Symbolic definite (and indefinite) integration: methods and open issues

Daniel Lichtblau
Wolfram Research, Inc.
100 Trade Center Dr.
Champaign IL USA 61820
danl@wolfram.com

Abstract

The computation of definite integrals presents one with a variety of choices. There are various methods such as Newton-Leibniz or Slater's convolution method. There are questions such as whether to split or merge sums, how to search for singularities on the path of integration, when to issue conditional results, how to assess (possibly conditional) convergence, and more. These various considerations moreover interact with one another in a multitude of ways. Herein we discuss these various issues and illustrate with examples.

1 Introduction

In principle, algorithmic integration is fairly well understood. There are algorithms to handle various sorts of integrands and integration bounds. But, in the words of librettist Ira Gershwin, "It ain't necessarily so" [6], and actual practice is an altogether different situation. Troublesome areas include:

Determining (possibly conditional) convergence.

Recognizing and handling the cancellation of additive singularities.

Implementing intricate algorithms, e.g. some may need possibly deep recursion, or rely on nontrivial transformations.

Handling of parameters.

Working with legacy code.

In this report I will describe and illustrate many of the issues I have encountered in the process of working on the *Mathematica* [13] Integrate code base. As best I can tell, they also apply in part to other existing programs (though approaches to handling said issues may vary).

This report discusses the situation primarily as it evolved through *Mathematica* versions 5 and 6 development. I wish to thank Oleg Marichev (in honor of whose 60 *th* birthday I spoke on this topic) [11] for many discussions where he patiently tried to walk me through issues involved in convolution of `MeijerG` functions. I also thank Bhuvanesh Bhatt, a senior Software Quality Assurance engineer at Wolfram Research, for keeping me apprised of the situation regarding integration bugs in *Mathematica*, and for the countless hours he spent tracking and diagnosing them, and making test cases.

While we focus on particulars of *Mathematica* handling, I would be remiss in not pointing out that there is other work on this topic, some of which takes very different approaches to similar issues. In particular, see Davenport's synopsis of this same topic [4], and various articles authored or coauthored by Jeffrey on removal of path singularities [7, 8, 9]. I also thank David Jeffrey and an anonymous referee for numerous helpful comments and for pointing out several mistakes in an earlier draft.

2 Basic Structure of `Integrate` Code in *Mathematica*

2.1 Indefinite Integration

The indefinite integration code consists primarily of a partial implementation of the Risch algorithm [2, 5], in addition to extensive table lookup methods. The former handles the elementary integral cases that do not involve algebraic extensions, and a few simple extension cases of low degree. The latter tackles integrands with exponentials, trigonometrics and/or hyperbolics, elliptic integrals, and integrands involving special functions, particularly after the Risch methods have given up. The actual situation is slightly more complicated in that Risch code may handle part of an integrand, sending the rest to table lookup code. That in turn may do transformations and call back the integrate code recursively. The *Mathematica* implementation resorts to the Risch code early on, followed by extensive table code for handling of special functions as well as elementary function integrands that this implementation of Risch fails to integrate.

In addition to what is stated above, some integrand transformations are attempted prior to all else. The idea here is to recognize cases where one integrand may be converted to something that is easier to work with, but equivalent modulo a multiplicative constant (for example, transforming $\sqrt{x^2}$ to $x\left(\frac{\sqrt{x^2}}{x}\right)$, and treating the second factor as a differential constant).

Also useful are mathematically equivalent transformations such as factorization and partial fractions decomposition. An added wrinkle is that such transformations can invert one another, and it is often difficult to recognize which one will be beneficial to a given integrand. Thus one must take care to avoid attempting them blindly, and, in so doing, incurring the infinite wrath of the gods of recursion. In the *Mathematica* code base an attempt is made to determine, in various places, whether a transformation of the above type might be useful, and to apply it if so. It should be emphasized that this is, at best, a heuristic process (and at worse, a VERY recursive one...)

Last we mention that there is no intrinsic reason to split into integrands handled by Risch vs. those that are not. Another approach might be to use table lookup (e.g. via pattern matching) as a first step, followed by a Risch implementation, followed by further lookup. In order for this to be useful the first lookup would need to be fast (that is, would need to fail quickly when it will fail at all). A table-based approach that might be useful for this first step is described [10, 12].

2.2 Definite Integration

Definite integration is done via a number of methods as indicated below.

Special case contour integration.

Newton-Leibniz code (in brief: compute an indefinite integral, then evaluate it at endpoints and perhaps in limits at path singularities). There are various places where *Mathematica* will use this approach. The first such is specialized for integrands of the form rational $\times$ trig or rational $\times$ exponential.

Newton-Leibniz code specialized for integrands containing logs and/or polylogs. This code is more complicated than for the previously described case.

A general case implementation of Newton-Leibniz integration. Still more complicated code.

An implementation of definite integration by convolution of `MeijerG` functions [1]. This requires that we integrate from 0 to infinity. It also requires that the integrand be represented as a power of the integration variable times one or two `MeijerG` functions. Since a `UnitStep` function may be represented as a `MeijerG`, we may waive the semi-infinite range requirement whenever the integrand otherwise requires but one `MeijerG`. This method in particular uses several transformation tactics to handle algebraics, trigs, logs, exponentials, and so forth. Hence it has its own tree of specialized code subsets.

The overall implementation is best described as a polyalgorithm that calls on any or all of the above in various ways, depending on heuristics (or perhaps the mood of the little man inside the code).

2.3 Brief Descriptions and Examples of the Two Primary Methods of Definite Integration

2.3.1 Newton-Leibniz

First we find an antiderivative to the integrand. We then try to assess from the integration path whether there are candidate points of discontinuity for integrand or antiderivative (e.g. due to branch cuts), or places where the integrand may spike to infinity. These we collectively refer to as path singularities. We treat each point in one of two ways (based on crude heuristics): either issue a proviso on parameters that guarantee the point is not a "bad" point, or else split the integration path at that point and use limits as we approach from either side. Once the integration path is split into suitable segments we take appropriate limits of the antiderivative as we approach the segment endpoints from appropriate directions, then sum these signed values to get the resulting definite integral. In cases where we have infinite spikes, we attempt to determine whether they are individually integrable.

Here is a classical example that incorrectly gave zero in a prior version of *Mathematica*. We integrate $1/x$ around a closed contour enclosing the origin. The syntax below indicates we are to use a segment-by-segment traversal, where segments connect the given path points. Note that one path, from $-1+i$ to $-1-i$, must be split where it crosses the (negative) real axis. This is due to the fact that the antiderivative is the logarithm function, and *Mathematica* uses the canonical choice of placing the branch cut along the negative real axis. (As a referee pointed out, a path split for a branch cut crossing must happen *somewhere*, since we integrate around a loop and would obtain zero without such a split.)

$$\textbf{Integrate}[1/z, \{z, 1+I, -1+I, -1-I, 1-I, 1+I\}]$$

$2i\pi$

2.3.2 Slater Convolution via MeijerG Products

The idea in this case is to make suitable transformations so that we have an integrand in $f(z)$ is expressed as a product of the form $z^k \text{MeijerG}\left[\textit{expression}_1\right] \text{MeijerG}\left[\textit{expression}_2\right]$ where we integrate from 0 to infinity. While this path may seem rather specialized, bear in mind that translations and rotations can move a path to the real axis, and we can represent `UnitStep` as a `MeijerG`

function. (Specifically, for $x > -2$ we have $\theta(x) = G^{0,1}_{1,1}\left(x+1 \middle| \begin{matrix} 1 \\ 0 \end{matrix}\right)$) This means we can treat a finite interval on the positive axis as though it were semi-infinite, at the cost of a `MeijerG` factor. The basic convolution theorem is discussed in [1].

Several common functions such as trigs, various exponentials, Bessel functions, and certain common algebraics e.g. $(a + bx^n)^m$ may be represented as `MeijerG` functions, while others may be brought to `MeijerG` form after some suitable transformation of variables. Hence we look for viable transformations and then do a table driven conversion in an effort to obtain the desired form.

Once in the factored form above (or perhaps a simpler form wherein any factor or factors are not present) we go into convolution code, at the heart of which lies a function that formulates the integrated result as a new `MeijerG` expression via the Slater theorem. It then performs some manipulations in an effort to recast in terms of better known functions.

Here is a simple example.

val = Integrate [BesselJ[2, x]$x^2 e^{-x+2}$, {x, 0, ∞}]

$\frac{3e^2}{4\sqrt{2}}$

This operated by first converting the integrand to an "inert" form representing the integrand product as $x^2\text{MeijerG}[\{\{\},\{\}\},\{\{0\},\{\}\},x]\text{MeijerG}\left[\{\{\},\{\}\},\{\{1\},\{-1\}\},\frac{x^2}{4}\right]$

3 Software Engineering Problems Associated with this Body (Corpus? Corpse?) of Integration Code

Below are several of the issues that confronted me in the process of fixing bugs in *Mathematica*'s symbolic integration. I suspect many are found in any general purpose symbolic integration code base.

Different modules, written by different people, were often not on the best of speaking terms. Hence various pieces of functionality of potentially general use were reinvented. In some cases this had bad consequences beyond code bloat, such as inconsistencies of bugs, different unstated underlying assumptions, and so on.

Much was legacy code, written over a period of about 15 years, by several different people. It remains spread over several dozen source code files (which is good, to the extent that it improves modularity). No one person ever understood all of it, and some parts are no doubt to this day not understood by anyone−−this is one of the pitfalls of legacy code. I find it unlikely that, in an area of this scope, a significant body of code will be entirely understood by the team that develops it.

Some of the steps used are quite fragile. Specifically they may be sensitive to small changes in seemingly unrelated functions such as `Together` (providing a "canonical" form to rational functions), `Apart` (a partial fractions decomposition, which has some ill-defined behaviors tied to needs of `Integrate`), `Factor`, `Solve`, `Simplify`, and others. After encountering this problem in many forms I have come to conclude that it really is endemic to symbolic integration and not simply an artifact of implementation details. The entire idea of working with transformations of integrands, coupled with the mathematical impossibility of creating canonical forms for all possible expressions, and the difficulty of forcing transformations to be improvements for a given purpose, make this a thorny issue. Indeed, certain types of standard tactics such as integration by parts or use of l'Hospital's rule for limit extraction can actually lead to infinite recursion if not done with considerable caution.

There are tradeoffs to be made between speed and power. One wants to try certain transformations, for example, in order to handle certain classes of problem. But there is no plausible way in general to delimit cases that may hang in the transformation process. Hence some form of delimiting is required e.g. by time or operation count.

Many classes of integration problem can make use of assumptions regarding, say, behavior of the integrand at infinity. Some parts of the legacy code were not everywhere cognizant of such assumptions, and may have made contradictory ones (which, not surprisingly, was a cause of may bug reports). And even when code appropriately tries to determine e.g. behavior at infinity via given assumptions, there is the issue noted above, where, say, limit extraction may hang if not suitably constrained. But when constraints on time, memory, or some measure of operation count are used, how is one to handle an aborted intermediate result? With a warning message? By giving up? By continuing as though it had been a "good" case e.g. of convergence at infinity?

Symbolic integration is one of the most complicated pieces of machinery to be found in algorithmic mathematics. The reliance on other functions as well as intricacies of exploring possible integrand transformation certainly implies that it will have a nontrivial implementation. Hence it is vital that the pieces be adequately documented. This is a rule throughout the field of software development, but the importance cannot be overstated in the context of mathematical integration (it is of course important for code integration as well).

When I began work on this body of code, more than one out of every four open bugs in the *Mathematica* kernel (with around 2000 functions) was in the category of definite integration (a part of one function, Integrate). This mass posed several problems in and of itself. First, the function clearly receives widespread usage, and hence it is difficult to start from scratch. Second, the scope made it a bit difficult to perform adequate triage (that is to say, it is hard to see the forest for the trees). Third, the massive overhaul needed strongly implied that there would be considerable breakage, at least in the short term. Fourth is the likelihood that the scope of trouble exceeds the capabilities of any one developer−−and I can attest that it certainly exceeds the capabilities of this particular developer.

4 Issues in the Implementation of Indefinite Integration

We will discuss in brief some issues that tend to be specific to indefinite integration. As our primary focus is on definite integration we defer to the next section those that are common to both.

4.1 The Curse of Recursing

Often an integral may be broken into two parts. I (facetiously) claim that the technical term for the first part is "done" while that for the second is "not done". The second term might be further rewritten in ways that return one to the original problem or a variant thereof, thus leading to recursive splitting. As an example of an integrand that might elicit such behavior $\int \frac{\sin(x)}{(x+1)\sqrt{a-x}}\,dx$ would go into deep recursion in some older versions of *Mathematica*. We fixed this by using hashing to recognize integrands that have previously been attempted in a given part of the code.

Another common cause of descent into the infinite is the utilization of pairs of inverse transformations. One code section might, for example, convert a trig to exponentials, while a subsequent handler might convert back to trigs. We try to avoid this pitfall by using *Mathematica*'s `Block` scoping construct to (what else?) block the dual handler when we do one such transformation.

There is a bright side. After substantial bug fixing, especially in pattern matching parts of the Integrate code, *Mathematica* handles more problems than in the past. Here is an example that works well due to several of transformations of the integrand. It comes from an integration test suite in [3].

$$\int \frac{\mathbf{Tan}[x]^2}{\sqrt{\left(1-\frac{\mathbf{Sin}[x]^2}{2}\right)\left(1-\frac{2\mathbf{Sin}[x]^2}{5}\right)}}\,dx$$

$$-\left(4i\sqrt{\tfrac{5}{3}}\mathrm{Cos}[x]^2\left(\mathrm{EllipticE}\left[i\mathrm{ArcSinh}\left[\sqrt{\tfrac{3}{5}}\mathrm{Tan}[x]\right],\tfrac{5}{6}\right]-\mathrm{EllipticF}\left[i\mathrm{ArcSinh}\left[\sqrt{\tfrac{3}{5}}\mathrm{Tan}[x]\right],\tfrac{5}{6}\right]\right)\right.$$
$$\left.\sqrt{(4+\mathrm{Cos}[2x])\mathrm{Sec}[x]^2}\sqrt{2+\mathrm{Tan}[x]^2}\right)/\left(\sqrt{25+14\mathrm{Cos}[2x]+\mathrm{Cos}[4x]}\right)$$

4.2 Out on a Limb with a Cut Branch

Frequently we require transformations that bring into play multivalued functions. The consequence is that we may arrive at an antiderivative that is only correct up to a piecewise multiplicative constant (i.e. caused by a transformation that is 1 in some places and -1 in others, such as $\sqrt{x^2} \to x$). Hence we now attempt to restore the proper factor. This is not always straightforward and often leads to a significantly enlarged form of result. Moreover it is not trivial to recognize in all cases how to correctly reverse effects of such transformations.

4.3 Transformations

In many cases it is well understood how one might obtain an antiderivative for a particular class of integrand e.g. rational functions of trigonometrics. But the needed transformations must be applied carefully in order to avoid potential explosion in intermediate complexity or that of the final result. An example of this would be $\int \frac{x}{\sin^2(2x)+\cos(x)}\,dx$. Version 4 of *Mathematica* gives a result about five times larger than what is obtained with more recent vintages.

4.4 To Expand or Not Expand, Indefinitely

This is a (very important) special case of a transformation. As the issues are a subset of those that arise in the context of definite integration we cover it there instead.

5 Issues in the Implementation of Definite Integration

In this section we arrive at the main focus of this report. Within symbolic definite integration one encounters a wide array of issues. In the subsections below I will endeavor to present and illustrate many of them.

5.1 To Expand or Not Expand, Definitely

When one has certain types of input it makes sense to expand over summands and integrate term-by-term. In other cases this can be a very bad thing to do. So the question is when should one expand the input and loop over summands. We will illustrate with a few examples.

Here is a case where one must not expand. The reason is that the summands have cancelling singularities at infinity (there is also an infinitely oscillatory, but nevertheless integrable, part at the origin).

$$\mathbf{ii} = \int_0^\infty \left(e^{\frac{i}{x}} - 1\right)\mathbf{Sin}[x]\,dx$$

$-1 + \text{BesselK}[1,2] + \frac{1}{2}i\pi(\text{BesselJ}[1,2] + i\text{BesselY}[1,2])$

Clearly we cannot expand and handle the summands separately. We attempt to do so below, using *Mathematica*'s `Map` to map the integration over separate summands.

```
Map[Integrate[#, {x, 0, Infinity}]&, Expand[(E^(I/x) - 1) Sin[x]]]
```

Integrate::idiv : Integral of Sin[x] does not converge on $\{0, \infty\}$. More...
Integrate::idiv : Integral of $e^{\frac{i}{x}}$Sin[x] does not converge on $\{0, \infty\}$. More...

$\int_0^\infty -\text{Sin}[x]\,dx + \int_0^\infty e^{\frac{i}{x}}\text{Sin}[x]\,dx$

By contrast, here is a case where preexpansion helps considerably.

```
integrand = (a c Cos[t])/(g s) + (b q Cos[2 t])/(c f) + (c Cos[3 t])/(a d) + (d f Cos[4 t])/(a h n) + (e q Cos[5 t])/(a g) +
 (f l Cos[6 t])/(m r w) + (b g Cos[7 t])/(n o x) + (h Sin[t])/(b c) + (i Sin[2 t])/(e h r) + (j y Sin[3 t])/(l p) + (d k Sin[4 t])/(c j) +
 (a l m Sin[5 t])/(b f h s) + (m p Sin[6 t])/(j k) + (n q Sin[7 t])/(c x);
Timing[Integrate[integrand, {t, 0, 2 Pi}]]
```

$\{1.\text{Second}, 0\}$

Versions of *Mathematica* that did not expand but instead tried to work with the entire integrand at one time took about a minute to handle the same integration. The reason is that transformations involving several trigonometric summands can be costly both in time and memory.

Yet another drawback to expansion is that the individual pieces might have different ideas regarding provisos for parameters, in particular if they go through different routes in the code. If conflicting provisos emerge we might either get a useless result (if we fail to recognize that the conditions cannot all hold) or else be forced to redo the integral, thus having wasted time processing summands individually.

The upshot to this subsection is that the question of whether or when to expand can be tricky.

5.2 Heuristics Involving Method Choices and Transformations of Integrand

Quite often one encounters an integrand that might be handled in different ways. As the result can vary considerably in terms of speed and/or complexity it is a (wide) open problem to optimally dispatch to appropriate subroutines based on the structure of the integrand. There is one nice feature to the decision process of whether or not to try an approach based on evaluating an antiderivative at endpoints: we must first find that antiderivative. If we fail at that stage, we certainly know it is not a viable approach!

Here is an example where use of the convolution approach gives a result faster than computing an antiderivative and evaluating at the endpoints via limits. While we do not explicitly show the latter method, the reason for the speed difference is that the limit extraction at infinity is quite slow.

```
Timing[Integrate[1/x^8 Sin[x] Sin[x/3] Sin[x/5] Sin[x/7] Sin[x/9] Sin[x/11] Sin[x/13] Sin[x/15], x]]
```

$\{18.12\text{Second}, \frac{46780792471344073869653786446 9\pi}{189651671721241513514111035029375000 0}\}$

Here, in contrast, is an example where evaluation of an antiderivative a la Newton-Leibniz gives a preferred form of result.

Integrate[Sin[$x - y$], {y, x, Pi}, Assumptions → {$0 < x <$ Pi}]

$-1 - \mathrm{Cos}[x]$

As a general remark, one would of course prefer that either method (when both are applicable) give the nicer forms of results in the examples above. It is an unfortunate fact that this state of affairs is quite difficult to achieve. Thus one must use heuristics to distinguish cases that might receive more favorable treatment for one or the other method. When one notes that the integrands in these examples are both primarily trigonometric functions of linear functions of the integration variable, it becomes clear that formulation of good heuristics is by no means trivial.

A related issue of heuristics is that frequently an integrand may appear amenable to any of several different transformations. For example, for a quadratic radical function of the integration variable we might want to transform to a `MeijerG` function, or do a linear change of coordinates in an effort to simplify or remove the radical.

Another related issue is that some transformations may not be valid for all possible values of a parameter. For example, one might wish to write $\left(a + bx^d\right)^n$ as $a^n \left(\frac{bx^d}{a} + 1\right)^n$ (for integration variable x) in order to convert to a `MeijerG` form. This transformation is of course not valid for all values of the parameter a. We then must choose between issuing a conditional answer, or abandoning this approach and trying another that might give an unconditional result.

Still another related issue is in how to transform products into `MeijerG` functions. There may be several possibilities, with the quality of outcome dependent on the choices made.

5.3 Generation of Provisos (Results That Depend on Conditions)

Below I indicate ways in which generation of conditions can be problematic. Several are illustrated with older versions of *Mathematica* because we have made improvements that render the specific examples obsolete. But the ideas behind them are general and no doubt related problems still lurk in the recent versions. Given the long-term propensity of this technology to cause trouble I was tempted to call the subsection "Generations of Generation of Provisos".

5.3.1 Excessive Conditions

Various algorithm implementations may force generation of unneeded conditions. For example, `MeijerG` convolution will require that certain values lie in "wedges" emanating from the origin, and Newton-Leibniz methods may issue conditions based on the specific form of the antiderivative.

Here is a simple example using version 5.0. Special case code for handling exponentials via convolution wants to insist that a parameter take a negative real part.

Integrate $\left[e^{-\frac{a(x-b)^2}{\mathrm{sigma}^2}}, \{x, -\infty, \infty\}, \mathrm{Assumptions} \to \{a > 0, \mathrm{sigma} \in \mathrm{Reals}\}\right]$

$$\mathrm{If}\left[\mathrm{sigma} \neq 0 \&\& \mathrm{Re}[b] < 0, \frac{\sqrt{\pi}\,\mathrm{Abs}[\mathrm{sigma}]}{\sqrt{a}}, \mathrm{Integrate}\left[e^{-\frac{a(b-x)^2}{\mathrm{sigma}^2}}, \{x, -\infty, \infty\}, \mathrm{Assumptions} \to \mathrm{sigma} \in \mathrm{Reals} \&\& a > 0 \&\& (\mathrm{sigma} == 0 \| \mathrm{Re}[b] \geq 0)\right]\right]$$

The current behavior is nicer. Notice a superfluous proviso is no longer present.

Integrate $\left[e^{-\frac{a(x-b)^2}{\mathrm{sigma}^2}}, \{x, -\infty, \infty\}, \mathrm{Assumptions} \to \{a > 0, \mathrm{sigma} \in \mathrm{Reals}\}\right]$

$$\text{If}\left[\text{sigma} \neq 0, \frac{\sqrt{\pi}\text{Abs[sigma]}}{\sqrt{a}}, \text{Integrate}\left[e^{-\frac{a(b-x)^2}{\text{sigma}^2}}, \{x, -\infty, \infty\}, \text{Assumptions} \to \text{sigma} == 0\right]\right]$$

5.3.2 Necessity of Proviso Generation

Often one may wish to avoid generating conditions. This happens for example when we know in advance that they will be satisfied by parameter values we may later choose. A second reason is that some integration problems become substantially slowed by the process of attempting to find appropriate provisos for parameter values.

A problem with avoiding generation of provisos is that results can depend on what route is taken in the code. If the integration is split into parts, say because the contour is split, and if the parts go through code that makes different implicit assumptions about parameter values, then results might be entirely incorrect. One common manifestation of this defect is incorrect cancellation. We illustrate with the preceding example. The code handles half the integration path using one assumption about the parameter b, handles the other half range using a contrary assumption, and the end result is quite wrong.

$$\textbf{Integrate}\left[e^{-\frac{\boldsymbol{a(x-b)^2}}{\textbf{sigma}^2}}, \{\boldsymbol{x}, -\boldsymbol{\infty}, \boldsymbol{\infty}\}, \textbf{GenerateConditions} \to \textbf{False},\right.$$
$$\left.\textbf{Assumptions} \to \{\boldsymbol{a} > \boldsymbol{0}, \textbf{sigma} \in \textbf{Reals}\}\right]$$

0

The upshot is that partial results cancelled, with no indication that they require conflicting conditions to hold.

We emphasize at this point that user assumptions on parameter ranges can be quite helpful for determining convergence and/or a need for provisos. This in turn can make some integrations proceed much faster than might otherwise be the case: convergence and proviso assessment are frequent bottlenecks in symbolic definite integration.

5.3.3 Genericity of Generated Conditions

Conditions generated may be only generically correct. Here is an example that exhibits this phenomenon. Specifically, when `Im`$[a]$==`Im`$[b]$ (that is, a vertical integration path) we have a problem because their difference appears in some denominators.

$$\boldsymbol{\int_a^b \textbf{Log}[\boldsymbol{x}]\, d\boldsymbol{x}}$$

$$\text{If}\left[\frac{\text{Im}[b]}{\text{Im}[a]-\text{Im}[b]} \geq 0 \left\| \frac{-\text{Im}[b]\text{Re}[a]+\text{Im}[a]\text{Re}[b]}{\text{Im}[a]-\text{Im}[b]} \geq 0 \right\| \frac{\text{Im}[a]}{\text{Im}[a]-\text{Im}[b]} \leq 0, a-b-a\text{Log}[a]+b\text{Log}[b], \text{Integrate}[\right.$$
$$\left.\text{Log}[x], \{x, a, b\}, \text{Assumptions} \to \text{Re}[b] < \frac{\text{Im}[b]\text{Re}[a]}{\text{Im}[a]} \&\& ((\text{Im}[a] > 0 \&\& \text{Im}[b] < 0) \| (\text{Im}[b] > 0 \&\& \text{Im}[a] < 0))\right]\right]$$

5.4 Assessment of Convergence

Testing for convergence is closely related to generation of conditions insofar as the former can depend on parameter values. But convergence testing, even in the absence of symbolic parameters, is no easy matter (it is complicated by oscillatory factors, possible cancellation of singular terms, and so forth). Hence code that tests integrands for convergence tends to be of an ad hoc nature. While it usually serves well it is by no means a science at this time, and I do not know how to make it one. Were one to issue "uncertain of convergence" messages in all possible cases where we cannot verify convergence or identify conditions to generate that would ensure convergence, we would have

a flood of messages. Even worse, we would lose many integrals. The reason is that summands can separately spawn conflicting conditions, and this can happen in cases when they should instead cancel singularities in pairs.

Here are some simple examples that may help to give an idea of the difficulties lurking within convergence assessment. *Mathematica* will evaluate them correctly but this was not always the case.

$$\int_0^\infty \frac{\text{Cos}[x^2]}{\text{Log}[x]}\,dx$$

Integrate::idiv : Integral of $\frac{\text{Cos}[x^2]}{\text{Log}[x]}$ does not converge on $\{0, \infty\}$.

$$\int_0^\infty \frac{\text{Cos}[x^2]}{\text{Log}[x]}\,dx$$

$$\int_0^\infty e^x \text{Sech}[ax] x\,dx$$

$$\text{If}\left[\text{Re}[a] > 1, \frac{-\text{PolyGamma}\left[1, \frac{3}{4} - \frac{1}{4a}\right] + \text{PolyGamma}\left[1, \frac{-1+a}{4a}\right]}{8a^2}, \right.$$
$$\left.\text{Integrate}\left[e^x x \text{Sech}[ax], \{x, 0, \infty\}, \text{Assumptions} \to \text{Re}[a] \le 1\right]\right]$$

The first diverges, but not due to problems at either endpoint (there is a pole at 1). The second converges conditionally, and version 4 of *Mathematica* even knows this, but then gets the condition wrong.

Below is an example that gives an idea of what is involved in sorting out conditions for convergence. We need to look at possibilities of oscillatory, exponentially growing, and exponentially damped factors in order to figure out the correct conditions on parameters. In this case the original assumption suffices to guarantee convergence.

$$\text{Integrate}\left[\frac{\text{Sin}[\beta r]\text{Sin}[\gamma r]}{e^{\alpha r^2}}, \{r, -\infty, \infty\}, \text{Assumptions} \to \text{Re}[\alpha] > 0\right]$$

$$\frac{\left(e^{-\frac{(\beta-\gamma)^2}{4\alpha}} - e^{-\frac{(\beta+\gamma)^2}{4\alpha}}\right)\sqrt{\pi}}{2\sqrt{\alpha}}$$

Here is a slightly more difficult variant, one that requires nontrivial conditions on parameters beyond what is given in the assumptions option.

$$\text{Integrate}[(\text{Sin}[\beta * r\text{\^{}}2] * \text{Sin}[\gamma * r\text{\^{}}2])/E\text{\^{}}(\alpha * r\text{\^{}}2), \{r, -\text{Infinity}, \text{Infinity}\},$$
$$\text{Assumptions} \to \text{Re}[\alpha] > 0]$$

$$\text{ConditionalExpression}[$$
$$\left(\sqrt{\pi}\left(\alpha\left(\frac{\alpha^2+(\beta+\gamma)^2}{\alpha^2}\right)^{3/4}\text{Cos}\left[\frac{3}{2}\text{ArcTan}\left[\frac{\sqrt{(\beta-\gamma)^2}}{\alpha}\right]\right] -\right.\right.$$
$$\alpha\left(\frac{\alpha^2+(\beta-\gamma)^2}{\alpha^2}\right)^{3/4}\text{Cos}\left[\frac{3}{2}\text{ArcTan}\left[\frac{\sqrt{(\beta+\gamma)^2}}{\alpha}\right]\right] +$$
$$\sqrt{(\beta-\gamma)^2}\left(\frac{\alpha^2+(\beta+\gamma)^2}{\alpha^2}\right)^{3/4}\text{Sin}\left[\frac{3}{2}\text{ArcTan}\left[\frac{\sqrt{(\beta-\gamma)^2}}{\alpha}\right]\right] -$$
$$\left.\left.\left(\frac{\alpha^2+(\beta-\gamma)^2}{\alpha^2}\right)^{3/4}\sqrt{(\beta+\gamma)^2}\text{Sin}\left[\frac{3}{2}\text{ArcTan}\left[\frac{\sqrt{(\beta+\gamma)^2}}{\alpha}\right]\right]\right)\right) \Big/$$
$$\left(2\alpha^{3/2}\left(\frac{\alpha^2+(\beta-\gamma)^2}{\alpha^2}\right)^{3/4}\left(\frac{\alpha^2+(\beta+\gamma)^2}{\alpha^2}\right)^{3/4}\right), \text{Re}[\alpha] \ge \text{Abs}[\text{Im}[\beta]] + \text{Abs}[\text{Im}[\gamma]]\Big]$$

As another example we show an integral that converges. The integrand is a sum of two terms, each of which separately will diverge. If the code splits this then it must recognize that there will be a cancellation of singular parts.

Integrate[(Log[x] − Log[a])/(x − a), {x, 0, a}, Assumptions → a > 1]

5.5 Difficulties Involving Parameters and Detection of Singularities

One issue is in finding singular points on the integration path. Even something as simple as a trig function can be problematic. And there are other tar pits lurking beneath the surface of the swamp.

5.5.1 Parametric Singularities

The presence of elliptic functions in the antiderivative usually removes any hope of correctly detecting parameter dependent singularities. Below is an example.

$$\int_0^{\frac{\pi}{2}} \left(-\left(m\,\text{MeijerG}\left[\{\{1,1\},\{\}\},\left\{\left\{\tfrac{1}{2},\tfrac{3}{2}\right\},\{\}\right\},\tfrac{1}{1-m}\right]\right)+\pi\,\text{EllipticK}[m]\left(1+(-1+m)\text{Sin}[y]^2\right)\right)\Big/ \left(\pi\sqrt{1+(-1+m)\text{Sin}[y]^2}\right)dy$$

If[Im[m] ≠ 0||Re[m] ≥ 0, EllipticK[1−m](EllipticE[m]−EllipticK[m])+EllipticE[1−m]EllipticK[m], Integrate $\left[\frac{\text{EllipticK}[m]}{\sqrt{1+(-1+m)\text{Sin}[y]^2}}-\frac{m\text{MeijerG}\left[\{\{1,1\},\{\}\},\left\{\left\{\frac{1}{2},\frac{3}{2}\right\},\{\}\right\},\frac{1}{1-m}\right]}{\pi\sqrt{1+(-1+m)\text{Sin}[y]^2}}-\frac{\text{EllipticK}[m]\text{Sin}[y]^2}{\sqrt{1+(-1+m)\text{Sin}[y]^2}}+\frac{m\text{EllipticK}[m]\text{Sin}[y]^2}{\sqrt{1+(-1+m)\text{Sin}[y]^2}},\left\{y,0,\frac{\pi}{2}\right\},\text{Assumptions}\to m<0\right]\right]$

Whether the result is worth the screen real estate it occupies is open to debate.

5.5.2 Transcendentals and Singularity Detection

Often finding singularities depends on finding roots of expressions. These need not be algebraic, and a general purpose transcendental root finder is a nontrivial undertaking (also it might be slow). In the example below, we find the bad point at the origin, and correctly decide the integral is divergent.

$$\int_{-\frac{1}{2}}^{\frac{1}{2}} \frac{-2+x^2+20\text{Cos}[x]}{(-6x+x^3+60\text{Sin}[x])^2}\,dx$$

Integrate::idiv : Integral of $\frac{-2+x^2+20\text{Cos}[x]}{(-6x+x^3+60\text{Sin}[x])^2}$ does not converge on $\left\{-\frac{1}{2},\frac{1}{2}\right\}$.

$\int_{-\frac{1}{2}}^{\frac{1}{2}} \frac{-2+x^2+20\text{Cos}[x]}{(-6x+x^3+60\text{Sin}[x])^2}\,dx$

But minor modifications give rise to a singularity that is not at an algebraic number nor at a "convenient" multiple thereof (e.g. π times a rational).

$$\text{ii} = \int_1^5 \frac{-2+x^2+20\text{Cos}[x]}{-6x+x^3+60\text{Sin}[x]}\,dx$$

$\frac{1}{3}(-\text{Log}[-5+60\text{Sin}[1]]+\text{Log}[95+60\text{Sin}[5]])$

But this integral actually diverges. The denominator has poles on the integration path, and they are roots of a transcendental equation. Recent versions of *Mathematica* will detect many cases of this sort, but certainly not all such.

Solve [−6x + x^3 + 60Sin[x] == 0&&1 < x < 5, x]

$\left\{\left\{x\to \text{Root}\left[\left\{60\text{Sin}[\#1]-6\#1+\#1^3\&, 3.5334931463518126574\right\}\right]\right\},\right.$
$\left.\left\{x\to \text{Root}\left[\left\{60\text{Sin}[\#1]-6\#1+\#1^3\&, 4.3462608083358731270\right\}\right]\right\}\right\}$

The fact that these particular singularities are now detected means that recent *Mathematica* versions will correctly assess the divergence in the preceding example.

5.5.3 Parametrized Singularities in Multiple Integrals

Parametrized singularities can cause trouble, particularly in multidimensional integration. Here is an example from [12]. We begin with a small variation on the actual problem.

```
Integrate[Abs[Sin[x - y]], {x, 0, Pi}, {y, 0, Pi}]
```

2π

Now we double the region of integration in both directions.

```
Integrate[Abs[Sin[x - y]], {x, 0, 2 Pi}, {y, 0, 2 Pi}]
```

4π

The second one is off by a factor of 2. The problem is that one singularity line, $x == y$, is recognized. But the broken line $x == y + \pi$ modulo 2π goes unrecognized. Hence we do not handle ranges correctly, splitting only into two rather than three pieces in the first level of integration. Note that this is in version 5.0 of *Mathematica*; later versions do get the correct value of 8π.

5.5.4 Branch Cuts Intersecting the Integration Path

It is important to assess whether an integration path crosses a branch cut of an antiderivative (so that we might split the path into segments). The example below does this in order to get the correct result.

```
Integrate[Log[x], {x, -1 - I, -2 + I}]
```

$\frac{1}{4}\left((4-8i)-(1-i)\pi+(4+8i)\text{ArcTan}\left[\frac{1}{2}\right]+(1+i)\text{Log}[4]-(4-2i)\text{Log}[5]\right)$

Version 4 of *Mathematica* failed to catch the crossing and gave a result with imaginary part off by 3π. Needless to say, this problem becomes vastly more difficult if the crossing might or might not exist based on parameter values. One might try the example below to get an indication of what might be a reasonable result.

```
Integrate[Log[x], {x, -1 - I, y}]
```

5.5.5 Problems with Algebraic Manipulation

Here is an example that was quite wrong prior to version 5.1 (we show the result from version 5.0).

```
ii = Integrate[2 y E^y BesselK[0, Sqrt[3] y Sign[y]], {y, 0, Infinity}]
N[ii]
NIntegrate[2 y E^y BesselK[0, Sqrt[3] y Sign[y]], {y, 0, Infinity}]
```

$1-\frac{\text{ArcSec}\left[\sqrt{3}\right]}{\sqrt{2}}$

0.324489

2.54593

The problem was in the innards of the convolution code. At a key step we made use of an exponential of the form $\exp\left((-2i)\pi - \log\left(\frac{4}{3}\right) + 2\log(2)\right)$ and we subsequently required a square root. Once we replace exp with *Mathematica*'s `Exp` function we lose track of a factor of (-1) and this gave rise to a bad result. While this example is now repaired, the general problem remains of how to find and forestall this phenomenon.

Mathematica's integration code once had an interesting method for eluding this branch cut sort of problem. The tactic was to convert various exponential things to trigs in the hope that anything "bad" e.g. explicitly complex values, would pack up for vacation. Often this worked, and indeed I introduced some bugs simply by disabling some of this and making conversions in the reverse direction. But my feeling is that the problem should be addressed at a more basic level, by finding the culprits and spanking the branch cuts out of them. As an indication that this earlier method had severe limitations, I note that it too failed on the example above.

The next examples have bugs in current versions due to branch cut issues not correctly handled in convolution code. They show that not all changes can be viewed as progress, insofar as they gave correct results in version 4.

$$\textbf{Integrate}\left[e^{-ixt}\left(e^{\frac{1}{3}(-i)x^3}-1\right)x^{-\frac{3}{2}},\{x,0,\infty\},\textbf{Assumptions}\to t>0\right]$$

$$(1+i)\sqrt{2\pi}\sqrt{t}+\frac{(-1)^{7/12}\pi\,\text{HypergeometricPFQ}\left[\left\{-\frac{1}{6}\right\},\left\{\frac{1}{3},\frac{2}{3}\right\},\frac{t^3}{9}\right]}{3^{2/3}\text{Gamma}\left[\frac{7}{6}\right]}-\frac{(-1)^{11/12}\pi t\,\text{HypergeometricPFQ}\left[\left\{\frac{1}{6}\right\},\left\{\frac{2}{3},\frac{4}{3}\right\},\frac{t^3}{9}\right]}{3^{1/3}\text{Gamma}\left[\frac{5}{6}\right]}$$

A numerical check will reveal that this is simply not correct.

The next example does not even involve a symbolic parameter. Again, the culprit deep down is in convolution of `MeijerG` functions.

```
ii = Integrate[ArcSin[v] * Log[v]/v, {v, 0, 1}]
N[ii]
NIntegrate[ArcSin[v] * Log[v]/v, {v, 0, 1}]
```

$\frac{1}{4}\pi\text{Log}[2]$

0.544397

−1.02331

The second of these examples has been fixed in recent versions, but the first remains as a festering reminder of the elusive nature of perfection in symbolic integration (or perhaps of the not terribly elusive imperfection of the author).

5.6 Handling Cancellation of Integral Singular Parts

Some integrals are well defined but may be written as sums of improper (e.g. divergent) integrals. In some cases it may be possible to transform them into a form that is explicitly well behaved. In other (unfortunate) cases this cannot be done, and they must be handled as sums, where we know there is cancellation. This raises a very difficult problem, of how to figure out what is the correct "finite" part, and what is the singular part, of each summand.

It turns out that some methods will almost automatically do this. Convolution methods often give results that have removed their singular parts. Newton-Leibniz methods often give antiderivatives where we can quikly see what part must be removed in order to obtain a finite result. We mention two difficulties.

(1) Sometimes there are singular parts involving logarithms. In such cases it is not obvious what is the finite part. For example, if we have $\log(2x)$ as x goes to infinity, are we to discard the entire thing, or split it into $\log(2)+\log(x)$ and only discard the latter term?

(2) If we need to split an integral e.g. because summands are amenable to different means of handling, or because there is a need to split a path, then different methods might get used for different parts. This can be problematic, because it turns out that different methods may have different ideas of what exactly is the finite part vs. the singular part. So when finite parts are combined we may still have an incorrect result, because we did not correctly cancel singular parts.

This is an open area for further research.

5.7 General Considerations

Here are some of the questions that require thought in the design and implementation of definite integration. They have emerged from study and overhaul of our existing code base, but I believe they apply more generally e.g. to definite summation and related polyalgorithm computational calculus.

Given a choice of methods, which should one attempt first? This can have serious repercussions in terms of speed. For example, what might be a fast `MeijerG` convolution can be very slow to evaluate as an indefinite integral followed by extraction of limiting values. And of course the opposite can happen. Or the methods might be comparable in speed but give results of vastly different complexity of form.

When or how should one presimplify the input?

When or how should one simplify the result?

Special case methods may be very helpful for speed or simplicity of result. But they also proliferate opportunities for bugs. Hence it requires care to think through what classes are important to handle in these ways.

Some methods require intrinsically "slow" technology. For example, refinement of conditions (which is sometimes essential in order that they not blow up) may require some level of CAD support behind the scenes. Even limit extraction for Newton-Leibniz methods can be slow. We are thus faced with questions of when to apply such technology and how to prevent it from causing many inputs to hang.

In regard to prevention of hanging in computationally intensive technology noted above, we have found it quite necessary to place time constraints on certain pieces of code. (Motivation: often they succeed. If they fail, so be it, and we then try other things.) This gives rise to a new set of problems. One is that asynchronous interrupt handling, required by `TimeConstrained`, is imperfect and in rare cases will cause a kernel crash. Another is that results now take on a platform dependent nature, and this is seriously unsettling. A possible future direction that will alleviate this: have potentially slow code stopped by some measure of operation count rather than asynchronous interrupts.

Indefinite integration is (generally) more powerful in version 5 than in the past. This has come at the price of speed: it simply tries more transformations and the like. This in turn means all Newton-Leibniz code is at risk of getting slower. We now cache some results from indefinite integration but this is at best a partial fix to the problem.

Provisional moral, or maybe conundrum: The more things improve, the more opportunity for closely related things to deteriorate.

6 Summary and Directions for Future Work

Development and overhaul of `Integrate` has had successes and at least some setbacks. One should hope these latter may prove to be temporary.

Among the successes, virtually all infinite recursion problems from earlier versions has been fixed. Cases where a method failed and we did not move on to try another method, that is, premature bailout, have likewise been addressed. Numerous problems in convergence assessment and singularity detection have likewise been fixed. Much of the idiosyncratic ad hoc code has been removed or rewritten. Most open problems fall into the various categories described above, which, while large, is better understood than the integration swamp that we had four years ago.

This has come at a cost of speed. Moreover we now have some level of platform dependent behavior. Work remains to iron out bugs in convolution code and elsewhere. Heuristics to determine use of expansion, simplification, refinement of conditions, and the like are crude and certainly imperfect. Modularity of the code, which affects usability, maintaince and further development, has improved but is likewise still far from perfect. The same may be said for its documentation.

This presents a synopsis of the issues faced in development of a robust definite integration. To some extent will also give a snapshot of the recent status of `Integrate` in *Mathematica*. Lest the reader be left with the idea that many of the issues presented above are specific to the *Mathematica* implementation of `Integrate`, we refer to [4] for a discussion that overlaps ours but is not specific to any particular body of code.

My opinion is that future work in definite integration within *Mathematica* should go in a few directions. I list various possibilites below.

Catching specialized integrals that now take substantial time but could be made significantly faster with special-case code. Integrals of rational and other simple functions over parametrized ranges would be an example.

Catching specialized integrals that now often give wrong results, but could be made correct much more often with special-case code. Elliptic integrals might be a good class of integrals in this category.

Strengthening the `MeijerG` convolution code, with attention paid to form of results, range of integrands covered, and correct handling of convergence and branch cut problems.

Better assessment of convergence conditions. This will include splitting the `GenerateConditions` option into two or three distinct ones so that meanings are more clear. At present the code may use it to test for convergence, or to look for parameter conditions that guarantee convergence.

Figuring out a platform independent way to measure work done, and base an interruption scheme thereon. It is a fact of life that certain possibly expensive operations are needed to make an integration code work well, and they can also hang examples if not terminated within reasonable time. Basing this on something as crude as TimeConstrained is problematic, among other reasons because results are not independent of the speed of the machine on which they are run.

Continued debugging based on existing bugs in that code. The overall count of bugs in `Integrate` is now about 30 percent of what it was when this work was first undertaken. This brings the quantity to a point where a careful categorization of specific problem areas becomes realistic. My anticipation is that some of these specific areas will simply need point fixes. What remains as problematic should be amenable, in parts, to rewriting without deleterious implications to the overall structure and function of Integrate code.

Continued addition to the test suite so as to avoid large scale breakage during overhaul. In past a problem that led to the massive bugs list was that we did not have an adequate test suite It had many integration examples, but not enough for a function of the scope of `Integrate`. Hence bug fixes might well cause breakage elsewhere and no "trip wire" existed in terms of a sufficiently large test suite. At present, virtually all fixed bugs get made into new tests.

References

[1] V. S. Adamchik and O. I. Marichev. The algorithm for calculating integrals of hypergeometric type functions and its realization in reduce system. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, ISSAC '90, pages 212–224, New York, NY, USA, 1990. ACM.

[2] M. Bronstein. *Symbolic integration I: transcendental functions*. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[3] K. Charlwood. Some integration trials on computer algebra systems. Manuscript. available from http://staff.bath.ac.uk/masjhd/Calculemus2003-paper.pdf, 2002.

[4] J. Davenport. The difficulties of definite integration. Manuscript. available from http://staff.bath.ac.uk/masjhd/Calculemus2003-paper.pdf, 2003.

[5] K. O. Geddes, S. R. Czapor, and G. Labahn. *Algorithms for Computer Algebra*. Kluwer Academic Publishers, Norwell, MA, USA, 1992.

[6] G. Gershwin and I. Gershwin. It ain't necessarily so [music] / words by Ira Gershwin ; [music by] George Gershwin, 1935. From: "Porgy and Bess".

[7] D. J. Jeffrey. Integration to obtain expressions valid on domains of maximum extent. In *Proceedings of the 1993 International Symposium on Symbolic and Algebraic Computation*, ISSAC '93, pages 34–41, New York, NY, USA, 1993. ACM.

[8] D. J. Jeffrey. Rectifying transformations for the integration of rational trigonometric functions. *Journal of Symbolic Computation*, 24:563–573, November 1997.

[9] D. J. Jeffrey and A. D. Rich. The evaluation of trigonometric integrals avoiding spurious discontinuities. *ACM Transactions on Mathematical Software*, 20:124–135, March 1994.

[10] D. J. Jeffrey and A. D. Rich. Reducing expression size using rule-based integration. In *AISC/MKM/Calculemus*, volume 6167 of *Lecture Notes in Computer Science*, pages 234–246. Springer, 2010.

[11] D. Lichtblau. Symbolic definite integration (mathematica technical conference talk), 2005. slides available from: http://library.wolfram.com/infocenter/Conferences/5832/.

[12] A. D. Rich and D. J. Jeffrey. A knowledge repository for indefinite integration based on transformation rules. In *Calculemus/MKM*, volume 5625 of *Lecture Notes in Computer Science*, pages 480–485. Springer, 2009.

[13] S. Wolfram. *The Mathematica Book*. Wolfram Media, Champaign, IL, USA, 5th edition, 2003.