Improving Exact Integrals From Symbolic Algebra Systems

Richard J. Fateman and W. Kahan
University of California, Berkeley

July 18, 2000

Abstract

Programs in symbolic algebraic manipulation systems can compute certain classes of symbolic indefinite integrals in closed form. Although these answers are ordinarily formally correct algebraic anti-derivatives, their form is often unsuitable for further numerical or even analytical processing. In particular, we address cases in which such "exact answers" when numerically evaluated may give less-accurate answers than numerical approximations from first principles! The symbolic formulas may also behave inappropriately near singularities. We discuss techniques, based in part on the calculus of divided differences, for improving the form of results of symbolic mathematics systems. In particular, computer algebra systems must take explicit account of the possibility that they are producing not "mathematics" but templates of programs consisting of sequences of arithmetic operations. In brief, mathematical correctness is not enough. Forms produced by rational integration programs are used for examples.

Introduction

The naive approach to the calculation of a definite integral $\int_x^y f(z)dz$ entails two steps: the first is the construction of the symbolic anti-derivative, $F(z) := \int f(z)dz$, and the second is the evaluation of $F(y) - F(x)$. This approach is feasible only for very special integrands $f(z)$, those for which $F(z)$ exists in *closed form.* For most $f(z)$, the integral must be approximated either by *numerical quadrature*, a process based upon weighted averaging of numerical samples of $f(z)$, or else by some kind of series expansion. Therefore the special case when a primitive F exists in closed form offers some hope that the integral can be calculated exactly, or at least more accurately than if an approximate method had to be chosen from the start.

In fact the ostensibly exact methods, including those used by symbolic integration programs (as in, for example, Mathematica, Maple, Macsyma, Axiom Reduce, Mupad etc. See [2] p. 221-243 for a survey), *may* produce answers, which, when ultimately reduced to numeric values, are *less* accurate than the approximate methods.

We assume from this point on that the formula for $F(z)$ will be evaluated in floating-point rather than exact (rational) arithmetic. Two hazards can degrade accuracy First, the formula for $F(z)$ may suffer from numerical instability unless it is rearranged in a way that takes account of the range of values intended for its argument z. Second, when the limits of integration $z = x$ and $z = y$ are relatively close, $F(y) - F(x)$ may mostly cancel off, leaving little more than rounding errors after the subtraction.

There is a third hazard that is independent of accuracy: sometimes $F(z)$ is so much more complicated than $f(z)$ that a quadrature program is preferable to direct evaluation. One example[9] is $f(x) = 1/(1 + z^{64})$ whose integral is

$$F(z) = \frac{1}{32}\sum_{k=1}^{16} c_k \operatorname{arctanh}\left(\frac{2c_k}{z + 1/z}\right) - s_k \arctan\left(\frac{2s_k}{z - 1/z}\right)$$

where $c_k := \cos((2k-1)\pi/64)$ and $s_k := \sin((2k-1)\pi/64)$.

Nothing much can be done about this third hazard, and it is unfortunate that symbolic integration programs may unwittingly assist in the production of even more expensive monstrosities.

The second hazard, and often the first too, can be mitigated with the aid of systematic techniques drawn from the Calculus of Finite Differences; see [15]. This calculus manipulates *divided differences* like

$$\triangle F([x, y]) := \begin{cases} (F(y) - F(x))/(y - x) & \text{if } x \neq y \\ F'(x) & \text{if } x = y \end{cases}$$

according to rules that resemble the chain rule, product and quotient rules, inverse and implicit function rules, etc., familiar in the calculus of derivatives. The resemblance is most striking when the argument z of $F(z)$ is a scalar, not a vector. Scalar divided differences, like derivatives, operate upon functions without raising their level of transcendence; the divided difference of a polynomial is a polynomial, of a rational function is rational, of an algebraic function is algebraic, of an elementary transcendental function is elementary, albeit with one more argument than before. In view of that resemblance, it may seem strange that computerized symbolic algebra systems, already so adept at simplification, differentiation and even integration, do not cater to divided differences. There is a reason for this.

Although the derivative of a function is not often much more complicated than that function, the divided difference is usually at least about twice as complicated and often much worse. Consequently, explicit divided differences reward human inspection far less often than derivatives do. Therefore there is little incentive to transform *displayed expressions* into their divided differences. Instead, divided differences pay off best when they are *implicit in algorithms* which then can operate upon numbers or expressions unseen by human eyes.

In other words, divided differences may well be thought of as transformations performed during a code-generation phase by computerized symbolic calculations, upon the *symbolic representation of a program*, to enhance its numerical stability. That is why programs to compute $F(y) - F(x)$ accurately might be expected to take special forms derived from the calculus of divided differences even though explicit divided difference expressions never appear.

The application of divided differencing to the evaluation of integrals from explicit symbolic formulas produced by computer algebra systems is therefore natural, and can enhance accuracy substantially without much extra computation.

In what follows, we present examples illustrating how the divided difference calculus succeeds in transforming numerically unsatisfactory expressions for definite integrals into algebraically equivalent but numerically satisfactory expressions and algorithms. The intent of the examples is to stimulate appreciation for and further study of that calculus; a detailed treatment of the subject must be sought elsewhere.

We do not provide a complete algorithm for the resolution of all accuracy problems that can result from integration. In fact, forms for divided differences of higher functions (even trigonometric forms or rational functions of vector arguments) require the introduction of classes of new non-elementary functions defined by integrals.

The techniques shown below work for integrals of polynomial and rational functions, and can be generalized in particular instances, to larger classes. A "universal" automated resolution is not in sight.

Integration Programs

To be specific in this paper we will present programs fragments in the language of the Macsyma computer algebra system (CAS) although other systems mentioned in the introduction are approximately equivalent for this purpose and could be used instead.

What is Provided by a CAS?

In Macsyma, the command `integrate(`$f(x), x$`)` invokes an algorithm which is intended to deliver the integral in closed form. That is, it is designed to produce an anti-derivative $F(x)$ of the function $f(x)$, or some indication of "failure". The computed anti-derivative $F(x)$ has the property that computing its derivative brings us back to $f(x)$ or something which can be simplified (often, but not always, by the CAS) to $f(x)$.

The definite integration program is invoked in a similar fashion, by the `integrate` command, but with two additional arguments, the lower and upper bounds. That is, `integrate(`$f(x), x, a, b$`)` returns an expression which should be the moral equivalent of $F(b) - F(a)$ in the usual circumstances. In some cases this simple formula does not work, but Macsyma is able to do the integral by contour integration or other methods. In the case of definite integration, failure is signalled by responses like the return of the requested integral unaltered in form, or the message "divergent."

What is Needed?

Unless the answer is very simple, more processing will be needed. Many users will proceed to put the formula from the CAS through additional steps. It is possible that repeated evaluation of some sort will be needed for plotting, or for approximation of higher-order integrals. For such purposes we would like to see not $F(x)$, a *mathematical formula*, but a *computer function* `Integral_of_f(a,b)`, an efficient and accurate value-returning procedure of two numerical arguments that provides an approximation to $F(b) - F(a)$ or its moral equivalent. A successful program `Integral_of_f` avoids disastrous numerical errors such as division by zero, minimizes loss of accuracy to cancellation, and promotes an efficient order of calculation.

There have been several papers recently on the combined use of symbolic and numeric processing [3, 4, 5, 7]. Two papers ([4, 5]) provide a bare-bones numeric/symbolic integrator for rational functions of a single variable. Such a program, using hardly more mechanism than an accurate polynomial zero-finder and simple arithmetic on polynomials, produces a reasonable approximate-algebraic answer: a rational function plus logarithms using floating-point approximations. In some ways it is more powerful than Macsyma's built-in integration program because it is not stymied by irreducible denominators of too high a degree. However, the answer is a poor *computer function*, even after mechanical conversion to FORTRAN, because no particular attention has been paid to accuracy or efficiency.

Demonstrating the formation of better `Integral_of_f` functions with the aid of divided differences is the primary objective here; in the appendix we show that a number of additional tasks must be discharged to completely analyze an integral.

Integration of Rational Functions

As is well known, indefinite integrals of rational functions are expressible as sums of polynomials, rational functions, logarithms of rational functions and perhaps arctangents, using as a base field algebraic number extensions.

We will assume that any algebraic numbers which appear in our problem can be approximated satisfactorily by floating point numbers, although this necessity does not arise in our examples here.

We trace the problems through a sequence of what appear to be trivial examples. They illustrate the problems quite well, however, and can be generalized without difficulty to cover all of rational function integrations.

Example 1:

Consider

$$I := \int_a^b \frac{1}{t^2+1} dt = \arctan b - \arctan a.$$

Unfortunately, the obvious formula for computing this, namely

$$q := \arctan b - \arctan a,$$

is neither accurate nor particularly fast.

Consider instead

$$r := \arctan\left(\frac{b-a}{1+ab}\right).$$

The formulas q and r are mathematically equivalent provided that $1+ab > 0$; otherwise $q - r = sign(b)\pi$. However, the formula for r trades the computation of one arctangent for three arithmetic operations, and so computing r is likely to be faster. A more important consideration may be the fact that q and r differ significantly in numerical properties when a is relatively close to b. In particular, using 16 decimal digits (IEEE 754 double precision) let a=5.0×10^7, $b = a + 1$; then $q = 4.44\ldots \times 10^{-16}$, $r = 3.99\ldots \times 10^{-16}$. As it happens, q is wrong in even the first digit, but r is good to about 16 digits. The trouble with q is that cancellation leaves nothing but the rounding errors in the two arctangents. In r, the cancellation is harmless.

Before leaving this example, we point out there is another hazard in such integration formulas, whether expressed as the difference of arctangents or our form: you may provide an incorrect integral by incorrect choice of branches of the artangent function. Moritsugu [16] points this out with the following example: If we differentiate the expression

$$f = \arctan(\frac{x^3-4x}{x^2-1})$$

we find the answer is

$$d = \frac{df}{dx} = \frac{x^4+x^2+4}{x^6-7x^4+14x^2+1}.$$

A sufficiently powerful algorithmic attack on algebraically integrating the rational function d could hardly do much better than f. Therefore, one might falsely conclude that we could compute the integral of d with respect to x from -2 to 2 by substituting the values -2 and 2 into f, and computing the difference. Those values are each 0, so the integral computed this way would be 0: yet d is strictly positive in that interval. (A proof: Algebraically, the numerator of d is clearly positive, and the denominator is expressible as a sum of squares: $(x^2-1)^2 + (x^3-4x)^2$.) The problem is that you, or the integration program, must notice that f, although bounded, is discontinuous in the range of integration (specifically at -1 and 1). In fact, a correct choice of branches for the arctan function shows that the area under the curve is 2π. Computation with divided differences will not guarantee correct branch choices.

Example 2:

Consider

$$\int_a^b \frac{1}{t} dt = \log b - \log a.$$

Let $a = 1.0 \times 10^{14}$, $b = a + 1$. Using IEEE 754 double precision, we directly compute $\log(b) - \log(a) = 7.105427357601002 \times 10^{-15}$. We can also directly compute $\log(b/a) = 9.992007221626359 \times 10^{-15}$. And we can directly compute from either formula below the answer $9.99999999999995 \times 10^{-15}$ which is correct to all 16 digits.

```
/*dlog(b,a) returns an accurate value for
   log(b)-log(a), a,b > 0*/
dlog(x,y):=block([z,r,w],
 z:(x-y)/y,
 r:1+z,
 if r=1 then w:1 else w:log(r)/(r-1),
 return(w*z))$
```

pol

If an accurate arctanh routine is available, the best that can be done is

```
dlog(x,y):= if (x/y < 0.5) or (2 < x/y)
 then  log(x/y)
 else  2*atanh((x-y)/(x+y))$
```

Example 3:

Integration of a polynomial $f(x)$ simply results in another polynomial $F(x) = \int f(x)dx$. Once again, the obvious way to compute $\int_a^b f(x)dx = F(b) - F(a)$ would be to compute $F(b)$ and $F(a)$ separately and then subtract. But if a and b are floating-point numbers relatively close to one another, the obvious way is likely to be an inaccurate way.

Below we provide a few simple Macsyma programs for dealing with a polynomial p expressed as an array of coefficients. The program `p` evaluates the polynomial, `dp1` computes the divided difference of p, `poly2array` converts the usual infix notation for a polynomial into an array of coefficients, and finally, `dp` puts these pieces together. Given a polynomial $p(z) := a_0z^n + a_1z^{n-1} + \cdots + a_n$, `dp` first places the coefficients $\{a_i\}$ of $p(z)$ in an array `%a`, then computes a divided difference `dp1(x,y,%a,n)` where n= degree(p,z). This result is then multiplied by $(x - y)$ to obtain a value for $p(x) - p(y)$. However, the arrangement of calculations in `dp` makes the last operation a multiplication rather than an addition or subtraction, promoting accuracy.

```
/* The program p evaluates a polynomial.
   "a" is the array of coefficients,
    x is the point for evaluation,
    n >= 0 is the degree of the polynomial */
p(x,a,n):=
 block([p:0,j],
       for j:0 thru n do (p:x*p+a[j]),
       p)$

  /* The program dp1 evaluates (p(x,a,n)-p(y,a,n))/(x-y) */
dp1(x,y,a,n):=
 block([d:0,p:0,j],
       for j:0 thru n-1 do (p:x*p+a[j], d:y*d+p),
       d)$

poly2array(p,x):=
  block([n:hipow(p,x),i],
 /* allocate n+1 elements: %a[0] thru %a[n] */
array(%a,n+1),
        for i:0 thru n do %a[n-i]:ratcoef(p,x,i),
        %a)$

/* dp(x,y,p) evaluates p(x)-p(y) accurately, given p=p(z) */
dp(x,y,p):= (x-y)*dp1(x,y,poly2array(p,z),hipow(p,z))$
```

Since the program works as well for polynomials with symbolic expressions as coefficients, we can try it out. In fact, `dp(`$x, y, az^2 + bz + c$`)` evaluates to $(x - y)\ (a\,y + a\,x + b)$.

In the spirit of using divided differences within the context of automatic symbol manipulation, the reader should notice the resemblances between the *programs* `p` and `dp1`. One might (correctly) conclude that "automatic programming" of `dp1` from `p` could be done by symbol manipulation.

We can test some polynomials for accuracy. Consider the polynomial of degree 10:

$$\begin{aligned} p(z) \quad &:= \quad z \cdot (z-1) \cdot (z-2) \cdot (z-3) \cdot (\cdots) \cdot (z-8) \cdot (z-9) \\ &= \quad (((((((((z-45) \cdot z + 870) \cdot z - 9450) \cdot z + 63273) \cdot z - 269325) \\ &\qquad \cdot z + 723680) \cdot z - 1172700) \cdot z + 1026576) \cdot z - 362880) \cdot z \end{aligned}$$

and suppose we wish to compute

$$\triangle p([x, y]) := \frac{p(x) - p(y)}{x - y}$$

for numerical values x and y very close together. For instance, consider $x = 5+\xi$ and $y = 5 - \eta$ for very tiny numerical values ξ and η. If the obvious formula for $\triangle p$ were used, roundoff in $p(x)$ and $p(y)$ might be all that was left after $p(x) - p(y)$ cancelled, so the formula could be very unsatisfactory numerically. Let us try several ways.

The polynomial evaluation procedure based on Horner's recurrence as program `p` is a reasonable way to compute `p` $\approx p(z) = \sum_0^n a_j z^{n-j}$. The augmented recurrence in program `dp1` computes `d` $\approx \triangle p([x, y])$ without losing all accuracy as $y \to x$.

As a specific example, if we take $\xi = \eta = 3 \times 10^{-11}$ on a 12-digit calculator, so that $x = 5.00000000003$ and $y = 4.99999999997$, we find that Horner's recurrence produces $p(x) \approx -0.001265$ instead of 8.64×10^{-8} and $p(y) \approx 0.001265$ instead of -8.64×10^{-8}. Then $(p(x) - p(y))/(x - y) \approx 4.2 \times 10^7$ instead of the value $\triangle p = 2880$ (which is correct to 21 significant decimals).

The augmented recurrence produces $\triangle p \approx d = 2879.999$, which is a vast improvement since it loses only five of the figures carried instead of all of them. Thus divided-difference-inspired routine works well, and in the absence of other information, is to be recommended[1].

Even without such marked symmetries in the polynomial, there are general methods for improving evaluation accuracy if there is sufficient reason to believe the evaluation will be done many times. Such an analysis could be based on analysis of nearby polynomial zeros (see Meszteny and Witzgall [14]).

Example 4:

Integration of rational functions (ratios of polynomials) results in a combination of rational functions, polynomials and logarithms (or arctangents). Therefore the remaining case is the evaluation of the difference of rational functions $R(b) - R(a)$ where

$$R(z) = \frac{N(z)}{D(z)}.$$

Rearrange the computation of $R(b) - R(a)$ as:

$$\frac{(D(b) + D(a))\ (N(b) - N(a)) \quad - \quad (D(b) - D(a))\ (N(b) + N(a))}{2\, D(a)\ D(b)}.$$

Evaluation of the polynomial differences in the numerator can be performed by the method of example 3, using the divided-difference program `dp1`, and factoring out the $(x - y)$. That is, if `lN` is the list of coefficients in N, `degN` is the degree of N, and similarly for D, a formula for evaluation of $R(x) - R(y)$ looks like

```
(x-y)/(2*D(x)*D(y)) * ( dp1(x,y,lN,degN)*(D(x)+D(y))
                        -dp1(x,y,lD,degD)*(N(x)+N(y))).
```

Evaluation of the non-differenced polynomials `D(x), D(y), N(x), N(y)` can be done by Horner's Rule or the method suggested by Meszteny and Witzgall [14]. As can be seen, the fundamental idea is again that of factoring out

[1]But for this test case we can do even better. We can retain near-full-accuracy by exploiting the special nature of $p(z)$ as a product of factors $(z-j)$ by ordering the factors to take account of our special interest in values $z = x$ and $z = y$ very near 5. Details are discussed in an extended earlier version of this paper[11]

$(x - y)$ so that the computation of the numerator ends with a multiplication rather than an addition or subtraction.

Example 5:

Consider now the combination of examples 1 or 2 and 4: the difference of logs (or arctangents) of a rational function. For the first case, $\log R(b) - \log R(a)$ we start by looking at the values for $x = R(b)$ and $y = R(a)$. As shown in example 2, if their ratio is far enough from 1, we can simply return $\log x/y$. If the ratio is between 1/2 and 2, we compute $x - y$ by the method in example 4, and proceed with (either of) the formulas in example 1. Similarly, for arctan, we apply the method in example 4 to the calculation of the numerator in $(x - y)/(1 + xy)$.

By means of these five examples, we have shown how one can reasonably rearrange (or better, produce *ab initio*) more useful forms for results from some existing computer algebra system's symbolic integration programs. Recall that our goal is the production of a program "`Integral_of_f`" of two arguments, the lower and upper bounds of the integral. This program evaluates the polynomial, rational function and log parts, through divided differences. (The arctangents can also be evaluated via complex logs).

In the next section we explain one extension to divided-difference reformulation that may be of particular use in our context of rational function integrals.

Sparse polynomials

The routine reformulation of the difference of *sparse* polynomials by the methods indicated above can spawn substantially more computation than is actually necessary.

For example, consider $p(x) - p(y) = x^{1024} - y^{1024}$. If the program equivalent to `dp` above were executed, it would involve some 2047 multiplications. By computing z^{1024} by ten squarings of z, we can compute $p(x) - p(y)$ in 20 multiplications. This result is inaccurate when x is very near y but it takes far less work. There is another way to compute $p(x, y)$ accurately and fast and, suitably extended, it can be applied to sparse polynomials generally.

Observe that $x^{1024} - y^{1024}$ can be expressed as

$$(x - y)\ (x + y)\ \left(x^2 + y^2\right)\ \left(x^4 + y^4\right)\ \left(x^8 + y^8\right) \cdots \left(x^{512} + y^{512}\right).$$

Also observe that by computing x^2, x^4, etc., by successive squaring, not much waste is involved in computing the factors, and the "extra" cost for additions and multiplications is proportional to the logarithm of the power. Evaluating this expression requires only 28 multiplications.

The power need not be a power of two: Two multiplications suffice to compute $z^3 = z \cdot z^2$, and two multiplications suffice to compute

$$\triangle \uparrow^3 ([x, y]) = \frac{x^3 - y^3}{x - y} = x^2 + xy + y^2 = x \cdot (x + y) + y^2.$$

The last formula derives directly from one of the product rules for $\triangle$ [12] which we quote here:

Suppose $f(z) = g(z) \cdot h(z)$. Then

$$\begin{aligned}
\triangle f([x,y]) &= (f(x) - f(y))/(x-y) \\
&= \triangle g([x,y]) \cdot h(x) + g(y) \cdot \triangle h([x,y]) \\
&= \triangle g([x,y]) \cdot h(y) + g(x) \cdot \triangle h([x,y]) \\
&= \triangle g([x,y]) \cdot \left(\frac{h(x)+h(y)}{2}\right) + \left(\frac{g(x)+g(y)}{2}\right) \cdot \triangle h([x,y]).
\end{aligned}$$

Although three multiplications suffice to compute $z^5 = z \cdot (z^2)^2$, five are needed for its divided difference:

$$(x^5 - y^5)/(x-y) = x^4 + x^3y + x^2y^2 + xy^3 + y^4 = x \cdot (x+y) \cdot (x^2+y^2) + (y^2)^2.$$

The last formula again derives directly from the product rules for $\triangle$:

$$z^5 = \uparrow^5 (z) = z \cdot \uparrow^2 (\uparrow^2 (z)),$$

so

$$\begin{aligned}
\triangle \uparrow^5 ([x,y]) &= x \cdot \triangle \uparrow^2 (\uparrow^2 ([x,y])) + 1 \cdot \uparrow^2 (\uparrow^2 (y)) \\
&= x \cdot \triangle \uparrow^2 ([\uparrow^2 (x), \uparrow^2 (y)]) \cdot \triangle \uparrow^2 ([x,y]) + y^4 \\
&= x \cdot (x^2 + y^2) \cdot (x+y) + y^4.
\end{aligned}$$

Five multiplications suffice to compute $z^{15} = (z^5)^3$. Eleven suffice for its divided difference:

```
x2:x*x, x4:x2*x2, x5:x*x4,
y2:y*y, y4:y2*y2, y5:y*y4,
d: (x5*(x5+y5)+y5*y5)*(x*(x+y)*(x2+y2)+y4).
```

This formula derives from

$$\triangle \uparrow^3 (\uparrow^5 ([x,y])) = \triangle \uparrow^3 ([x^5, y^5]) \cdot \triangle \uparrow^5 ([x,y])$$

which is a consequence of the chain rule for divided differences which we quote here:

Suppose $f(x) = g(h(x))$. Then

$$\triangle f([x,y]) = \triangle g([h(x), h(y)]) \cdot \triangle h([x,y]).$$

Given a program to compute x^n (by repeated squaring, addition chains, etc.) these techniques can be (automatically) applied to produce an appropriate divided difference program.

Summary and Conclusions

Symbolic integration programs can give exact answers as formulas. If the formulas are going to be used for numerical evaluation, then they should be produced by a program designed to generate formulas for efficient and accurate computation. It is possible but somewhat more difficult to have the answers as currently produced post-processed by another program to re-express the results in arithmetically desirable forms, although analysis of domains of validity (see the appendix) is a subtle issue.

For rational function integration where the answers are in terms of arctangents and logarithms, reformulations of the typical results of integration programs have been suggested above.

A substantial collection of additional techniques for stable evaluation of these and other forms can be found by considering the manipulations developed in the calculus of divided differences. (see [12], or [15]).

One might assume that these reformulations require additional arithmetic steps, but these transformations generally use about the same amount of computation (and sometimes less) when compared to the naive approaches. Observe that the expression (in normal mathematical notation) and the program (in some programming language) will not necessarily resemble each other.

More work should be done in the synthesis of symbolic and numerical computing not only in the context of integration, but in any area in which tedious reformulation and manipulation of symbolic information results in executable code. As symbolic manipulation programs become more readily available to persons generating scientific programs, material such as is contained in this paper should be incorporated into standard libraries of symbolic routines, much as there are libraries of numeric routines. Surprisingly even evaluation of polynomials can be a challenge, given the variety of contrainst on this problem [6]. It is plausible that scientific problem solving environments may provide appropriate frameworks for such material.

In this paper our intent has not been to present exhaustively the transformations that would be of interest for such a library. Our intent has been instead to awaken awareness of these transformations, and illustrate a few key ones. Automation of these transformations (for example to "compile" an arbitrary program `h(z)` into a divided-difference version `dh(x,y)`) requires addressing a number of subtleties beyond the scope of this brief paper. See however, [12] for more details.

Acknowledgments

Work reported herein was supported in part by the Army Research Office, grant DAAG29-85-K-0070, (R. Fateman and W. Kahan) and by the Office of Naval Research N00014-85-K-0180 (W. Kahan) through the Center for Pure and Applied Mathematics, University of California, Berkeley, and the National Science

Foundation CCR-9901933 (R. Fateman) through the Electronics Research Laboratory, UC Berkeley.

Appendix

This is a case study of diverse ways to evaluate a particular integral, and where those methods might be useful. Reformulation using divided differences is only one of the tools used. Careful examination of the behavior of the integral in different regions is also necessary. Consider $F(z) = \int f(z)dz + C$ when

$$f(z) = \frac{z+1}{(z-1)^3 \; z^2}.$$

The closed form for $F(z)$ produced by Macsyma, namely

$$-4 \, \log z + 4 \, \log (z-1) + \frac{4\, z^2 - 6\, z + 1}{z^3 - 2\, z^2 + z}, \tag{1}$$

is algebraically a "correct" anti-derivative, but it is too simple; it fails to take into account three difficulties:

1. We must decide whether to regard $F(z)$ as a multi-valued function continuous everywhere in the complex plane except at its logarithmic poles $z = 0$ and $z = 1$, or as a single-valued function continuous everywhere except across a slit that joins those poles. Even if z is restricted to the real axis, the difficulty persists, manifesting itself as a change in the "constant" of integration as z passes each pole.

2. Calculating $F(z)$ numerically from this explicit formula or any other *single* formula runs a serious risk of inaccuracy. At least three formulas are useful: one for the region between the poles, another for huge $|z|$, and a third elsewhere.

3. As we have illustrated in this paper, calculating $\int_x^y f(z)dz = F(y) - F(x)$ numerically from the explicit formula (1) suffers from further inaccuracy caused by cancellation when x and y are relatively close.

If we take the principal values ([10]) of the logarithms in formula (1) it defines $F(z)$ to be an analytic function everywhere except on the "slit" $0 \le z \le 1$, across which $F(z)$ jumps by $8\pi i$. The behavior of $F(z)$ as $z \to \infty$ is not obvious from formula (1) but can be inferred roughly from the following equivalent formula:

$$F(z) = -8 \, \mathrm{arctanh}\left(\frac{1}{2z-1}\right) + \frac{3}{z-1} - \frac{1}{(z-1)^2} + \frac{1}{z}. \tag{2}$$

This formula (2) was derived from (1) by using partial fraction expansion and the divided difference formula

$$\log y - \log x = (y-x) \triangle \log([x,y]) = 2\, \mathrm{arctanh}\left(\frac{y-x}{y+x}\right).$$

Evidently $F(\infty) = 0$. How rapidly does $F(z)$ decay to 0 as $z \to \infty$? By direct computation of a truncated Taylor series for large z (in Macsyma, one can compute `taylor(F(z),[z,0,7,asymp])`) we find that, as $1/z \to 0$, $F(z)$ is approximated by

$$-\frac{1}{3\,z^3} - \frac{1}{z^4} - \frac{9}{5\,z^5} - \frac{8}{3\,z^6} - \frac{25}{7\,z^7} + \cdots . \tag{3}$$

Another approach in Macsyma which requires some tinkering to get the answer in the right form, uses exact computation of power series:

$$f(z) = \sum_{i=1}^{\infty} \frac{n^2}{z^{n+3}}$$

and term-by-term integration yields

$$F(z) = -\sum_{i=1}^{\infty} \frac{n^2}{(n+2)z^{n+2}}.$$

These Taylor (actually Laurent) formulas are valid for all $|z| > 1$, and the truncated version is useful computationally only for $|z| \gg 1$. For instance, calculations on a ten-digit calculator produce

$$F(123.4567) = \begin{cases} -0.00000018475 & \text{from (1)} \\ -0.00000018153 & \text{from (2)} \\ -0.0000001815155894 & \text{from (3)} \end{cases}$$

The last evaluation is correct to ten significant decimals using only the first five terms of the series (3). Although formula (2) is preferable to (1) for all $z > 1$ and all $z < 0$, cancellation leaves it far less accurate than (3) when $|z| > 10$.

Formulas (1) and (2) produce a complex value $F(z) = G(z) \pm 4\pi i$ on the slit $0 < z < 1$, where

$$G(z) := 4\log\left(\frac{1-z}{z}\right) + \frac{3}{z-1} - \frac{1}{(z-1)^2} + \frac{1}{z}.$$

This $G(z)$ provides as accurate a way as any to compute the real integral $\int f(z)dz = G(z)+C$ over any subinterval of $0 < z < 1$. Since $G(0.30947141907...) = 0$, relative accuracy is lost unavoidably as z approaches the zero of $G(z)$. Fortunately, the computation of $G(y) - G(x)$ need not suffer inaccuracy caused by cancellation when x and y are close if, again, divided differences are used to reformulate the calculation.

Provided x and y both lie between 0 and 1,

$$\begin{aligned} &G(y) - G(x) = \\ &\quad 8\,\mathrm{arctanh}\left(\frac{x-y}{y(1-x)+(1-y)x}\right) \\ &\quad + (x-y)\left(\frac{3}{(y-1)(x-1)} - \frac{(x-1)+(y-1)}{(y-1)^2(x-1)^2} + \frac{1}{yx}\right). \end{aligned}$$

The same formula works for $F(y)-F(x)$ when x and y either both exceed 1 or are both negative, provided either x or y lies in the domain where formula (2) would be used. And when both $|x|$ and $|y|$ are so huge that the series (3) should be used to compute $F(x)$ and $F(y)$, but x and y are so close that $F(y)-F(x)$ suffers serious loss of accuracy because of cancellation, proceed as follows. Since x and y are both close and huge, the same initial terms of the series (3) would be used to calculate both $F(x)$ and $F(y)$. These terms amount to a polynomial $P(1/z)$; in other words, $F(x) = P(1/x)$ and $F(y) = P(1/y)$ to working accuracy. To about the same relative accuracy,

$$F(y) - F(x) = (x-y)\texttt{dp1}(1/y, 1/x, P(z))/(yx),$$

where `dp1` is the program that calculates the polynomial divided difference of P:

$$\triangle P([a,b]) = (P(a) - P(b))/(a-b).$$

As should be evident from the analysis in this appendix, a number of computational and at least partly symbolic steps may be needed beyond deriving an anti-derivative to obtain a full measure of usefulness from a symbolic integration program, even in the case of a rational function integrand.

References

[1] Manuel Bronstein. *Integration of elementary functions.* Ph.D. dissertation, Mathematics Dept. Univ. Calif. Berkeley. April, 1987.

[2] Bruno Buchberger, George Collins, Rudiger Loos, A. Albrecht, (eds). *Computer Algebra*, Springer Verlag, 1984.

[3] James Davenport. “Integration: Formal and Numeric Approaches,” in *Methods and Languages for Scientific Computation* (North-Holland) also Appendix 2 in *Integration Formelle* R.R. No. 375, Lab. d’informatique et de Math. Appl. de Grenoble. March, 1983.

[4] Richard J. Fateman. “Symbolic Manipulation Languages and Numerical Computation: Trends,” in J. K. Reid (ed) *The Relationship between Numerical Computation and Programming Languages*, (North-Holland), 1982, 117-130.

[5] Richard J. Fateman. “Computer Algebra and Numerical Integration,” in P. S. Wang (ed) *ACM Proc. SYMSAC 81*, Aug., 1981, 228-232.

[6] Richard J. Fateman. “Code generation: evaluating polynomials” in preparation.

[7] Keith R. Geddes. “Numerical Integration in a Symbolic Context,” in B. W. Char (ed) *ACM Proc. SYMSAC 86*, July 1986, 185-191.

[8] W. Kahan. “Divided Differences of Algebraic Functions,” Course notes for Math 228A, Univ. Calif. Berkeley, Fall, 1974. 15 pages, unpublished.

[9] W. Kahan. “Handheld Calculator Evaluates Integrals,” *Hewlett-Packard Journal 31*, 8, 1980, 23-32.

[10] W. Kahan. “Branch Cuts for Complex Elementary Functions, or Much Ado about Nothing’s Sign Bit,” Chapter 7 of A. Iserles and M. J. D. Powell (eds). *The State of the Art of Numerical Analysis*, Oxford Univ. Press, 1987.

[11] W. Kahan and R. J. Fateman. Improving Exact Integrals from Symbolic Computation Systems, Tech. Rept. Ctr. for Pure and Appl. Math. PAM 386, Univ. Calif. Berkeley. 1986.

[12] W. Kahan and R. J. Fateman. Symbolic Computation of Divided Differences, (original notes dated 1992), SIGSAM Bulletin Volume 33, Number 2, June, 1999, Issue 128, 7-28.

[13] The Mathlab Group. *Macsyma Reference Manual*, Lab. for Comp. Sci, MIT, Jan, 1983 (2 volumes: version 10), available also from the National Energy Software Center (NESC), Argonne, IL. Similar manuals are available from Symbolics, Inc.

[14] C. Meszteny and C. Witzgall. “Stable Evaluation of Polynomials,” *J. of Res. of the Nat’l Bur. of Stds.-B, 71B*, no 1 (Jan., 1967) 11-17.

[15] L. M. Milne-Thomson. *The Calculus of Finite Differences*, Macmillan, N.Y. 1933.

[16] Shuichi Moritsugu and Makoto Matsumoto. “A Note on the Numerical Evaluation of Arctangent Functions,” SIGSAM Bulletin 23, no. 3.(July, 1989) 8-12.

[17] Barry M. Trager. *Integration of Algebraic Functions*, PhD. dissertation, Mass. Inst. of Techn. EECS Dept. 1984.