

# Revisiting numeric/symbolic indefinite integration of rational functions, and extensions

Richard Fateman

October 15, 2008

## Abstract

We know we can solve this problem: Given any rational function  $f(x) = p(x)/q(x)$ , where  $p$  and  $q$  are univariate polynomials over the rationals, compute its *indefinite* integral, using if necessary, algebraic numbers. But in many circumstances an approximate result is more likely to be of use. Furthermore, it is plausible that it would be more useful to solve the problem to allow definite integration, or introduce additional parameters so that we can solve multiple definite integrations. How can a computer algebra system best answer the more useful questions? Finally, what if the integrand is not a ratio of polynomials, but something more challenging?

Consider a question of the form “Please give me  $F$  the indefinite integral of  $f(x)$ , a mathematical function of  $x$ .” There is a belief among advocates of computer algebra systems, as well as among teachers of calculus, that there is an advantage to providing  $F$  as a formula, especially when the formula is expressible in terms of “elementary” functions. We suspect that many calculus teachers believe that the integration of  $f(x)$  when  $f$  is *rational* can be done algorithmically by methods given in every calculus book, in spite of the fact that these methods work only for sufficiently low-degree denominators or other special factorable cases.

A possibly more useful answer for many users of computer systems for mathematics, is a computational procedure  $F(a, b)$  — perhaps a very fast subroutine compiled to machine language — which for any particular pair of numerical arguments  $a$  and  $b$  returns a number for the integral from  $a$  to  $b$ . The internal structure of  $F$  could be opaque, but the conventional technique would repeatedly call a subroutine that computes the value of  $f(x)$ , and the program  $F(a, b)$  would use some kind of quadrature program. There are dozens of well-known ones as well as procedures for choosing amongst them. <http://www.netlib.org/quadpack/>. See also [4] for implementations of some programs that gain some versatility (as they likely lose in speed) by implementation in a computer algebra system.

Alternatively the program  $F(a, b)$  might be a program that evaluates an algebraic expression, “the exact indefinite integral” at  $a$  and  $b$ , and computes the difference, relying on the Fundamental Theorem of Integral Calculus and the continuity of  $f$ . Or perhaps it evaluates the result of expanding  $f$  as a Taylor series and integrating term by term.

The numerical approach from pure quadrature is very powerful and well-supported by carefully constructed subroutine libraries. Our own minor contribution in this area, the implementation in arbitrary precision floating-point of some simple tools for integration, is described elsewhere [4]. For the remainder of this paper let us assume that for some reason (speed, accuracy, compactness, presence of parameters), that the quadrature route is not appropriate.

That is, we wish to somehow create the black-box result  $F(a, b)$  by looking at the symbolic representation of  $f$ . In particular we will start with a univariate rational function  $f(x)$ .

## 1 Integrate $f(x) = p(x)/q(x)$ semi-symbolically

Force  $q$  to be monic, if necessary by multiplying  $p$  by a suitable factor.

Find the (usually approximate) complex numeric factors of  $g$ , and their multiplicities,  $\prod_i (x - r_i)^{m_i}$ , and from this form a partial fraction expansion of the expression. This in turn can be integrated trivially. (A 20 line program in Macsyma, attached below, can do all this).

Question 1: Don't we lose something important by using approximate complex numbers for representation of the roots of the denominator?

Answer 1: Perhaps, but an exact closed-form representation of the roots has problems too.

- Once you realize that the neat examples shown in a typical textbook never involve something worse than a quadratic polynomial, your faith in exact representations in the real world should fade. If the world were as simple as the textbook examples, we would never need polynomial rootfinders.
- Speaking exactly, you can't rely on the roots being expressible except implicitly as "the roots of" some polynomial. The complex floating-point numbers look rather better if you use some constraint to limit the number of digits displayed, though in computations you want to have as much precision as can be justified by the accuracy of the numerical rootfinder.
- Even when the roots *are* technically expressible in algebraic surd notation, the usefulness of the result in further computations, except in the very simplest examples, is debatable. The explicit cubic or quartic formulas are so unwieldy that having provided them, the computer algebra system is rather hard-pressed to carry on significant additional steps beyond substituting back into the polynomial and —after significant effort— confirming the solution. In other computations using these roots to pursue an exact formula becomes ever less likely to look like a desired end result for a human consumer of mathematical results.
- Programs to find root approximations to any, (even absurdly high), accuracy, can be found. A program that promises to find *any finite number of additional digits* can be easily constructed if you have a programming environment supporting arbitrary precision numbers. Such environments exist in computer algebra systems as well as independent libraries such as GMP, MPFR, and ARPREC.
- Problems that originate in the natural world, as opposed to the abstract exact world of pure mathematics, tend to have coefficients of limited precision and accuracy. Treating such approximations as exact values and running expensive algorithms to propagate the imputed accuracy to find roots is rather more expensive than necessary in relation to the benefit (of identifying dubious digits). Few real-world measurements can be obtained to better than double-precision accuracy.

Question 2: Instead of using rootfinding, wouldn't it be better to factor the denominator over the rationals first?

Answer 2: Yes, an exact partial-fraction decomposition (typically one command in a computer algebra system) as a preliminary stage would make this integration result even better. Even a square-free decomposition of the denominator would be helpful. It presumes that such pricey exact operations are acceptable expenses even if their chances of success are negligible. These computations can be done fairly routinely through a combination of pretty good algorithms and very fast computers. They can work even if the denominator or some factors of the denominator have symbolic parameters other than the variable of integration.

## 2 A program

Here's the Macsyma program we promised, to do "floating point" partial fraction expansion and rational function integration. This was published in Proc. ISSAC'82, "Computer Algebra and Numerical Integration," by R. Fateman p 228 - 232.

For brevity we have removed the comments. A chattier version is in an appendix.

```

fpint(p,x):=
  block ([num, den,div, res,keepfloat:true,lroots,distinctroots,qq, L, c],
    local(r,q),
    p:rat(p), num:ratnumer(p), den:ratdenom(p), div: divide(num,den),
    res: integrate(ratdisrep(div[1]),x),
    num: ratdisrep(div[2]/ratcoef(den,x^hipow(den,x))),
    lroots:map(rhs,allroots(den)),
    r[i]:=0, for c in lroots do r[c]:r[c]+1,
    distinctroots: map(first,rest(arrayinfo(r),2)),
    qq:1,
    for c in distinctroots do qq:qq*(x-c)^r[c],
    for c in distinctroots do q[c]:qq/(x-c)^r[c],
    for c in distinctroots do for i: 1 thru r[c] do
      res: res+1/(r[c]-i)!
      *ratsimp(subst(c,x,diff(num/q[c],x,r[c]-i)))
      *(if i=1 then log(x-c) else (x-c)^(1-i)/(1-i)),
    return(res));

```

As mentioned earlier, if we wish to argue that the input to this process is exact and therefore we can justify some exact symbolic computations to help this integration along, we can try to compute a partial fraction decomposition first, and then integrate term by term. In this case we may get integrands that look like  $A/B^n$  for integer  $n > 1$  and a polynomial  $B$  of degree greater than one. Computing the partial fraction expansion requires about seven additional lines of Macsyma code. (Though we had to define a more general version of the extended GCD program than was built-in to Macsyma.)

```

/*integrate num/denom^n where
  num and denom are polynomials in x with coefficients
  over the integers, or rationals, or for that matter, rational
  functions in parameters.
  gcd(num,denom)=1,
  denom is square-free,
  n is required to be a particular positive integer. A minor variation
  on this program, namely putting a quote mark before the recursive
  call to i_ab2n, takes just one step and reduces a "symbol" n by one.
*/

```

```

i_ab2n(num,denom,n,x):= /*integrate a/b^n */
if n=1 then integrate(num/denom,x) else
block([r,s,g,u,v, ddenom: diff(denom,x)],
  [r,s,g]:extended_gcd(ddenom,denom,x),
  r:r/g, s:s/g, /* r*ddenom+s*denom=1 */
/* We rearrange the expression num/denom^n =
  num*r*ddenom/denom^n + num*s/denom^(n-1).
  The first term can be (partially) integrated by parts;
  the second term, and the rest of the first term are
  combined, and poured back into this program, since they
  have the order of denominators reduced by one */

u: num*r, v:denom^(-n+1)/(-n+1),

```

```

/* u*v -integral(v du)+integral(num*s/denom^(n-1),x) =
   u*v - 1/(-n+1)* i_ab2n(diff(u,x),denom,n-1,x)
           +i_ab2n(num*s,denom,n-1,x)= ...*/
return( u*v + i_ab2n(1/(n-1)*diff(u,x)+num*s,denom,n-1,x)))$

/* We used this subroutine, almost the same as ext_gcd in Macsyma, but
ours is more general. Using Knuth's notation alg 4.5.2X p 325 of vol
2, Art of Computer Programming 2nd ed. ...Find u1, u2, u3 such that
u1*u+u2*v=u3, with suitable degree bounds for u1,u2,u3 as polynomials
in x.*/

extended_gcd(u,v,x):=
block([u1,u2,u3,v1,v2,v3,t1,t2,t3],
      u: rat(u,x), v: rat(v,x), [u1,u2,u3]:[rat(1),rat(0),u],
      [v1,v2,v3]:[rat(0),rat(1),v],
      while v3#0 do
        (q: quotient(u3,v3,x),
         [t1,t2,t3]:[u1,u2,u3]-q*[v1,v2,v3],
         [u1,u2,u3]:[v1,v2,v3],[v1,v2,v3]:[t1,t2,t3]),
      [u1,u2,u3])$

```

### 3 Parameters in the denominator

This approach described above breaks down if  $q$  has symbolic coefficients, since rootfinding is then problematic. Finding algebraic *functions*  $r_i$  in closed form may not be possible. (We have noted that if only the numerator  $p$  has symbolic coefficients, this is not a problem.)

What can we do in this case?

Let us eliminate items in  $q$  that represent multiple roots, i.e. now assume  $m_i = 1$  for all  $i$ , by squarefree factorization, done symbolically, since we know how to reduce to this case. Let us also make the (false) assumption that for the remaining roots we can compute, or at least designate, a set of *distinct* roots. We can, in this case, name the roots and designate a sum over them, hoping that at some future time if parameters are resolved the roots can be found numerically. This is often the approach taken by computer algebra programs, since the computation can be tossed off as a resultant calculation.

If the assumption about distinct roots above is false, it has consequences. Consider the indefinite integral of  $1/(1 + 5x + ax^2 + 10x^3 + 5x^4 + x^5)$  which standard CAS can express as a logarithmic expression summed over a set of roots of a polynomial. (“rootsum” in some systems) With the substitution  $a = 10$ , the correct integral looks like  $-1/4(1 + x)^4$ , and has *no logarithms*. Substituting  $a = 10$  into the result of the previous method does not produce a correct integral.

There remain several possibilities for further progress. One is to use “absolute polynomial factorization” which has been a popular subject for researchers recently. This sometimes leads to an approximate factorization (whose definition needs to be clarified) in several variables. Unfortunately, in spite of recent progress on approaching the problem with asymptotically faster methods, it appears that no one has programmed the predictably messy routines for incorporation in a general CAS. The intent would be to find multivariate near-factors for the denominator.

The second method we are aware of is to find algebraic approximate expansions for the denominator’s factors. In this case we have methods based on Newton-Puiseux expansion for finding roots. As an example,

we use `taylor_solve`, an implementation of this idea in `Macsyma`. Assume that the denominator is

$$x^5 + a x^3 + b = 0,$$

an irreducible polynomial in  $x$  with *TWO* extra parameters. We have some choices to make, but let us assert that we wish to find factors  $(x - r_i)$  where  $r_i$  is a series in the parameter  $a$ , expanded around a point which could be arbitrary, but let us choose for our illustration to be 0. This allows us to deal with solutions for arbitrary “small”  $a$  as series. To fourth order, `Macsyma` produces

$$\left\{ \left\{ x = k0 - \frac{a}{5k0} + \frac{a^2}{25k0^3} + \frac{2a^4}{625bk0^2} + \dots, k0^5 = -b \right\} \right\}.$$

This says that we can run through the solutions of  $k0^5 = -b$ , and for each of the 5 solutions, say  $k$  produce a particular approximate factor

$$x - k + \frac{a}{5k} - \frac{2a^4}{625bk^2} - \frac{a^2}{25k^3} + \dots$$

Instead of summing over exact roots of  $x^5 + a x^3 + b = 0$ , we are summing an approximate symbolic result over roots of  $k^5 = -b$ . Whether this is closer to the desired form of the solution depends on the intended uses for the result.

Rabinowitz [11] points out that if an integrand can be factored so that the parameter occurs in only one factor, some numerical quadrature methods can benefit more than others from such an arrangement. That is, if the integrand is of the form  $f(x)g(x, \alpha)$  and if the values of  $f(x)$  at various abscissae can be cached, a non-adaptive numeric method may be cheaper since it need recompute only  $g(x, \alpha)$  at those fixed abscissae. If the integration method is adaptive, the benefit may be more elusive.

## 4 Almost-rational integrals

Consider integrating  $1/(ax^2 + bx + c)^n$ . This is an uncomfortable problem which `Mathematica` resolves by returning a hypergeometric  ${}_2F_1$  function. After some effort this can then be “simplified” to an expression using an incomplete Beta function.

$$- \left( \frac{\sqrt{b^2 - 4ac} \left( - \left( \frac{a(c+x(b+ax))}{b^2 - 4ac} \right) \right)^n \beta\left(\frac{1}{2} - \frac{b+2ax}{2\sqrt{b^2 - 4ac}}, 1 - n, 1 - n\right)}{a(c+x(b+ax))^n} \right)$$

This function seems to have some of the appropriate properties; yet the complexities of simplification of hypergeometric functions seem to cause new problems. It is not clear what properties it might be missing. The “unintegrated” integral is, we suspect, preferable for many purposes. It is relatively small, its derivative is easily found, and its numerical evaluation by quadrature as a function of  $a$ ,  $b$ ,  $c$  and  $n$  may be less onerous than evaluating the special functions in the displayed “answer.” An alternative indicated above is to reduce “symbolically” the given integral by expressing it as a function of a reduced integral, namely of  $1/(ax^2 + bx + c)^{n-1}$ . This is done by executing one step in the integration process. `Macsyma` reduces this particular problem, for  $n > 1$  to

$$\frac{2ax + b}{(4ac - b^2)(n-1)(ax^2 + bx + c)^{n-1}} + \frac{(4an - 6a)}{(4ac - b^2)n - 4ac + b^2} \int \frac{1}{(ax^2 + bx + c)^{n-1}} dx.$$

In combination with the formula for an integrand with a numerator of  $x$ , that is, the integral of  $x/(x^2 + bx + c)^n$  where we have, without loss of generality, made the denominator monic, we find forms for arbitrary

power-of-quadratic denominator:

$$\frac{x(2x+b)}{(4c-b^2)(n-1)(x^2+bx+c)^{n-1}} + \frac{1}{(4c-b^2)(n-1)} \int \frac{(4n-8)x-b}{(x^2+bx+c)^{n-1}} dx$$

It starts to get bulky pretty soon, but the integration of  $x^k/(x^3+ax^2+bx+c)^n$  has an initial rational part of

$$\frac{x^k(6bx^2-2a^2x^2-9cx+7abx-2a^3x-3ac+4b^2-a^2b)}{(27c^2-18abc+4a^3c+4b^3-a^2b^2)(n-1)(x^3+ax^2+bx+c)^{n-1}}$$

plus an integral reduction (which may have more rational parts) to

$$\int \frac{x^{k-1}(((18b-6a^2)n+(2a^2-6b)k-30b+10a^2)x^2+((-27c+15ab-4a^3)n+(9c-7ab+2a^3)k+36c-22ab+6a^3)x+(3ac-4b^2+a^2b)k)}{(27c^2-18abc+4a^3c+4b^3-a^2b^2)(n-1)(x^3+ax^2+bx+c)^{n-1}} dx$$

Clearly additional formulas can be generated, although in our opinion current CAS do not really address the issue of how worthwhile such formulas are, compared to their generating methods. Certainly it seems implausible for most purposes to generate enormously complex higher-degree formulas with completely symbolic coefficients only to subsequently evaluate the formula at one particular set of coefficient values.

This approach of symbolic reduction, which in the case of univariate polynomials eventually leads to the simple case of our first program, should fit somewhere into the armamentum of symbolic/numeric integration. How to best make use of this is not clear. The hypothetical target audience for integration (mostly calculus students solving homework problems) is not interested in relatively hard problems in the first place, and so we can hardly guess what might be of most use to the widest audience. We do note that simple versions of such information can be found in integrals tabulated and published for human use, and so the interest predates the computer algebra systems.

## 5 Another stab at the problem

As indicated above, any decomposition of the denominator into linear factors can be used in the `fpint` program. A factorization into a product using, if necessary, named roots  $\{r_1, r_2, \dots\}$  will lead to a formal answer. The assumption here is that this will be necessary when the denominator has parameters that enter into the problem in an essential way. While representing the result in terms of these roots, we should apply the simplifications that are inherent in symmetric expressions of the roots (e.g. the product  $r_1 \cdot r_2 \cdot \dots$  is the constant coefficient of the denominator).

This gets sufficiently messy, sufficiently rapidly, that one would have to seriously consider returning to our black-box program using quadrature that includes a way of feeding in extra parameters.

## 6 Display

Consider the problem of displaying the integral of  $R = x^5 + x + 3^{-1}$ . The denominator cannot be factored, and so the (by now) “traditional” symbolic version of the result is to produce a summation over the zeros of the denominator of  $R$ . Macsyma 2.4<sup>1</sup> says,

$$\text{root\_sum}(r \log(16875x - 16216384r^4 + 4054096r^3 - 1003284r^2 + 255941r + 256), 253381)$$

<sup>1</sup>We have taken some liberties with the displays in order to get them to fit on the page here.

where

$$r^5 - 160r^3 - 80r^2 - 15r - 1 = 0.$$

Maple 7 produces this version

$$\sum_{R=\text{RootOf}(253381\_Z^5-160\_Z^3-80\_Z^2-15\_Z-1)} R \ln \left( x - \frac{16216384}{16875} R^4 + \frac{4054096}{16875} R^3 - \frac{111476}{1875} R^2 + \frac{255941}{16875} R + \frac{256}{16875} \right).$$

Mathematica 5.0 produces a somewhat different form (and also does not find the same minimal polynomial as the previous two programs):

`RootSum[3 + #1 + #1^5 & , Log[x - #1]/(1 + 5*#1^4) & ]`.

Compare this to the form produced by Macsyma from the floating-point integration program:

$$\begin{aligned} & (0.08827i + 0.00584) \log(x + 1.12969i + 0.47538) + (0.03263i - 0.05996) \log(x + 0.82287i - 1.04188) \\ & + (-0.03263i - 0.05996) \log(x - 0.82287i - 1.04188) + (0.00584 - 0.08827i) \log(x - 1.12969i + 0.47538) \\ & + 0.10824 \log(x + 1.13298) \end{aligned}$$

This last version has the major benefit in terms of numerical usage of showing the roots explicitly. It (or a higher-precision version) can be slightly massaged to conform to the syntax of the Fortran or C language, and then can be emitted directly into the middle of a piece of code.

For symbolic computation, having only finite precision means that we have injected some perturbation into the computation stream. For example, if we differentiate the single-precision expression above, we wander off-track in the following way. The result, after collecting the expression over a common denominator, is

$$\begin{aligned} & -5.21542 \cdot 10^{-8} x^4 - 8.94071 \cdot 10^{-8} x^3 + (2.98024 \cdot 10^{-7} - 2.38419 \cdot 10^{-7} i) x^2 \\ & + 1.78814 \cdot 10^{-7} i x - 5.96046 \cdot 10^{-8} i - 1.0 / \\ & 1.0 x^5 - 5.96046 \cdot 10^{-7} x^4 + 4.76838 \cdot 10^{-7} x^3 \\ & + (2.62261 \cdot 10^{-6} i + 4.76838 \cdot 10^{-7}) x^2 + (1.00004 - 9.53675 \cdot 10^{-7} i) x + 2.9999 \end{aligned}$$

Equating such a result to the original input requires that we systematically chop “small” quantities to zero, to produce

$$\frac{1.0}{1.0x^5 + 1.00004x + 2.9999}$$

We might then replace near-integer floating-point numbers like 1.00004 by their exact integer counterparts. These transformations are not foolproof. The unease experienced by looking at such expressions with many floating-point numbers embedded in them is easily remedied: don’t look at them unnecessarily!

Another variation is to express the result as a very simple formula:

$$\sum_{i=0}^5 a_i \log(x - b_i)$$

which could clearly be implemented in C or Fortran. This would be augmented by two suitably defined constant arrays of complex numbers,  $\{a_i\}$  and  $\{b_i\}$ . In general the summation form for the result will have to include rational and polynomial parts as well, perhaps using Horner’s rule for the evaluation of those parts.

## 7 Other directions

We have written separately on related topics which represent other directions in making the integration programs in computer algebra systems more successful. These include (a) identifying symmetries of integrands which simplify computations; (b) factoring out oscillatory functions so that more accurate and efficient quadrature programs (e.g. Filon-like) can be used, and (c) in conjunction with the previous item, detecting when an expression has the characteristic of being a product of oscillatory and slowly-varying functions. [3, 6, 5].

## 8 Previous work

We've written on this topic, as have others, some of whom are referenced below. For (far more) information on symbolic integration see the monograph by M. Bronstein [1]. For experience in the development of the Mathematica definite integration program see the paper by D. Lichtblau [10].

## 9 Conclusion

CAS integration programs tend to be collections of algorithms with a patchwork of control structure navigating among them. Unfortunately, most systems don't really try to find out what the user would prefer for an answer. Our guess is that it is highly unlikely that a user will actually be interested in an expression involving algebraic "root-of" components. This paper shows how to provide an alternative especially if there are no parameters in the denominator of a rational integrand. This paper also makes some suggestions even if there are a few parameters.

## References

- [1] M. Bronstein, *Symbolic Integration-I*, Springer-Verlag, Heidelberg, 1997.
- [2] R. Fateman, "Computer Algebra and Numerical Integration," Proc. SYMSAC-81, ACM p. 228-232.
- [3] R. Fateman, Methods for integration of (anti)-symmetric functions, draft, 2003.
- [4] R. Fateman, Numerical quadrature in a symbolic/numeric setting, (DRAFT) <http://www.cs.berkeley.edu/~fateman/papers/quad.pdf>.
- [5] R. Fateman, Numerical Integration for Oscillatory Integrands: a Computer Algebra Perspective, 2008 <http://www.cs.berkeley.edu/~fateman/papers/oscillate.pdf>.
- [6] R. Fateman, When is a Function Oscillatory? <http://www.cs.berkeley.edu/~fateman/papers/oscfun.pdf>.
- [7] K.O. Geddes and G.J Fee, "Hybrid symbolic-numeric integration in Maple." Proc. ISSAC'92 p. 36-41.
- [8] M.T. Noda, E. Miyahiro, "On the Symbolic/Numeric Hybrid Integration". Proc. ISSAC'90 p. 304.
- [9] James H. Davenport, "Symbolic and Numeric Manipulation of Integrals." in *Accurate Scientific Computations 1985*: 168-180.
- [10] Daniel Lichtblau, "Issues in symbolic definite integration", Int'l Conf. on Applications of Computer Algebra (ACA 2003), Session of Symbolic Summation (July, 2003).
- [11] P. Rabinowitz, "Automatic Integration of a Function with a Parameter", CACM, vol 9 no. 11, Nov, 1966: 804-807.

## 10 Appendix: A commented version of fpint

```
fpint(p,x):= /* floating-point rational function integration */

    block ([num, den,div, res,keepfloat:true,roots,
           distinctroots,qq, L, c],
    local(r,q),
/* Simplify the input */
    p:rat(p), num:ratnumer(p), den:ratdenom(p),
/* separate polynomial and rational parts */
    div: divide(num,den),
/* Integrate polynomial if any, and initialize result */
    res: integrate(ratdisrep(div[1]),x),
/* Set num to numerator/leading coefficient of denominator */
    num: ratdisrep(div[2]/ratcoef(den,x^hipow(den,x))),

/* Call the Macsyma polynomial zero-finder; get a list of roots. */
    roots:map(rhs,allroots(den)),
/* The number of roots at c, a point in the complex plane
is stored in r[c] */
    r[i]:=0,
    for c in roots do r[c]:r[c]+1,
/*distinctroots is set to a list of the roots [c1 ,c2 ,... ].
computed by extracting info from Macsyma representation. */
    distinctroots: map(first,rest(arrayinfo(r),2)),
/* qq is a reconstruction of the denominator, factored */
    qq:1,
    for c in distinctroots do qq:qq*(x-c)^r[c],
/* qq[c] is qq with the r[c] roots at c taken out */
    for c in distinctroots do q[c]:qq/(x-c)^r[c],

/* The formula below gives the (constant) residue
needed in the numerators in the partial fraction expansion.
Integration of expressions of the form K/(x-c)^i is easy.
Note that r[c] is likely to be 1
    [the case of isolated roots of denominator]. */
    for c in distinctroots do for i: 1 thru r[c] do
res: res+1/(r[c]-i)!
    *ratsimp(subst(c,x,diff(num/q[c],x,r[c]-i)))
    *(if i=1 then log(x-c) else (x-c)^(1-i)/(1-i)),
return(res));
```

This code has been lightly modified from the version of 1992 to work in commercial Macsyma 2.4. Limitations: the denominator of the rational function must be a univariate polynomial of some degree  $d$  in the given variable, all of whose coefficients can be represented in floating-point, and where the rootfinding program (Macsyma has a few) must be sufficient to find all  $d$  roots to an accuracy that is satisfactory for the rest of the processing. The result may be complicated to view because the many digits of the roots, especially those that are complex numbers, take space on the page. Also we do not make any effort to treat complex

conjugate pairs of roots in a way that would simplify the output form, though that can be added by an appropriate variation of the rootfinder and corresponding changes to our program.

This program has some surprising extras (good features): The numerator as well as the polynomial part of the expression may have additional arbitrary parameters. E.g.  $(a * x + b)/(x^3 + x + 1) + c * x^4$  can be integrated, even though it contains non-numeric parameters a, b, c that would prevent one from using a numerical quadrature program.