to avoid time consuming gcd operations in extension fields.

Namely, the resultant of two polynomials $a(x)$, $b(x)$ is more efficiently calculated using the *subresultant polynomial remainder sequence* (prs), which in addition to giving the resultant of $a(x)$, $b(x)$, also gives a sequence of polynomials with some useful properties to be discussed below.

A subresultant polynomial remainder sequence is a generalization of the Euclidian algorithm where in each step, the remainder r_i is divided by a constant β_i. The Fundamental Theorem of prs's shows how to compute specific β_i such that the resultant can be calculated from the polynomials in the sequence.

It turns out that the polynomials $v_i = \gcd\left(p(x) - c_i \cdot q'(x), q(x)\right)$ encountered in the previous algorithm will appear in the subresultant prs of $p(x) - y\cdot q'(x)$ and $q(x)$. Furthermore, we can use the prs to immediately find the resultant $r = \operatorname{res}(p(x) - y\cdot q'(x), q(x))$, which as we saw, is all we need in order to compute the logarithmic part.

Here is the algorithm:

```
Python] def int_log_part_S(p, q, x):
            """
            Input: Polynomials p, q with deg(p) < deg(q) and q monic
            and square-free.
            Output: The integral ∫ p/q, provided the roots of q can be
            expressed in closed form (without the RootOf function).
            """
            # compute the subresultant prs
            sr = subresultants(p - y*q.diff(x), q, x)

            # form a deg:poly dictionary of the polys in sr
            degs = [Poly(sr[i], x).degree() for i in range(len(sr))]
            d = {degs[i]:sr[i] for i in range(len(sr))}

            # retrieve the resultant from the prs
            r = sr[-1]  ## the last element of the sequence
            r = r / content(r)

            # square-free factor r
            sqfr = sqf_list(r)[1]
            # print 'sqf_list resultant', sqfr

            # add log terms
            integral = []
            for i in range( len(sqfr) ):
                # normalize to make monic
                w = LC( d[ sqfr[i][1] ], x )
                s, t, _ = gcdex( w, sqfr[i][0] )
                ddi = rem( s*d[ sqfr[i][1] ], sqfr[i][0] , y)
                ddi = ddi / content( ddi, x )

                fr = factor_list(sqfr[i][0])[1]

                for j in range( len(fr) ):
                    degj = degree( fr[j][0], y )
                    if degj == 1:
                        c = solve( fr[j][0], y )
                        integral.append(c[0]*log(ddi.subs(y, c[0])))
                    elif degj == 2:
                        c = solve( fr[j][0], y )
                        for k in range( degj ):
                            integral.append(c[k]*log(ddi.subs(y, c[k])))
                    else:
                        c = solve( fr[j][0], y )
                        for k in range(degj):
                            integral.append(c[k]*log(simplify( ddi.subs(y, c[k]))))

            return simplify( sum(integral) )
Python]
Python] p = x**3 + 9*x**2 - 18*x + 9
```

```
Python] q = x**4 - 7*x**2 - 18
Python] int_log_part(p, q, x)

    21*log(x - 3)/22 - 39*log(x + 3)/22 + (10/11 - 9*sqrt(2)*I/44)*log(x - sqrt(2)*I) +
   (10/11 + 9*sqrt(2)*I/44)*log(x + sqrt(2)*I)

Python] int_log_part_S(p, q, x)

    21*log(x - 3)/22 - 39*log(x + 3)/22 + 10*log(x - sqrt(2)*I)/11 - 9*sqrt(2)*I*log(x -
   sqrt(2)*I)/44 + 10*log(x + sqrt(2)*I)/11 + 9*sqrt(2)*I*log(x + sqrt(2)*I)/44

Python] integrate( p/ q, x)

    21*log(x - 3)/22 - 39*log(x + 3)/22 + 10*log(x**2 + 2)/11 +
   9*sqrt(2)*atan(sqrt(2)*x/2)/22

Python]


Python] p = 1
Python] q = x**7 + 1
Python] int_log_part(p, q, x)

    log(x + 1)/7 + log(x + 7*RootOf(117649*y**6 + 16807*y**5 + 2401*y**4 + 343*y**3 +
   49*y**2 + 7*y + 1, 0))*RootOf(117649*y**6 + 16807*y**5 + 2401*y**4 + 343*y**3 +
   49*y**2 + 7*y + 1, 0) + log(x + 7*RootOf(117649*y**6 + 16807*y**5 + 2401*y**4 +
   343*y**3 + 49*y**2 + 7*y + 1, 1))*RootOf(117649*y**6 + 16807*y**5 + 2401*y**4 +
   343*y**3 + 49*y**2 + 7*y + 1, 1) + log(x + 7*RootOf(117649*y**6 + 16807*y**5 +
   2401*y**4 + 343*y**3 + 49*y**2 + 7*y + 1, 2))*RootOf(117649*y**6 + 16807*y**5 +
   2401*y**4 + 343*y**3 + 49*y**2 + 7*y + 1, 2) + log(x + 7*RootOf(117649*y**6 +
   16807*y**5 + 2401*y**4 + 343*y**3 + 49*y**2 + 7*y + 1, 3))*RootOf(117649*y**6 +
   16807*y**5 + 2401*y**4 + 343*y**3 + 49*y**2 + 7*y + 1, 3) + log(x +
   7*RootOf(117649*y**6 + 16807*y**5 + 2401*y**4 + 343*y**3 + 49*y**2 + 7*y + 1,
   4))*RootOf(117649*y**6 + 16807*y**5 + 2401*y**4 + 343*y**3 + 49*y**2 + 7*y + 1, 4) +
   log(x + 7*RootOf(117649*y**6 + 16807*y**5 + 2401*y**4 + 343*y**3 + 49*y**2 + 7*y + 1,
   5))*RootOf(117649*y**6 + 16807*y**5 + 2401*y**4 + 343*y**3 + 49*y**2 + 7*y + 1, 5)

Python] int_log_part_S(p, q, x)

    log(x + 1)/7 + log(x + 7*RootOf(117649*y**6 + 16807*y**5 + 2401*y**4 + 343*y**3 +
   49*y**2 + 7*y + 1, 0))*RootOf(117649*y**6 + 16807*y**5 + 2401*y**4 + 343*y**3 +
   49*y**2 + 7*y + 1, 0) + log(x + 7*RootOf(117649*y**6 + 16807*y**5 + 2401*y**4 +
   343*y**3 + 49*y**2 + 7*y + 1, 1))*RootOf(117649*y**6 + 16807*y**5 + 2401*y**4 +
   343*y**3 + 49*y**2 + 7*y + 1, 1) + log(x + 7*RootOf(117649*y**6 + 16807*y**5 +
   2401*y**4 + 343*y**3 + 49*y**2 + 7*y + 1, 2))*RootOf(117649*y**6 + 16807*y**5 +
   2401*y**4 + 343*y**3 + 49*y**2 + 7*y + 1, 2) + log(x + 7*RootOf(117649*y**6 +
   16807*y**5 + 2401*y**4 + 343*y**3 + 49*y**2 + 7*y + 1, 3))*RootOf(117649*y**6 +
   16807*y**5 + 2401*y**4 + 343*y**3 + 49*y**2 + 7*y + 1, 3) + log(x +
   7*RootOf(117649*y**6 + 16807*y**5 + 2401*y**4 + 343*y**3 + 49*y**2 + 7*y + 1,
   4))*RootOf(117649*y**6 + 16807*y**5 + 2401*y**4 + 343*y**3 + 49*y**2 + 7*y + 1, 4) +
   log(x + 7*RootOf(117649*y**6 + 16807*y**5 + 2401*y**4 + 343*y**3 + 49*y**2 + 7*y + 1,
   5))*RootOf(117649*y**6 + 16807*y**5 + 2401*y**4 + 343*y**3 + 49*y**2 + 7*y + 1, 5)

Python] integrate(p/q, x)

    log(x + 1)/7 + RootSum(117649*_t**6 + 16807*_t**5 + 2401*_t**4 + 343*_t**3 + 49*_t**2
   + 7*_t + 1, Lambda(_t, _t*log(7*_t + x)))

Python]
```

5 Examples

The next two rational functions have no logarithmic part:

```
Python] p = 8*x**5 - 10*x**4 + 5
Python] q = (2*x**5 - 10*x + 5)**2
Python] [rat_part, log_part] = int_rat_part_O(p, q, x)
Python] rat_part
    (-x + 1)/(2*x**5 - 10*x + 5)
Python] log_part
    0
Python] [rat_part, log_part] = int_rat_part_H(p, q, x)
Python] rat_part
    (-x + 1)/(2*x**5 - 10*x + 5)
Python] log_part
    0
Python]
Python] p = 4*x**5 - 1
Python] q = (x**5 + x + 1)**2
Python] [rat_part, log_part] = int_rat_part_O(p, q, x)
Python] rat_part
    -x/(x**5 + x + 1)
Python] log_part
    0
Python] [rat_part, log_part] = int_rat_part_H(p, q, x)
Python] rat_part
    -7*x/(7*x**5 + 7*x + 7)
Python] log_part
    0
Python]
```

The following example has both a polynomial part and a rational part:

```
Python] p = 441*x**7 + 780*x**6 - 2861*x**5 + 4085*x**4 + 7695*x**3 + 3713*x**2 - 43253*x
        + 24500
Python] q = 9*x**6 + 6*x**5 - 65*x**4 + 20*x**3 + 135*x**2 - 154*x + 49

Python] [poly_part, p] = int_poly_part(p, q, x)
Python] poly_part
    49*x**2/2 + 54*x
Python] p
    6615*x**4 + 3969*x**2 - 37338*x + 21854

Python] [rat_part, log_part] = int_rat_part_O(p, q, x)
```

```
Python] rat_part
    (-6615*x**3 + 6615*x**2 + 6762*x - 6272)/(9*x**4 - 6*x**3 - 36*x**2 + 54*x - 21)
Python] log_part
    0
Python] [rat_part, log_part] = int_rat_part_H(p, q, x)
Python] rat_part
    (-6615*x**3 + 6615*x**2 + 6762*x - 6272)/(9*x**4 - 6*x**3 - 36*x**2 + 54*x - 21)
Python] log_part
    0
Python]
```

Consider now the example:

```
Python] p = 36*x**6 + 126*x**5 + 183*x**4 + (13807/S(6))*x**3 - 407*x**2 - (3242/S(5))*x +
        3044/S(15)
Python] p
    36*x**6 + 126*x**5 + 183*x**4 + 13807*x**3/6 - 407*x**2 - 3242*x/5 + 3044/15
Python] q = (x**2 + S(7)/6*x + S(1)/3)**2 * (x - S(2)/5)**3
Python] q.expand()
    x**7 + 17*x**6/15 - 263*x**5/900 - 1349*x**4/2250 + 2*x**3/1125 + 124*x**2/1125 +
   4*x/1125 - 8/1125
Python] [rat_part, log_part] = int_rat_part_O(p, q, x)
Python] rat_part
    (158130*x**3 + 118641*x**2 - 186108*x + 42852)/(150*x**4 + 55*x**3 - 66*x**2 - 12*x +
   8)
Python] log_part
    (1080*x**2 + 35010*x + 53235)/(30*x**3 + 23*x**2 - 4*x - 4)
Python] [rat_part, log_part] = int_rat_part_H(p, q, x)
Python] rat_part
    (158130*x**3 + 118641*x**2 - 186108*x + 42852)/(150*x**4 + 55*x**3 - 66*x**2 - 12*x +
   8)
Python] log_part
    (17280*x**2 + 560160*x + 851760)/(480*x**3 + 368*x**2 - 64*x - 64)
Python] int_log_part( numer(log_part) , denom(log_part), x )
    37451*log(x - 2/5)/16 - 8000*log(x + 1/2) + 91125*log(x + 2/3)/16
Python] int_log_part_S( numer(log_part) , denom(log_part), x )
    37451*log(x - 2/5)/16 - 8000*log(x + 1/2) + 91125*log(x + 2/3)/16
Python]
Python] p = 36
Python] q = x**5 - 2*x**4 - 2*x**3 + 4*x**2 + x - 2
Python] [rat_part, log_part] = int_rat_part_O(p, q, x)
Python] rat_part
```

```
    (12*x + 6)/(x**2 - 1)
Python] log_part
    (12*x - 12)/(x**3 - 2*x**2 - x + 2)
Python] [rat_part, log_part] = int_rat_part_H(p, q, x)
Python] rat_part
    (12*x + 6)/(x**2 - 1)
Python] log_part
    12/(x**2 - x - 2)
Python] int_log_part( numer(log_part) , denom(log_part), x )
    4*log(x - 2) - 4*log(x + 1)
Python] int_log_part_S( numer(log_part) , denom(log_part), x )
    4*log(x - 2) - 4*log(x + 1)
Python] integrate(p / q, x)
    36*(2*x + 1)/(6*x**2 - 6) + 4*log(x - 2) - 4*log(x + 1)
Python]
Python] p = x
Python] q = x**2 + 2*x -3
Python] int_log_part(p, q, x)
    log(x - 1)/4 + 3*log(x + 3)/4
Python] int_log_part_S(p, q, x)
    log(x - 1)/4 + 3*log(x + 3)/4
Python] integrate(p /q, x)
    log(x - 1)/4 + 3*log(x + 3)/4
Python]
Python] p = 4*x - 1
Python] q = (x - 1) * (x + 2)
Python] int_log_part_S(p, q, x)
    log(x - 1) + 3*log(x + 2)
Python] integrate(p / q, x)
    log(x - 1) + 3*log(x + 2)
Python]
Python] p = x
Python] q = (x**2 + 1) * (x**2 + 2)
Python] int_log_part(p, q, x)
    log(x**2 + 1)/2 - log(x**2 + 2)/2
Python] logcombine( int_log_part(p, q, x), force=True )
    log(sqrt(x**2 + 1)/sqrt(x**2 + 2))
Python] int_log_part_S(p, q, x)
    log(x**2 + 1)/2 - log(x**2 + 2)/2
```

```
Python] integrate( p / q, x)
    log(x**2 + 1)/2 - log(x**2 + 2)/2
Python]
Python] p = x
Python] q = x**2 - 1
Python] int_log_part(p, q, x)
    log(x**2 - 1)/2
Python] int_log_part_S(p, q, x)
    log(x**2 - 1)/2
Python] integrate( p / q, x)
    log(x**2 - 1)/2
Python]
Python] p = 1
Python] q = (x - 1)**2 * (x + 4)
Python] [rat_part, log_part] = int_rat_part_0(p, q, x)
Python] rat_part
    -1/(5*x - 5)
Python] log_part
    -1/(5*x**2 + 15*x - 20)
Python] int_log_part(numer(log_part), denom(log_part), x)
    -log(x - 1)/25 + log(x + 4)/25
Python] int_log_part_S(numer(log_part), denom(log_part), x)
    -log(x - 1)/25 + log(x + 4)/25
Python] integrate( p / q, x)
    -log(x - 1)/25 + log(x + 4)/25 - 1/(5*x - 5)
Python]
Python] p = 2*x + 3
Python] q = (x + 1)**2
Python] [rat_part, log_part] = int_rat_part_0(p, q, x)
Python] rat_part
    -1/(x + 1)
Python] int_log_part(numer(log_part), denom(log_part), x)
    2*log(x + 1)
Python] integrate(p / q, [x, 0, 1])
    1/2 + 2*log(2)
Python]
Python] p = x**2 + 1
Python] q = x**2 - x
Python] integrate( p / q, x)
    x - log(x) + 2*log(x - 1)
```

```
Python] [poly_part, rat_part] = int_poly_part(p, q, x)
Python] poly_part
    x
Python] rat_part
    x + 1
Python] [rat_part, log_part] = int_rat_part_0(rat_part, q, x)
Python] rat_part
    0
Python] log_part
    (x + 1)/(x**2 - x)
Python] int_log_part(numer(log_part), denom(log_part), x)
    -log(x) + 2*log(x - 1)
Python]
Python] p = 1
Python] q = x**2 - a**2
Python] a = Symbol('a', positive=True)
Python] with assuming(Q.real(a), Q.is_true(a != 0)):
            print integrate(p / q, x )
   (log(-a + x)/2 - log(a + x)/2)/a
Python] int_log_part(p, q, x)
    log(-a + x)/(2*a) - log(a + x)/(2*a)
Python] with assuming(Q.real(a), Q.is_true(a != 0)):
            print int_log_part_S(p, q, x)
   Traceback (most recent call last):
   ValueError: univariate polynomial expected
Python]
Python]
```