Symbolic Computation of Divided Differences

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

December 8, 1999

Abstract

Divided differences are enormously useful in developing stable and accurate numerical formulas. For example, programs to compute $f(x) - f(y)$ as might occur in integration, can be notoriously inaccurate. Such problems can be cured by approaching these computations through divided difference formulations. This paper provides a guide to divided difference theory and practice, with a special eye toward the needs of computer algebra systems that should be programmed to deal with these often-messy formulas.

Contents

Introduction

The calculus of finite differences[8] manipulates *divided differences* like

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

according to rules that resemble the chain rule, product and quotient rules, inverse and implicit function rules, etc., familiar in the calculus of derivatives.

Divided differences are enormously useful in developing stable and accurate numerical formulas. To illustrate this point let's give a very simple example: Suppose we wish to compute $x^2 - y^2$. We consider the alternative $(x-y) * \triangle F((x,y))$ where F is the function $F(z) = z^2$. The divided difference of F on the set$\{x, y\}$ is $x + y$. Therefore the technique suggests computing $d_1 = x^2 - y^2$ by computing instead $d_2 = (x-y)(x+y)$. In fact using IEEE double-precision binary standard arithmetic (about 16 decimal digits), let $x = 10^{15} + 1$ and $y = 10^{15} - 1$. Then d_1 is about $4.01108 \cdot 10^{15}$. The evaluation of d_2 results in $4.0 \cdot 10^{15}$. The expression d_2 is correct to all 16 places, but d_1 is wrong in the third digit.

The most commonly useful version of computation of divided differences (namely the scalar, first order case) resembles ordinary derivatives. In this case the operation on an expression typically does not raise the level of its 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 adept at simplification, differentiation and even integration, do not cater to divided differences.

There are several good reasons for this:

1. Although the derivative of a function is generally about as complicated as the function, the divided difference is generally about twice as complicated, but often much worse.

2. There is considerable variety in the nature of divided difference forms: determining which of the numerous possible forms is most appropriate can be difficult to automate; it is sensitive to the context in which the result will be used, and may require careful analysis or "on-line" interactive decision-making.

3. The usefulness of the techniques, and details of the needed manipulations, have not been made generally available in a useful form.

Although the result for a simple example is easy to observe, the more usual growth of divided difference forms makes human inspection of explicit results much less rewarding than examining derivatives. 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, perhaps by computer algebra systems, on 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 evaluation of integrals from explicit symbolic formulas often leads to the computation of forms like $F(y) - F(x)$. The application of divided differencing to such forms, especially when produced by computer algebra systems is therefore natural [2].

The subsequent sections of this paper consist of a general discussion of divided differences, and then a more detailed "program-oriented" approach to their automatic computation. Our intention is that this paper will begin to address problems in notation and specification a modest collection of useful formulas, and suggestions for computation and interaction by means of computer algebra systems.

1 Divided Differences of Algebraic Functions

This section[1] is intended to survey aspects of divided differences likely to be pertinent to a computer algebra system.

[1]This section is based on notes originally produced by W. Kahan for the course Math 228A, Univ. Calif. Berkeley, Fall, 1974

1.1 Definition

Given a function $f(\xi)$ with a *scalar* argument ξ we define

$$\triangle f((x,y)) \equiv \begin{cases} (f(y)-f(x))/(y-x) & \text{if } x \neq y \\ \frac{d}{d\xi} f(\xi) \quad \text{evaluated at } \xi = x & \text{if } x = y \end{cases}$$

with the understanding that whenever possible the division is to be carried out before evaluation in order to avoid the possibility of 0/0 arising when $x = y$. For example, let $\uparrow^n$ be the power function defined by $\uparrow^n (x) := x^n$ for integer n, positive, negative or zero. Then

$$\triangle \uparrow^n ((x,y)) = \sum_{j=1}^{n} x^{j-1} y^{n-j}.$$

As another example, provided $x \geq 0$ and $y > 0$, $\triangle\sqrt{((x,y))} = 1/(\sqrt{x}+\sqrt{y})$.

In general we see that $\triangle f((x,y)) = \triangle f((y,x))$ and $f(x) = f(y) + (x-y)\triangle f((x,y))$. These equations can be generalized to

- higher order divided differences: $\triangle^n f((w,x,\ldots,y,z))$
- functions of several scalar arguments:

$$\triangle f((w_1,x_1), w_2, (w_3,x_3,\ldots,y_3), x_4, (w_5,x_5),\ldots)$$

- functions of vector arguments:

$$f(x) = f(y) + \triangle f((y,z))(x-y) + \triangle^2 f((x,y,z))(x-y)(x-z)$$

We shall examine these generalizations in order, starting with higher order divided differences of functions with one scalar argument.

1.2 Higher Order Differences

Let $f(x)$ be an infinitely differentiable function of a scalar variable x. For a set (x) with only one scalar member x we define

$$\triangle^0 f((x)) \equiv f(x);$$

and for a set $(w,x,\ldots,y,z)$ with $n+1$ members, $(n+1 \geq 2)$,

$$\begin{aligned} \triangle^n f((\underbrace{w,x,\ldots,y,z}_{n+1})) &\equiv \frac{\triangle^{n-1} f((w,x,\ldots,y)) - \triangle^{n-1} f((x,\ldots,y,z))}{w-z} \qquad \text{if } \ w \neq z, \\ &\equiv \left.\frac{\partial}{\partial \xi}\triangle^{n-1} f((\xi,x,\ldots,y))\right|_{\xi=z} \text{if } w = z \end{aligned}$$

Theorem 1.1 *If all the elements x_j of $(x_1,x_2,\ldots,x_{n-1},x_n)$ are distinct (i.e., $x_i = x_j \Rightarrow i = j$) then*

$$\triangle^{n-1} f((x_1,x_2,\ldots,x_{n-1},x_n)) = \sum_{j=1}^{n} \big(f(x_j) / \prod_{k=1,k\neq j}^{n} (x_j - x_k)\big)$$

Proof 1.1 *By induction on n.*

Corollary 1.1 *If all the elements x_j of $(x_1,x_2,\ldots,x_{n-1},x_n)$ are distinct, then $\triangle^{n-1} f((x_1,x_2,\ldots,x_{n-1},x_n))$ is unchanged by permutations of the x_j's; i.e. $\triangle^{n-1} f$ depends only upon the* unordered *set $(x_1,x_2,\ldots,x_{n-1},x_n)$ and f.*

This corollary remains true for *confluent* divided differences (i.e. , when the x_j's in $(x_1, x_2, \ldots, x_{n-1}, x_n)$ are not all distinct) but the proof requires somewhat more complicated machinery.

Let m_i be the number of x_j's in $(x_1, x_2, \ldots, x_{n-1}, x_n)$ with the same value as x_i; e.g. for $(1,2,1,3,1)$, $m_1 = m_3 = m_5 = 3$ and $m_2 = m_4 = 1$. Then let $Pf(x; (x_1, x_2, \ldots, x_{n-1}, x_n))$ be the polynomial in x of minimal degree which interpolates f with order m_i at x_i for $i = 1, 2, \ldots, n$; i.e.

$$(\frac{\partial}{\partial \xi})^j Pf(\xi; (x_1, x_2, \ldots, x_{n-1}, x_n))\Big|_{\xi = x_i} = (\frac{d}{d\xi})^j f(\xi)\Big|_{\xi = x_i}$$

for $j = 0, 1, 2, \ldots, m_i - 1$, for $i = 1, 2, \ldots, n$, e.g. $Pf(x; (a, b)) = f(a) + (x - a)\triangle f((a, b))$.

Theorem 1.2 *$Pf(x; (x_1, x_2, \ldots, x_{n-1}, x_n))$ is uniquely determined by the foregoing definition as a polynomial in x of degree less than n, and depends only upon the unordered set $(x_1, x_2, \ldots, x_{n-1}, x_n)$ and f, and for any ordering is given by Newton's Divided Difference formulae:*

$$\begin{aligned} Pf(x; (x_1, x_2, \ldots, x_{n-1}, x_n)) &= f(x_1) + (x - x_1)\triangle f((x_1, x_2)) \\ &\quad + (x - x_1)(x - x_2)\triangle^2 f((x_1, x_2, x_3)) + \ldots \\ &\quad + (x - x_1)(x - x_2)(\ldots)(x - x_{n-1}) \\ &\quad \cdot \triangle^{n-1} f((x_1, x_2, \ldots, x_{n-1}, x_n)) \end{aligned}$$

Proof 1.2 *Too long to give here.*

Corollary 1.2 *Same as the previous corollary without requiring distinct x_j's.*

Proof 1.3 *Look at the coefficient of x^{n-1} in Pf.*

Corollary 1.3

$$\begin{aligned} f(x) &= Pf(x; (x_1, x_2, \ldots, x_{n-1}, x_n)) \\ &\quad + \prod_{j=1}^{n} (x - x_j) \cdot \triangle^n f((x_1, x_2, \ldots, x_{n-1}, x_n, x)) \end{aligned}$$

The last term provides a remainder term in certain cases.

Theorem 1.3 *(Hermite & Genocchi). Write $f^{(n-1)}(\xi) \equiv (\frac{d}{d\xi})^{n-1} f(\xi)$; Then*

$$\triangle^{n-1} f((x_1, x_2, \ldots, x_{n-1}, x_n)) = \int\int \cdots \int f^{n-1}(\textstyle\sum_{j=1}^{n} \sigma_j x_j) d\sigma_1 d\sigma_2 \ldots d\sigma_n$$

$$\textit{All } \sigma_j \geq 0 \textit{ and} \sum \sigma_j = 1$$

$$= \int_{\theta_1=0}^{1} \int_{\theta_2=0}^{\theta_1} \cdots \int_{\theta_{n-1}=0}^{\theta_{n-2}} f^{n-1}(x_1 + \theta_1(x_2 - x_1) + \theta_2(x_3 - x_2) + \ldots + \theta_{n-1}(x_n - x_{n-1}))$$

$$\cdot d\theta_{n-1} d\theta_{n-2} \ldots d\theta_2 d\theta_1$$

Proof 1.4 *By induction, integrating the previous corollary by parts.*

Corollary 1.4 $\triangle^{n-1} f((x_1, x_2, \ldots, x_{n-1}, x_n)) = \frac{1}{(n-1)!}$ *{A positively weighted average of $f^{(n-1)}(\xi)$ over the smallest interval containing all s_j's } and hence bounds upon $f^{(n-1)}$ apply also to $\triangle^{n-1} f$. In particular,*

$$\triangle^{n-1} f((\underbrace{x, x, \ldots, x, x}_{n})) = f^{(n-1)}(x)/(n-1)!$$

Warning: Don't forget the denominator.

Corollary 1.5 *Taylor series with remainder:*

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + \frac{h^2 f"(x)}{2} + \ldots + \frac{h^{n-1} f^{(n-1)}(x)}{(n-1)!} \\ &\quad + h^n \triangle^n f((\underbrace{x, x, \ldots, x}_{n}, x+h)) \end{aligned}$$

and

$$\triangle^n f((\underbrace{x, x, \ldots, x}_{n}, x+h)) = \int_0^1 \frac{(1-\theta)^{n-1} f^{(n)}(x+\theta) d\theta}{(n-1)!}$$

The first n terms provide $Pf(x+h; (\underbrace{x, x, \ldots, x}_{n}))$.

1.3 Functions of Several Scalar Arguments

Let $f(w, x, \ldots, y, z)$ be an infinitely differentiable function of several scalar variables $w, x, \ldots, y, z$. Once again we may define recursively

$$\triangle^0 f((w), (x), \ldots, (y), (z)) \equiv f(w, x, \ldots, y, z)$$

$$\begin{aligned} &\triangle f((w_1, w_2, \ldots, w_m), (x_1, x_2, \ldots, x_n), \ldots) \\ &\equiv \frac{\triangle f((w_1, w_2, \ldots, w_{m-1}), (x_1, x_2, \ldots, x_n), \ldots) - \triangle f((w_1, w_2, \ldots, w_m), (x_2, x_3, \ldots, x_n), \ldots)}{w_1 - w_m} \\ &\quad \text{if } w_1 \neq w_m \\ &\equiv \frac{\partial}{\partial \omega} \triangle f((\omega, w_2, w_3, \ldots, w_{m-1}), (x_1, x_2, \ldots, x_n), \ldots)\Big|_{\omega = w_1} \\ &\quad \text{if } w_1 = w_m \\ \text{or} & \\ &\equiv \frac{\triangle f((w_1, w_2, \ldots, w_m), (x_1, x_2, \ldots, x_{n-1}), \ldots) - \triangle f((w_1, w_2, \ldots, w_m), (x_2, x_3, \ldots, x_n), \ldots)}{x_1 - x_n} \\ &\quad \text{if } x_1 \neq x_n \\ &\equiv \frac{\partial}{\partial \xi} \triangle f((w_1, w_2, \ldots, w_m), (\xi, x_2, \ldots, x_{n-1}), \ldots)\Big|_{\xi = x_1} \\ &\quad \text{if } x_1 = x_n \\ \text{etc.} & \end{aligned}$$

Theorem 1.4 $\triangle f((w_1, w_2, \ldots, w_m), (x_1, x_2, \ldots, x_n), \ldots)$ *depends only upon the unordered sets* $(w_1, w_2, \ldots, w_m)$, $(x_1, x_2, \ldots, x_n), \ldots$ *and f.*

Proof 1.5 *As before.*

In other words, whether we difference first with respect to w or first with respect to x does not matter.

However, the analogue of Newton's Divided Difference formula *does* depend upon the ordering of the variables now in a complicated way. For example, with a function $f(w, x)$ of just two scal¡ar arguments

$$\begin{aligned} f(w+h, x+k) &= f(w, x) + h\triangle f((w+h, w), x) + k\triangle f(w+h, (x+k, x)) \\ &= f(w, x) + h\triangle f((w+h, w), x+k) + k\triangle f(w, (x+k, x)) \end{aligned}$$

More generally, $Pf(w; (w_1, w_2, \ldots, w_m), x; (x_1, x_2, \ldots, x_n))$ is quite well defined as a polynomial in w and x which interpolates $f(w, x)$ for w in $(w_1, w_2, \ldots, w_m)$ and x in $(x_1, x_2, \ldots, x_n)$. Indeed, Pf is of degree $< m$ in w and $< n$ in x, with mn coefficients that are determined by mn equations requiring that Pf match f at (w_i, x_j) for i in $(1, 2, \ldots, m)$ and j in $(1, 2, \ldots, n)$. But the remainder $f - Pf$ cannot be expressed simply in terms of one divided difference as before. This is why high-order divided differences of functions of several scalar arguments are not much used. We shall return to the remainder problem when we consider divided differences of functions of vector arguments.

As far as the symbolic computation of divided differences is concerned, several scalar arguments pose no new problems not present for one scalar argument. Here is an example: bi-linear interpolation to a function $f(w,x)$:

$$\begin{aligned}
Pf(w;(w_0,w_1),x;(x_0,x_1)) &= f(w_0,x_0)+(x-x_0)\triangle f(w_0,(x_0,x_1)) \\
&\quad +(w-w_0)\triangle f((w_0,w_1),x_0) \\
&\quad +(w-w_0)(x-x_0)\triangle^2 f((w_0,w_1),(x_0,x_1)) \\
&= f(w,x)-(x-x_0)(x-x_1)\triangle f(w,(x,x_0,x_1)) \\
&\quad -(w-w_0)(w-w_1)\triangle f((w,w_0,w_1),x) \\
&\quad +(x-x_0)(x-x_1)(w-w_0)(w-w_1) \\
&\quad \cdot\triangle f((w,w_0,w_1),(x,x_0,x_1))
\end{aligned}$$

In some cases another form for the remainder

$$\begin{aligned}
f(w,x)-Pf(w;(w_0,w_1),x;(x_0,x_1)) &= \\
&\quad (x-x_0)(x-x_1)P\triangle f(w;(w_0,w_1),(x,x_0,x_1)) \\
&\quad +(w-w_0)(w-w_1)\triangle f((w,w_0,w_1),x)
\end{aligned}$$

may be advantageous; here $P\triangle f(w;(w_0,w_1),(x,x_0,x_1))$ is the linear polynomial in w which interpolates $\triangle f(w,(x,x_0,x_1))$ at $w=w_0$ and $w=w_1$. Hence we may deduce that the remainder has the form

$$\begin{aligned}
&(x-x_0)(x-x_1)\ \{\text{a positively weighted average of } \frac{\partial^2 f}{\partial x^2}(w,x) \text{ over a rectangle } R\} \\
&+(w-w_0)(w-w_1)\ \{\text{a positively weighted average of } \frac{\partial^2 f}{\partial w^2}(w,x) \text{ over a rectangle } R\}
\end{aligned}$$

where the rectangle R has corners $(w_0,x_0),(w_0,x_1),(w_1,x_0),(w_1,x_1)$, *provided* (w,x) lies in R.

1.4 Rules

We consider here a collection of rules, analogous to those for differentiation, by which the symbolic computation of divided differences may be simplified.

(a) *Linearity.*

$$\triangle(f+g)((x_1,x_2,\ldots,x_n)) = \triangle f((x_1,x_2,\ldots,x_n))+\triangle g((x_1,x_2,\ldots,x_n))$$

(b) *Product.*

$$\triangle(fg)((x,y)) = \triangle f((x,y))\frac{g(x)+g(y)}{2}+\frac{f(x)+f(y)}{2}\triangle g((x,y))$$

(c) *Reciprocal.*

$$\triangle(\frac{1}{f})((x,y)) = -\triangle f((x,y))/(f(x)f(y))$$

(d) *Chain.*

$$\triangle f(g)((x,y)) = \triangle f((g(x),g(y)))\triangle g((x,y))$$

(e) *Power.* Let $\uparrow^n$ be the function $\uparrow^n(x)\equiv x^n$. Then

$$\triangle\uparrow^n((x,y)) = \sum_{j=1}^{n} x^{n-j}y^{j-1}$$

for all integers n, negative included.

(f) *Inverse function.* If $g = f^{-1}$ in the sense that $f(g) = \uparrow$ then

$$\Delta f^{-1}((x,y)) = \Delta g((x,y)) = 1/\Delta f((g(x),g(y)))$$

(g) *Quotient.*

$$\Delta(f/g)((x,y)) = \frac{\Delta f((x,y))\frac{g(x)+g(y)}{2} - \frac{f(x)+f(y)}{2}\Delta g((x,y))}{g(x)g(y)}$$

(h) *Implicit Function.* If $f(g(x),x) = 0$ defines $g(x)$ then

$$\Delta g((x,y)) = -\Delta f(g(y),(x,y))/\Delta f((g(x),g(y)),x) = \ldots \begin{pmatrix} \textit{exchange} \\ x\&y \end{pmatrix} \ldots$$

Aside: The foregoing eight rules are obviously not independent. If economy of rules is important, we can get by with just the following:

Linearity. If α and β are constants,

$$\Delta(\alpha f + \beta g)((x,y)) = \alpha \Delta f((x,y)) + \beta \Delta g((x,y))$$

First Powers.

$$\Delta \uparrow ((x,y)) = 1, \quad \Delta \uparrow^0 ((x,y)) = 0 \quad (\uparrow^0 \equiv 1)$$

General Chain. The divided difference of $f(g_1(x), g_2(x), \ldots, g_n(x))$ is

$$\begin{aligned}\Delta f(g_1, g_2, \ldots, g_n)((x,y)) \quad = \quad & \sum_{j=1}^{n} (\Delta f(g_1(y), \ldots, g_{j-1}(y), (g_j(y), g_j(x)), \\ & g_{j+1}(x), \ldots, g_n(x))) \cdot \Delta g_j((x,y))\end{aligned}$$

There are $n!$ versions of this rule corresponding to the permutations of the subscripts in $g_1, g_2, \ldots, g_n$; by averaging all such versions we may obtain another version of the General Chain Rule which is formally symmetric in x and y.

From this chain rule we may deduce the rule above for Implicit Functions, and from these last four rules obtain all the rest by appropriate substitution and recursion. For example, we obtain the product rule by examining $f(g_1(x), g_2(x)) = g_1(x)g_2(x)$, which is linear in each factor separately. $\Delta \uparrow^n$ for $n > 1$ is dealt with recursively. $\Delta \uparrow^{-1}$ is obtained from the implicit function rule with $f(g,x) \equiv xg - 1$, whence follow the rules for reciprocals, quotients and negative powers.

But there are other kinds of economy besides minimization of the number of rules to be remembered. The next rule saves time and storage, and the rule after that saves precision.

(i) *Polynomials.* If $P(\tau) \equiv \sum_0^n \alpha_j \tau^j$ is a polynomial in τ then an obvious expression

$$\Delta P((\sigma,\tau)) = \sum_0^n \alpha_j \Delta \uparrow^j ((\sigma,\tau)) = \sum_{j=1}^{n} \alpha_j \sum_{k=1}^{j} \sigma^{k-1}\tau^{j-k}$$

provides an inefficient computation. Better is Horner's recurrence: $\beta_n = \alpha_n$ & $\gamma_{n+1} = 0$, $\beta_j = \sigma\beta_{j+1} + \alpha_j$ and $\gamma_{j+1} = \tau\gamma_{j+2} + \beta_{j+1}$ for $j = n-1, n-2, \ldots, 0$ in turn. Then $P(\sigma) = \beta_0$ and $\Delta P((\sigma,\tau)) = \gamma_1$.

(j) *Algebraic functions.* If each coefficient α_j of $P(\tau)$ above is a polynomial $\alpha_j(\xi)$, then setting $P(\tau) = 0$ defines τ as a function $\tau(\xi)$. To get $\Delta\tau((\xi,\eta))$ directly by computing $\tau(\xi)$ and $\tau(\eta)$ and then $(\tau(\xi) - \tau(\eta))/(\xi - \eta)$ is futile when ξ and η are close because of cancellation in the numerator. But

$$\Delta\tau((\xi,\eta)) = -\frac{\sum_0^n \Delta\alpha_j((\xi,\eta))\tau(\eta)^j}{\Delta P((\tau(\xi),\tau(\eta)))\Big|_{\alpha_j = \alpha_j(\xi)}}$$

can be computed without fore-ordained cancellation.

(k) *Higher order divided differences.* If we regard $\triangle^{n-1} f((x_1, x_2, \ldots, x_n))$ as a function of n scalar variables, i.e.

$$g(x_1, x_2, \ldots, x_n) = \triangle^{n-1} f((x_1, x_2, \ldots, x_n)),$$

then

$$\triangle^{n} f((x_1, x_2, \ldots, x_n, x_{n+1})) = \triangle g(x_1, x_2, \ldots, x_{n-1}, (x_n, x_{n+1})).$$

Since the ordering of the x_j's is irrelevant, there are n such formulas.

(l) *Logarithms.* Let

$$L(z) \equiv \frac{1}{2z}\ln(\frac{1+z}{1-z}) = \frac{1}{z}\tanh^{-1} z = \sum_{0}^{\infty} \frac{z^{2j}}{2j+1} \qquad \text{if } |z| < 1$$

.

The series formula for L is displayed to show that the function is defined even if $z = 0$. If one does not have a suitably accurate program for arctanh, one can nevertheless be provided for L.

Then

$$\triangle \ln((x,y)) = \frac{2}{x+y} L(\frac{x-y}{x+y})$$

(m) *Exponentials.* Let

$$S(z) := \begin{cases} 1 & \text{if } z = 0 \\ \frac{\sinh(z/2)}{(z/2)} = \sum_0^\infty \frac{(z/2)^{2j}}{(2j+1)!} & \end{cases}$$

$$T_1(z) := \begin{cases} 1 & \text{if } z = 0 \\ \tanh(z)/z & \text{otherwise} \end{cases}$$

The formula for S is displayed as a series for the same reasons as given for L above.

Then

$$\begin{aligned} \triangle \exp((x,y)) &= \exp(\frac{x+y}{2}) S(x-y) \\ \triangle \sinh((x,y)) &= \cosh(\frac{x+y}{2}) S(x-y) \\ \triangle \cosh((x,y)) &= \sinh(\frac{x+y}{2}) S(x-y) \\ \triangle \tanh((x,y)) &= T_1(x-y)(1 - \tanh(x)\tanh(y)) \end{aligned}$$

(n) *Trigonometric Functions.*

When x and y are substantially separated in value, there is not much advantage, if any, to be gained by using an alternate formula for the divided difference. In the case that the formula based directly on the definition is about as good, as especially if it is less work, we recommend it (see the tangent formula). These formulas work for any real x and y. Let

$$H(z) := \begin{cases} 1 & \text{if } z = 0 \\ \frac{\sin(z/2)}{(z/2)} = \sum_0^\infty \frac{(-1)^j (z/2)^{2j}}{(2j+1)!} & \text{otherwise} \end{cases}$$

$$T_2(z) := \tan(z)/z$$

Again, we provide a formula for H displayed as a series for the same reasons as above. (There is a series for T_2 as well; however, the closed form involves Bernoulli numbers. It begins $1 + x^2/3 + 2x^4/15 + 17x^6/315 + \cdots$.)

Then

$$\begin{aligned}
\triangle \sin((x,y)) &= \cos(\frac{x+y}{2})H(x-y) \\
\triangle \cos((x,y)) &= -\sin(\frac{x+y}{2})H(x-y) \\
\triangle \tan((x,y)) &= \begin{cases} \sec^2 x & \text{if } x=y \\ T_2(x-y)(1+\tan(x)\tan(y)) & \text{if } \tan x \tan y > -1/2 \\ (\tan x - \tan y)/(x-y) & \text{otherwise} \end{cases}
\end{aligned}$$

Note. $\triangle L((x,y))$, $\triangle S((x,y))$, and $\triangle H((x,y))$ have no known simple forms free from both symbolic division and infinite series. This makes higher-order differences of trigonometric, logarithmic, or exponential/hyperbolic forms problematical.

(o) *Inverse Trigonometric Functions.*

There are a variety of forms possible here. (Some of these below can be found in Gradshteyn [3] section 1.625). They can be derived with some algebra and trigonometric simplifications by considering (for example), the solution of $\sin(\arcsin(x)-\arcsin(y)) = z$ by expansion of $\sin(a+b)$ and simplification of the resulting forms (like $\sin(\arccos(x))$). Other choices can be made – for example solving $\tan(\arcsin(x) - \arcsin(y)) = z$ for a different form. If x and y are sufficiently far apart, it may be simpler to use the definition of the divided difference itself.

In each case it is necessary to consider the effects of cancellation for loss of significance, as well as possible problems with domains for choice of the appropriate branch of the inverse. The following formulas are valid for $-1 \le x, y \le 1$. Establishing the validity of the formulas in the complex plane is difficult, and somewhat to our disappointment, computer algebra systems have not been much help. We believe that the second formula for arcsin can be used for all complex x and y except if $x^2 > 1$ or $y^2 > 1$ or $|x-y| > |\sqrt{1-x^2} + \sqrt{1-y^2}|$.

$$\begin{aligned}
\triangle \arcsin((x,y)) &= \begin{cases} (1-x^2)^{-1/2} & \text{if } x=y \\ \arcsin\left(\frac{(x-y)(x+y)}{x\sqrt{1-y^2}+y\sqrt{1-x^2}}\right)/(x-y) & \text{if } xy \ge 0 \text{ or } x^2+y^2<1 \\ \arcsin\left(x\sqrt{1-y^2}-y\sqrt{1-x^2}\right)/(x-y) & \text{(another form)} \\ (-1)^k \frac{\arcsin\left(x\sqrt{1-y^2}-y\sqrt{1-x^2}\right)}{x-y} + k\pi \quad \text{for} \quad \pm k = 0,1,2,\cdots & \text{otherwise} \end{cases} \\
\triangle \arccos((x,y)) &= \begin{cases} -(1-x^2)^{-1/2} & \text{if } x=y \\ \arctan\left(\frac{(y-x)(y+x)}{x\sqrt{1-x^2}+y\sqrt{1-y^2}}\right)/(x-y) & \text{if } xy \ge 0 \\ \arcsin\left(\frac{(y-x)(y+x)}{x\sqrt{1-y^2}+y\sqrt{1-x^2}}\right)/(x-y) & \text{if } xy \le 0 \end{cases} \\
\triangle \arctan((x,y)) &= \begin{cases} (x^2+1)^{-1} & \text{if } x=y \\ \arctan\left(\frac{y-x}{1+xy}\right)/(y-x) & \text{if } x \ne y \text{ and } xy > -1/2 \\ (\arctan x - \arctan y)/(x-y) & \text{otherwise} \end{cases}
\end{aligned}$$

(p) *Inverse Hyperbolic Functions.*

$$\triangle \text{arcsinh}((x,y)) = \begin{cases} (1+x^2)^{-1/2} & \text{if } x=y \\ \text{arcsinh}\left(x\sqrt{1+y^2}-y\sqrt{1+x^2}\right)/(x-y) & \text{if } xy<0 \\ \text{arcsinh}\left(\frac{(x-y)(x+y)}{x\sqrt{1+y^2}+y\sqrt{1+x^2}}\right)/(x-y) & \text{if } xy \ge 0 \end{cases}$$

$$\triangle \text{arccosh}((x,y)) = \begin{cases} (x^2-1)^{-1/2} & \text{if } x = y \\ \text{arcsinh}\left(y\sqrt{x^2-1} - x\sqrt{y^2-1}\right)/(x-y) & \text{if } xy \geq 0 \\ \text{arcsinh}\left(\frac{(x-y)(x+y)}{y\sqrt{x^2-1}+x\sqrt{y^2-1}}\right)/(x-y) & \text{if } xy < 0 \end{cases}$$

$$\triangle \text{arctanh}((x,y)) = \begin{cases} \text{arctanh}\left(\frac{x-y}{1-xy}\right)/(y-x) & \text{if } x \neq y \\ (1-x^2)^{-1} & \text{if } x = y \end{cases}$$

(q) *Other Functions.*

There is no guarantee at all that there is a compact, efficient, and more-accurate alternative form for the divided difference of a function. Examination of "addition formulas" may provide some suggestions for divided differences of additional special functions. Among these, however, Bessel (or related Cylinder) functions are not particularly promising. In accordance with a theorem of Weierstrass (cited by Watson [10] section 11.1), it is not possible to express $J_\nu(x+y)$ as an algebraic function of $J_\nu(x)$ and $J_\nu(y)$. There are Jacobi elliptic function addition formulas but they are sufficiently more complicated than the ones we have exploited for hyperbolic and trigonometric functions that very simple results are unlikely to emerge.

1.5 Examples

(a) The solution of $ax^2 - 2bx + c = 0$ is

$$x_\pm = \frac{b \pm \sqrt{b^2 - ac}}{a}.$$

When, say, $b > 0$ (for simplicity) and $b^2 \gg ac$, the smaller root x_- is hard to compute directly from the formula above without encountering trouble with cancellation and lost figures. If we use

$$\triangle\sqrt{((u,v))} = 1/(\sqrt{u} + \sqrt{v}) \text{ when } u > 0 \text{ and } v > 0,$$

we can arrange our formula to use a divided difference (with $v := b^2$ and u:=$b^2 - ac$)

$$x_- = c\frac{\sqrt{b^2} - \sqrt{b^2 - ac}}{(b^2) - (b^2 - ac)} = c\triangle\sqrt{((b^2, b^2 - ac))} = \frac{c}{b + \sqrt{b^2 - ac}}$$

and the last expression provides a numerically stable result.

(b) We wish to solve

$$\left\{ \begin{array}{c} x^2 + z + 2 = 0 \\ y^2 + z - 2 = 0 \\ xyz - 1 = 0 \end{array} \right\}$$

numerically. However, we wish to avoid finding the solutions with $x = y$ because they are trivial ($x = y = \pm 1, \quad z = 1$). To prevent the numerical method from converging to these solutions we replace

$$\left\{ \begin{array}{c} x^2 + z - 2 = 0 \\ y^2 + z - 2 = 0 \end{array} \right\}$$

by

$$\left\{ \begin{array}{c} x^2 + z - 2 = 0 \\ \triangle : \frac{(x^2+z-2)-(y^2+z-2)}{x-y} = x + y = 0 \end{array} \right\}$$

to obtain the solutions $x = -y = \pm\sqrt{1+\sqrt{2}}$, $z = 1 - \sqrt{2}$.

(c) Examples of divided differences for applications in interpolation, quadrature, or approximation are generally covered by standard texts on numerical analysis or finite differences.

(d) Sometimes a divided difference of $f(x)$ is hard to compute because of cancellation although that of $f^{-1}(y)$ is easy; thus $\triangle f((x,y)) = (f(x) - f(y))/(x-y)$ may suffer from too much cancellation in the numerator and we should use $1/\triangle f^{-1}((f(x), f(y))$ in place of $\triangle f((x,y))$. Let $X = f(x)$, $Y = f(y)$ and consider

$$\triangle f = 1/\triangle f^{-1} = \frac{X - Y}{f^{-1}(X) - f^{-1}(Y)};$$

the denominator may suffer less from cancellation than the numerator, especially if $y = 0$, e. g.

$$\triangle \exp((x,0)) = 1/\triangle \ln((e^x, 1)) = \frac{(\epsilon^x) - 1}{\ln(e^x) - 0};$$

calculate $X = e^x$ *rounded first* and then if $X \neq 1$

$$\triangle \exp((x,0)) = 1/\triangle \ln((X, 1)) = \frac{X - 1}{\ln X - 0}.$$

Observe that there is no cancellation in the denominator. Actually we get $\triangle \exp((x + \mathit{error}, 0))$ where $x + \mathit{error} = \ln(X) = \ln(e^x \mathit{rounded})$, but this is not a problem here.

Without this trick we might easily lose half the figures carried, or more.

1.6 Other Notations

Notations for divided differences are nowhere near standardized, and suffer from anomalies. For example, a widespread notation for $\triangle f(x_1, x_2, \dots, x_n)$ is just $[x_1, x_2, \dots, x_n]$, which is adequate only if just one function f is under consideration. Another notation, used by L. M. Milne-Thomson [8] and many others, is $f[x_1, x_2, \dots, x_n]$, which causes difficulties when we come to functions of several scalar arguments. Even the notation of this paper $\triangle f((x_1, x_2, \dots, x_n))$ which was introduced in ref. [5] is unsuited for computerized algebraic symbolic manipulation, since the introduction of "operator notation" in most such systems leads to inconveniences.

An adequate notation might be something like `Dvd(x in {x[1], x[2], ...}, f(x))`. Then the generalization $\triangle f((x,y),(a,b,c))$ would be typed `Dvd(u in {x,y}, v in {a,b,c}, f(u, v))` or perhaps `Dvd({u in {x,y}, v in {a,b,c}}, f(u, v))`

$\triangle(\uparrow^n + \uparrow^m)((x,y))$ would be typed `Dvd({z in {x,y}, z^n+z^m)`. We work with a variant of this notation in section 3.

1.7 Vector Arguments

Let $f(x)$ be an infinitely differentiable function of the vector argument x. Here by $f'(x)$ we mean a linear operator with the property (Frechet) $f(x+h) = f(x) + (f'(x) + e(x, x+h)) \cdot h$ for all x and h and $e(x, x+h) \to 0$ as $h \to 0$.

Similarly, $f''(x)$ is a bilinear operator ($f''(x) \cdot a \cdot b$ is a linear function of each vector a and b) and symmetric ($f''(x) \cdot a \cdot b = f''(x) \cdot b \cdot a$).

In terms of the components x_i of x we can write

$$\begin{aligned} f'(x) \cdot h &= \sum_i \frac{\partial f(x)}{\partial x_i} h_i \\ f''(x) \cdot a \cdot b &= \sum_i \sum_j \frac{\partial^2 f(x)}{\partial x_i \partial x_j} \cdot a_i \cdot b_j \\ &= f''(x) \cdot b \cdot a \quad \text{because } \frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x_j \partial x_i}. \end{aligned}$$

We wish to define a divided difference $\triangle f((x,y))$ which will satisfy certain relations:

(1) $f(x) = f(y) + \triangle f((x,y)) \cdot (x-y)$

This does not define $\triangle f((x,y))$ uniquely at all. For example, if

$$x = \binom{x_1}{x_2}$$

then we could have for $\triangle f((x,y))$

$$\left(\frac{f\binom{x_1}{y_2} - f\binom{y_1}{y_2}}{x_1 - y_1}, \frac{f\binom{x_1}{x_2} - f\binom{x_1}{y_2}}{x_2 - y_2} \right),$$

or the same with x and y exchanged, or the average of the two foregoing, or

$$\int_0^1 f'(y + \theta(x-y)) d\theta$$

(cf. Hermite's formula), or infinitely many others differing from each other by linear operators which annihilate $(x-y)$.

(2) $\triangle f((x,y)) = \triangle f((y,x))$

This does not define $\triangle f$ uniquely either; the last two examples above satisfy (2) and yet are different in general.

Rather than add abstract relations and deduce a formula, let us leap ahead and state the principal purpose which such divided differences must serve. Analogous to the Taylor series

$$\begin{aligned} f(x+h) &= f(x) + f'(x) \cdot h + \frac{1}{2} f''(x) \cdot h \cdot h + \ldots \\ &\quad + \frac{1}{(n-1)!} f^{(n-1)}(x) \cdot \overbrace{h \cdot h \ldots h}^{n} \\ &\quad + \frac{1}{n!} (f^{(n)}(x) + e_n(x,h)) \cdot \underbrace{h \cdot h \ldots h}_{n} \\ &\quad \text{where} \quad e_n(x,y) \to 0 \quad \text{as} \quad h \to 0, \end{aligned}$$

should be a Newton Divided Difference formula

$$\begin{aligned} f(x) &= f(a) + \triangle f((a,b)) \cdot (x-a) + \triangle^2 f((a,b,c)) \cdot (x-a) \cdot (x-b) + \ldots \\ &\quad + \triangle^{n-1} f((a,b,\ldots,p,q)) \cdot (x-a) \cdot (x-b) \ldots (x-p) \\ &\quad + \triangle^n f((a,b,\ldots,p,q,x)) \cdot (x-a) \cdot (x-b) \ldots (x-p)(x-q). \end{aligned}$$

Just as $f^{(n)}(x)$ is a symmetric multi-linear operator (linear in each of n vector arguments and independent of their order) so should we wish to have $\triangle f((x_o, x_1, \ldots, x_n))$ symmetric and linear in n arguments, and we should also like to ensure that

$$\triangle f((x_0, x_1, \ldots, x_{n-1}, y, y)) = \frac{\partial}{\partial y} \triangle f((x_0, x_1, \ldots, x_{n-1}, y))$$

too in order to preserve continuity in a simple way. The upshot of these requirements turns out to be a unique definition of divided differences, namely Hermite's formula

$$\begin{aligned} \triangle f((x_0, x_1, x_2, \ldots, x_n)) &= \textstyle\int_{\theta_1=0}^{1} \int_{\theta_2=0}^{\theta_1} \cdots \int_{\theta_n=0}^{\theta_{n-1}} f^{(n)}(x_0 + \sum_1^n \theta_j (x_j - x_{j-1})) \\ &\qquad \cdot d\theta_n d\theta_{n-1} \cdots d\theta_2 d\theta_1 \\ &= \tfrac{1}{n!} \{ \text{a positively weighted average of } f^{(n)} \\ &\qquad \text{over the possibly degenerate simplex} \\ &\qquad \text{spanned by the points} \quad x_0, x_1, \ldots, x_n \}. \end{aligned}$$

However, there is a disagreeable aspect to this formula; even if $f(x)$ is a rational function of the elements of x, $\triangle f$ may be non-rational, even transcendental, e. g. $f\binom{x_1}{x_2} = \frac{x_1}{x_2}$ implies

$$\triangle f(\begin{pmatrix} a \\ b \end{pmatrix}, \begin{pmatrix} a+h \\ b+k \end{pmatrix}) = (\frac{1}{k}\ln(1+\frac{k}{b}), \frac{-a}{b(b+k)} - \frac{h}{k}(\frac{1}{k}\ln(1+\frac{k}{b}) - \frac{1}{b+k}))$$

Therefore, except for polynomial functions of vector arguments, divided differences with vector arguments will not be easy to use explicitly; they will probably be more often estimated (bounded) than computed.

2 A Taste of Some Applications to Integration

Recall that our ambition is to provide techniques that can be applied by a computer algebra system to manipulate formulas "sight unseen" so they will be more efficient and accurate considered as numerical recipes. Here are some examples with obvious application to the domain of symbolic integration [2].

2.1 Difference of Arctangents

Consider

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

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 $q - r = \mathrm{sign}(b)\pi$. However, the formula for r trades the computation of one arctangent for three arithmetic operations, and so evaluation of 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 (double precision on a DEC VAX[2]), let a=1.0×10^8, $b = a + 1$; then $q = 8.32\ldots \times 10^{-17}$, $r = 9.99\ldots \times 10^{-17}$. 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.

2.2 Difference of Logarithms

Consider

$$\int_a^b \frac{1}{x} dx = \log b - \log a.$$

Let $a = 1.0 \times 10^{14}$, $b = a + 1$. Using IEEE double precision and a version of the logarithm function in a UNIX operating system we 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 the Macsyma [6] computer algebra system using either of the two programs 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)*/
dlog(x,y):=block([z:(x-y)/y,r],
 r: 1+z,
 z * (if r=1 then 1 else log(r)/(r-1)))$
```

[2]DEC and VAX are trademarks of Digital Equipment Corporation.

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))$
```

These kinds of repairs are easy in any computer algebra system (where the results of symbolic integrations might be first produced). A program equivalent to the first of these techniques using the Mathematica computer algebra system [11] is

```
 dlog[x_,y_]:=Block[{z=(x-y)/y,r},
  r=1.0+z,
  z (If [r==1.0, 1.0, Log[r]/(r-1)])]
```

and in the Maple system [1],

```
 dlog :=proc(x,y) local z,r,w;
  z:=(x-y)/y;   r:=1;
  if r = 1.0 then w := 1.0  else w := log(r)/(r-1) fi;
  z*w
 end;
```

2.3 Difference of Polynomials

If $f(x)$ is a polynomial then $F(x) = \int f(x)dx$ is another. 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 magnify any roundoff incurred in computing F.

For specificity, we provide a few simple Macsyma [6] 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. That is, 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 order 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[0]*x^n +a[1]*x^(n-1) + ... + a[n]
    "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],
        array(%a,n+1), /* allocate n+1 elements: %a[0] thru %a[n] */
        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))$
```

The usual meaning of all the operations here would be to provide explicit computation on floating-point numbers. However, since we are actually dealing with a symbolic computation system we are not restricted to numerical coefficients, and can perform this transformation on polynomials with non-numeric coefficients. With this interpretation, we can try, for example, $\mathtt{dp}(x, y, az^2+bz+c)$. This evaluates to $(x-y)\,(a\,y+a\,x+b)$, giving us a "symbolic proof" that — if we were doing the calculations exactly instead of approximately — we would be computing exactly the required function.

In the spirit of using divided differences within the context of automatic symbol manipulation, the reader should notice the resemblance of the *programs* `p` and `dp1`. One might (correctly) conclude that derivation of `dp1` from `p` could be done without human intervention.

We can try some examples. Consider the polynomial of degree 10:

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

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 other ways.

The polynomial evaluation procedure based upon Horner's recurrence has been given above as program `p` which computes $\mathtt{p} = p(z) = \sum_0^n a_j z^{n-j}$. The augmented recurrence in program `dp1` computes $\mathtt{d} = \triangle p((x,y))$ without losing all accuracy as $y \to x$.

For instance, 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.

Even though the divided difference calculation is far superior, in this 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 $(z-j)$. We order the factors to take account of our special interest in values $z = x$ and $z = y$ very near 5. Let us evaluate in turn

$$\begin{aligned} u(z) &:= z; \\ T(z) &:= (z-9)\cdot(z-1); \\ t(z) &:= T(z)\cdot u(z); \\ S(z) &:= (z-8)\cdot(z-2); \\ s(z) &:= S(z)\cdot t(z); \\ R(z) &:= (z-7)\cdot(z-3); \\ r(z) &:= R(z)\cdot s(z); \\ Q(z) &:= (z-6)\cdot(z-4); \\ q(z) &:= Q(z)\cdot r(z); \\ p(z) &:= (z-5)\cdot q(z). \end{aligned}$$

When we difference this program and observe that (where we use the operator notation $\uparrow^n(x) := x^n$)

$$\triangle(\uparrow^2 -10\uparrow +\mathrm{Constant})((x,y)) = (x-5)+(y-5) = \xi-\eta,$$

so that

$$\triangle T = \triangle S = \triangle R = \triangle Q = (\xi - \eta),$$

we find the following formulas:

$$\begin{aligned}
\triangle t &= (\xi - \eta) \cdot x + T(y); \\
\triangle s &= (\xi - \eta) \cdot t(x) + S(y) \cdot \triangle t; \\
\triangle r &= (\xi - \eta) \cdot s(x) + R(y) \cdot \triangle s; \\
\triangle q &= (\xi - \eta) \cdot r(x) + Q(y) \cdot \triangle r; \\
\triangle p &= q(x) + (y - 5) \cdot \triangle q.
\end{aligned}$$

This sequence produces $\triangle p = 2880$ to 11 significant decimal figures on that 12-digit calculator.

The production of such pleasant results requires unlikely symmetries in the polynomial, but there are general methods for improving evaluation accuracy based on analysis of nearby polynomial zeros (see Meszteny and Witzgall [7]).

2.4 Difference of Rational Functions

Consider 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 section 2.3, 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 [7]. As can be seen, the fundamental idea is again that of factoring out $(x - y)$ so that the computation of the numerator ends with a multiplication rather than an addition or subtraction.

2.5 Difference of 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 is inaccurate when x is very near y but requires 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 $\triangle$.

Three multiplications suffice to compute $z^5 = z \cdot (z^2)^2$, but 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.

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.

3 Divided Difference Manipulation

In this section we discuss in more detail the development of sample programs for rearranging formulas in accordance with the various suggested forms, and how they fit together. Although we are using Macsyma here, virtually any computer algebra system with a suitable floating-point computation model could be used for this purpose.

We emphasize that the true objective, by and large, for divided difference manipulation should **NOT** be merely, as it is in the formulas in section 1, another algebraic expression. The objective should be to develop a subroutine or program fragment that may indeed have intermediate expressions computed and re-used, and may have decision points that depend on numerical inequalities. Such a program would, over a large class of inputs, provide efficient and accurate values for a divided difference expression.

While it is possible (and some might argue, especially convenient) in Macsyma and its underlying system, Lisp, to write "program-writing programs" such meta-programs are not particularly easy to read (Although the interested reader might refer to Norvig [9] for examples of this technique). Instead we illustrate via a suite of programs how perform symbolic rearrangements that, by and large return mere expressions. These can then be turned into programs. The expressions have an implied order of evaluation that will, we hope, be faithfully followed. An attempt to algebraically "simplify" such expressions may lead to disaster with respect to our objectives. We concentrate in the remainder of this paper on scalar first-order differences, with a nod to higher-order differences. The difficulties of vector-valued functions, notationally and computationally suggest that any general approach will just bog down in details, and are not pursued here.

3.1 Scalar Expressions

This program assists in the production of a divided difference from a scalar expression of one variable. The program will in some cases have to be interactive because there are, in general, a number of ways in which a divided difference may be expressed, and they are each about equally plausible.

The simplest program `ddiv(f,z,x,y)` is used to compute an explicit expression for $\triangle f((x, y))$, where `f` is an expression that depends explicitly on the variable `z`. The parameters `x` and `y` are presumably also atomic variables, and they must not occur explicitly in `f` (Technically, they do not occur “free” in `f`).

The output g is an expression $(x - y)g = f(x) - f(y)$. If possible, g will be an ordinary algebraic expression. If, however, there are representations of arbitrary functions, then g will contain divided difference components. Such functions will look like “`Dvd`” forms of sections 2. In the parlance of Macsyma, this is the “noun” form of the divided difference.

```
/*
A MACSYMA program which computes a scalar expression's
two-point divided difference */

/* input: f(z), z, x,y.  z is scalar, as is x and y.

   output: a function g of x and y such that, algebraically,
   g* (x-y) = f(x)-f(y).  If necessary the notation
   dvd(z in {x,y}, f(z)) is used to mean (f(x)-f(y))/(x-y) unless x=y,
   in which case the notation means f'(x). */
/* Important limitations: "undefined" functions must involve
   z in their first argument. Sum, product, integerpower, log,
   and the chain rule work. Exp, which  involves z in its second
   also works. */

/* set up some notation (once) so we can write Dvd(z in {x,y}, f(z)) */
unless dvdnotation=true do
 (matchfix("{","}"), infix("in",180,20), alias(Set,"{"), dvdnotation:true)$

dvdsimp2(f,z,x,y):=
if freeof(z,f) then 0
   else if atom(f) then 1 /* must be z */
   /* could check for sparse polynomial .. */
   else if is_poly(f) then (f:rat(f),dp1(x,y,poly2array(f,z),hipow(f,z)))
   else if is_sum(f) then map(ddiv1,f)
   else if is_prod(f) then ddivprod(f)
   else if is_powr(f) then ddivpowr(f)
   else if is_exp(f) then ddivexp(inpart(f,2))
/* use data-directed dispatch for typical "1-adic"
   operators including log, sin, sinh etc. */
   else if notfalse(get(part(f,0),ddivprog))
       then apply(get(part(f,0),ddivprog),
             [subst(x,z,inpart(f,1)),
                      subst(y,z,inpart(f,1))])
       *ddiv1(part(f,1))
   else if freeof(z,f) then 0
   else if inpart(f,1)=z then Dvd(z in {x,y}, f)
   else ddivchain(f)$

notfalse(p):= not(p=false)$

/* 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],
        array(%a,n+1), /* allocate n+1 elements: %a[0] thru %a[n] */
        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))$


ddiv1(r):= dvdsimp2(r,z,x,y)$  /*use the global values of z, x y */

is_poly(r):= ?polyinx(r,z,false)$ /*built-in. True is r is a poly in z*/
is_sum(r):= is (inpart(r,0)="+")$
is_prod(r):= is (inpart(r,0)="*")$
is_powr(r):= is ((inpart(r,0)="^") and
 (inpart(r,1)=z))$
is_exp(r):= is ((inpart(r,0)="^") and
 (inpart(r,1)=%e) /* inpart(r,2) depends on z */
                  )$

ddivprod(f):= block([%r:inpart(f,1), %s:f/inpart(f,1)],
   ddiv1(%r)* (subst(z=x,%s)+subst(z=y,%s))/2
   +
   ddiv1(%s)* (subst(z=x,%r)+subst(z=y,%r))/2
   )$

ddivpowr(f):= block([%n:inpart(f,2),sumhack:true],
sum(x^(%n-j)*y^(j-1),j,1,%n))$

ddivchain(f) := /* assume f's first argument is the only one which
depends on z */
   block([%g:inpart(f,1)],
  dvdsimp2(substinpart(%u,f,1),%u, subst(z=x,%g),subst(z=y,%g))
  * ddiv1(%g))$


/* see text: S is used for Exp, Cos, Sin. etc.  This is, however,
NOT the way to compute S to minimize errors.  Consider the series
1 +m^2/24+m^4/1920 + m^6/322560 + ... which has no divisions */

auxs(m):= /*sinh(m/2)/(m/2)*/ S(m)$


/* for ddivexp(h), all we know is the expression is of
the form exp(f(z)), not necessarily exp(z) */

ddivexp(h)  := if h=z then exp((x+y)/2)*auxs(x-y) else
/* use the chain rule */
```

```
dvdsimp2(h,z,x,y)
              * dvdsimp2(exp(z),z,subst(z=x,h),subst(z=y,h))$

auxh(z):=2*sin(z/2)/z$

put(sin,lambda([x,y],cos((x+y)/2)*auxh(x-y)), ddivprog)$
put(sinh,lambda([x,y], cosh((x+y)/2)*auxs(x-y)),ddivprog)$
put(cos,lambda([x,y],-sin((x+y)/2)*auxh(x-y)),ddivprog)$
put(cosh,lambda([x,y], sinh((x+y)/2)*auxs(x-y)),ddivprog)$
put(tan,lambda([x,y],cos((x+y)/2)(auxh(x-y)/cos(x)/cos(y))),ddivprog)$
put(tanh,lambda([x,y],(1-(tanh((x+y)/2))^2)*auxs(x-y)/cosh((x-y)/2)),ddivprog)$
put(log,lambda([x,y],
          block([z],
2/(x+y)*subst(z=(x-y)/(x+y),1/z*atanh(z)))), ddivprog)$

/* etc for asin, acos, asinh, acosh, atanh */
```

3.2 Scalar Examples

In the interests of saving space, we show the answers not as they would appear on the screen, but how they appear after being processed by the TEX typesetting system.

```
(c35) dvdsimp2(exp(z^2),z,x,y);
```

(D35) $$(y+x)\ e^{\frac{y^2+x^2}{2}}\ S\left(x^2-y^2\right)$$

```
/*test the answer */
(c36) d35*(x-y)-exp(x^2)+exp(y^2);
```

(D36) $$(x-y)\ (y+x)\ e^{\frac{y^2+x^2}{2}}\ s\left(x^2-y^2\right)+e^{y^2}-e^{x^2}$$

```
(c37) d36,s(z):=sinh((z/2))/(z/2);
```

(d37) $$\frac{2\ (x-y)\ (y+x)\ e^{\frac{y^2+x^2}{2}}\sinh\left(\frac{x^2-y^2}{2}\right)}{x^2-y^2}+e^{y^2}-e^{x^2}$$

```
(c38) %,exponentialize; /* convert sinh to exponentials */
```

(d38) $$-\frac{(x-y)\ (y+x)\ \left(e^{\frac{y^2-x^2}{2}}-e^{-\frac{y^2-x^2}{2}}\right)\ e^{\frac{y^2+x^2}{2}}}{x^2-y^2}+e^{y^2}-e^{x^2}$$

```
(c40) ratsimp(%); \*rationally simplify the result */
```

(d40) $$0$$

3.3 Higher-order Differences

Here we illustrate two versions of a divided difference function that takes a list of points of arbitrary length. The second version illustrates a technique available in Macsyma for remembering useful subexpressions. Since this produces only one of the many possible forms, it is clear that more context-dependent guidance would be needed in general for the most useful results.

```
dvd(exp,var,vals):=
block([n_points:length(vals)],
    if n_points=1
        then at(exp,var=first(vals))
        else if is(first(vals)=last(vals))
                   then at(diff(exp,var,n_points-1) / (n_points-1)!,
                                   var=first(vals))
        else
          (dvd(exp,var,rest(vals,-1)) -dvd(exp,var,rest(vals)))/
           (first(vals)-last(vals)))$

/* This is a faster version, caching values..  */
newdvd(e,vr,vl):= (local (ndvd, atmemo),
atmemo[r,s]:=at(r,s),  /* Two "memo" local functions. They could be global. */
ndvd[exp,var,vals]:=
block([n_points:length(vals)],
    if n_points=1
        then atmemo[exp,var=first(vals)]
        else if is(first(vals)=last(vals))
                   then at(diff(exp,var,n_points-1) / (n_points-1)!,
                                   var=first(vals))
        else
          ratsimp((ndvd[exp,var,rest(vals,-1)] -ndvd[exp,var,rest(vals)])/
               (first(vals)-last(vals)))),
ndvd[e,vr,vl] /*return this.. */)$
```

3.4 Higher-order Examples

```
(C9) ratsimp(dvd('f(x),x,[x-d,x,x+d]));
                         F(X + D) + F(X - D) - 2 F(X)
(D9)                     ----------------------------
                                        2
                                     2 D
l:[1,2,3,4,5,6,7,8,9,10]$
ratsimp(dvd('f(x),x,l)); /* compute and rearrange the result */
(D12) (F(10) - 9 F(9) + 36 F(8) - 84 F(7) + 126 F(6) - 126 F(5) + 84 F(4)

                                        - 36 F(3) + 9 F(2) - F(1))/362880
```

4 Conclusions

Computer algebra systems can give exact answers as formulas. But if these formulas are going to be used for numerical evaluation, it is not sufficient that they be exact! They should be expressed with a view toward efficient and accurate computation. Ideally, manipulation programs should be written with such an end use in mind. If not, it is still possible in some cases to have the answers as currently produced “post-processed” by another program to re-express the results in arithmetically desirable forms, as illustrated here.

For specific problems such as those appearing as the result of rational function integration, where the answers are in terms of arctangents and logarithms plus rationals functions, we have suggested reformulations of the typical results. We give a more detailed explanation of exactly these cases and other hazards in using the result of integration programs, elsewhere[2].

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 we have in general tried to replace an expression in normal "mathematical notation" by *a program* (in some programming language). The two will not necessarily resemble each other superficially.

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. The recent interesting results collected Griewank and Corliss [4] on automatic differentiation constitute additional particularly relevant targets for divided difference techniques.

As symbolic manipulation programs become more readily available to persons generating scientific programs, we hope that material such as is contained in this paper will be incorporated into standard libraries of symbolic routines, much as there are libraries of numeric routines.

In this paper our intent has not been to present an exhaustive list of 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 completely "compile" an arbitrary program `h(z)` into a divided-difference version `dh(x,y)`) requires addressing a number of subtleties that are beyond the scope of this paper.

5 Acknowledgments

Work reported herein has been 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, grant N00014-85-K-0180 (W. Kahan) and the National Science Foundation, grants CCR-8812842 and CCR-9214963 (R. Fateman).

Computing facilities have been supported in part through National Science Foundation Infrastructure Grant number CDA-8722788.

References

[1] B. W. Char, K. O. Geddes, *et al. First Leaves: A Tutorial Introduction to Maple*, (two other volumes – Library Reference and Language Reference are also available). Springer-Verlag, 1992.

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

[3] I. S. Grahshteyn and I. M. Ryzhik. *Table of Integrals, Series, and Products*, Acad. Press 1980.

[4] A. Griewank and G. F. Corliss (eds.). *Automatic Differential of Algorithms*, SIAM , 1991.

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

[6] 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 e.g. version 11, 1985 for VAX UNIX Macsyma) are available from Macsyma, Inc. Arlington, MA.

[7] 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.

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

[9] P. Norvig. *Paradigms of Artificial Intelligence Programming*, Morgan Kaufmann, 1992.

[10] G. N. Watson. *A Treatise on the Theory of Bessel Functions*, 2nd ed., Cambridge Univ. Press, 1980.

[11] S. Wolfram. *Mathematica – A System for doing Mathematics*, Addison-Wesley, 2nd ed. 1991.