1-15-2006

TSIL: A Program for the Calculation of Two-Loop Self-Energy Integrals

Steve P. Martin
Fermilab & Northern Illinois University

David G. Robertson
Otterbein University

hep-ph/0501132

TSIL: a program for the calculation of two-loop self-energy integrals

Stephen P. Martin[a,b] and David G. Robertson[c]

(a) *Department of Physics, Northern Illinois University, DeKalb IL 60115*
(b) *Fermi National Accelerator Laboratory, P.O. Box 500, Batavia IL 60510*
(c) *Department of Physics and Astronomy, Otterbein College, Westerville OH 43081*

Abstract

TSIL is a library of utilities for the numerical calculation of dimensionally regularized two-loop self-energy integrals. A convenient basis for these functions is given by the integrals obtained at the end of O.V. Tarasov's recurrence relation algorithm. The program computes the values of all of these basis functions, for arbitrary input masses and external momentum. When analytical expressions in terms of polylogarithms are available, they are used. Otherwise, the evaluation proceeds by a Runge-Kutta integration of the coupled first-order differential equations for the basis integrals, using the external momentum invariant as the independent variable. The starting point of the integration is provided by known analytic expressions at (or near) zero external momentum. The code is written in C, and may be linked from C, C++, or Fortran. A Fortran interface is provided. We describe the structure and usage of the program, and provide a simple example application. We also compute two new cases analytically, and compare all of our notations and conventions for the two-loop self-energy integrals to those used by several other groups.

Contents

1 Program summary

Title of program: `TSIL`

Version number: 1.3

Available at: `http://www.niu.edu/spmartin/TSIL/`
`http://faculty.otterbein.edu/DRobertson/tsil/`

Programming Language: C

Platform: Any platform supporting the GNU Compiler Collection (gcc), the Intel C compiler (icc), or a similar C compiler with support for complex mathematics

Distribution format: ASCII source code

Keywords: quantum field theory, Feynman integrals, two-loop integrals, self-energy corrections, dimensional regularization

Nature of physical problem: Numerical evaluation of dimensionally regulated Feynman integrals needed in two-loop self-energy calculations in relativistic quantum field theory in four dimensions

Method of solution: Analytical evaluation in terms of polylogarithms when possible, otherwise through Runge-Kutta solution of differential equations

Limitations: Loss of accuracy in some unnatural threshold cases that do not have vanishing masses

Typical running time: Less than a second

2 Introduction

The precision of measurements within the Standard Model requires a level of accuracy obtained from two-loop, and even higher-order, calculations in quantum field theory. The future discoveries of the Large Hadron Collider and a future e^+e^- linear collider may well extend these requirements even further. For example, supersymmetry predicts the existence of many new particle states with perturbative interactions. The most important observables in softly broken supersymmetry are just the new particle masses. Therefore, comparisons of particular models of supersymmetry breaking with experiment will require systematic methods for two-loop computations of physical pole masses in terms of the underlying Lagrangian parameters.

In this paper, we describe a software package called `TSIL` (Two-loop Self-energy Integral Library)[†] that can be used to compute two-loop self-energy functions. In a general quantum field theory, the necessary dimensionally regularized Feynman integrals can be expressed in the form:

$$\mu^{8-2d} \int d^d k \int d^d q \frac{(k^2)^{n_1} (q^2)^{n_2} (k\cdot p)^{n_3} (q\cdot p)^{n_4} (k\cdot q)^{n_5}}{[k^2+x]^{r_1}[q^2+y]^{r_2}[(k-p)^2+z]^{r_3}[(q-p)^2+u]^{r_4}[(k-q)^2+v]^{r_5}}, \tag{2.1}$$

for non-negative integers n_i, r_i. (See the master topology **M** in Figure 1.) The integrations are over Euclidean momenta in

$$d = 4 - 2\epsilon \tag{2.2}$$

dimensions. An algorithm has been derived by O.V. Tarasov [1], and implemented in a Mathematica program TARCER by Mertig and Scharf [2], that allows all such integrals to be systematically reduced to linear combinations of a few basis integrals. The coefficients are ratios of polynomials in the external momentum invariant and the propagator squared masses x, y, z, u, v. In the remainder of this section, we will describe our notations and conventions[‡] for the two-loop basis integrals and some related functions, and describe the strategy used by `TSIL` to compute them.

First, we define a loop factor

$$C = (16\pi^2)\frac{\mu^{2\epsilon}}{(2\pi)^d} = (2\pi\mu)^{2\epsilon}/\pi^2. \tag{2.3}$$

The regularization scale μ is related to the renormalization scale Q (in the modified minimal subtraction renormalization scheme based on dimensional regularization [6], or dimensional reduction

[†]In the Hopi culture indigenous to the American southwest, Tsil is the Chili Pepper Kachina, one of many supernatural spirits represented by masked doll-like figurines and impersonated by ceremonial dancers. Tsil is one of the runner Kachinas. When he overtakes you in a race, he may stuff your mouth with hot chili peppers.

[‡]These are the same as in refs. [4, 5]; the correspondences with some other papers is given in Appendix A.

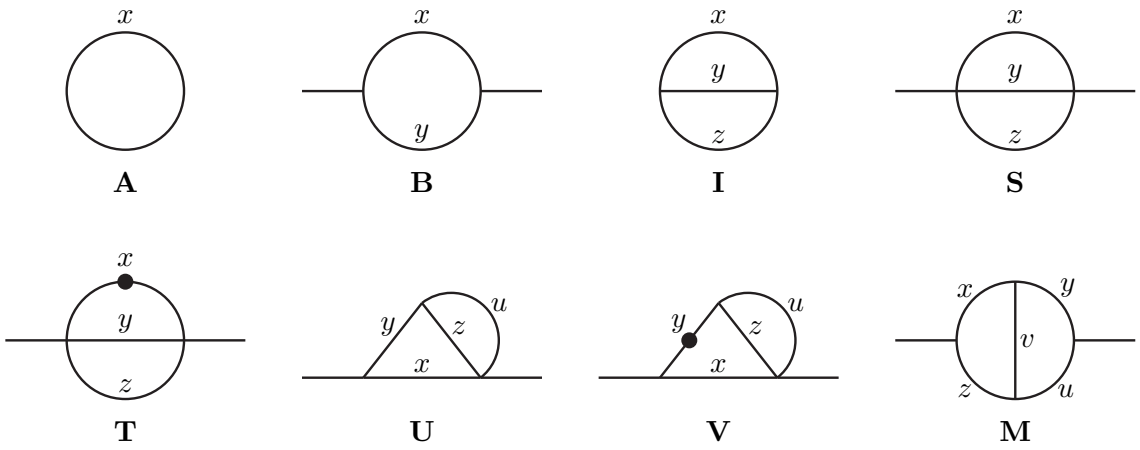

Figure 1: Feynman diagram topologies for the one- and two-loop vacuum and self-energy integrals as defined in this paper.

[7] for softly-broken supersymmetric theories) by

$$Q^2 = 4\pi e^{-\gamma}\mu^2. \tag{2.4}$$

Logarithms of dimensionful quantities are always given in terms of

$$\overline{\ln} X \equiv \ln(X/Q^2). \tag{2.5}$$

The loop integrals are functions of a common external momentum invariant

$$s = -p^2 \tag{2.6}$$

using a Euclidean or signature $(-+++)$ metric. (Note that the sign convention is such that for a stable physical particle with mass m, there is a pole at $s = m^2$.) On the physical sheet, s has an infinitesimal positive imaginary part. Since all functions in any given equation typically have the same s and Q^2, they are not included explicitly in the list of arguments in written formulas. A prime on a squared-mass argument of a function indicates differentiation with respect to that argument. Thus, for example

$$f(x', x, z') = \left[\frac{\partial^2}{\partial y \partial z} f(y, x, z)\right]_{y=x}. \tag{2.7}$$

We now define one-loop vacuum and self-energy integrals as:

$$\mathbf{A}(x) = C\int d^d k \frac{1}{[k^2 + x]} \tag{2.8}$$

$$\mathbf{B}(x, y) = C\int d^d k \frac{1}{[k^2 + x][(k-p)^2 + y]}. \tag{2.9}$$

We also define two-loop integrals according to:

$$\mathbf{S}(x, y, z) = C^2 \int d^d k \int d^d q \frac{1}{[k^2 + x][q^2 + y][(k + q - p)^2 + z]} \tag{2.10}$$

$$\mathbf{I}(x,y,z) = \mathbf{S}(x,y,z)|_{s=0} \tag{2.11}$$

$$\mathbf{T}(x,y,z) = -\mathbf{S}(x',y,z) \tag{2.12}$$

$$\mathbf{U}(x,y,z,u) = C^2 \int d^dk \int d^dq \frac{1}{[k^2+x][(k-p)^2+y][q^2+z][(q+k-p)^2+u]} \tag{2.13}$$

$$\mathbf{V}(x,y,z,u) = -\mathbf{U}(x,y',z,u) \tag{2.14}$$

$$\mathbf{M}(x,y,z,u,v) = C^2 \int d^dk \int d^dq \frac{1}{[k^2+x][q^2+y][(k-p)^2+z][(q-p)^2+u][(k-q)^2+v]}. \tag{2.15}$$

The corresponding Feynman diagram topologies are shown in fig. 1. These integrals have various symmetries that are clear from the diagrams: $\mathbf{B}(x,y)$ is invariant under interchange of x,y; the "sunrise" integral $\mathbf{S}(x,y,z)$ and $\mathbf{I}(x,y,z)$ are invariant under interchange of any two of x,y,z; $\mathbf{T}(x,y,z)$ is invariant under $y \leftrightarrow z$; $\mathbf{U}(x,y,z,u)$ and $\mathbf{V}(x,y,z,u)$ are invariant under $z \leftrightarrow u$; and the "master" integral $\mathbf{M}(x,y,z,u,v)$ is invariant under each of the interchanges $(x,z) \leftrightarrow (y,u)$, and $(x,y) \leftrightarrow (z,u)$, and $(x,y) \leftrightarrow (u,z)$.

It is often convenient to introduce modified integrals in which appropriate divergent parts have been subtracted and the regulator removed. At one-loop order, we define the finite and ϵ-independent integrals:

$$A(x) = \lim_{\epsilon\to 0} [\mathbf{A}(x) + x/\epsilon] = x(\overline{\ln} x - 1) \tag{2.16}$$

$$B(x,y) = \lim_{\epsilon\to 0} [\mathbf{B}(x,y) - 1/\epsilon] = -\int_0^1 dt\, \overline{\ln}[tx + (1-t)y - t(1-t)s]. \tag{2.17}$$

At two loops, we let

$$S(x,y,z) = \lim_{\epsilon\to 0} \left[\mathbf{S}(x,y,z) - S^{(1)}_{\rm div}(x,y,z) - S^{(2)}_{\rm div}(x,y,z)\right], \tag{2.18}$$

where

$$S^{(1)}_{\rm div}(x,y,z) = \left(\mathbf{A}(x) + \mathbf{A}(y) + \mathbf{A}(z)\right)/\epsilon \tag{2.19}$$

$$S^{(2)}_{\rm div}(x,y,z) = (x+y+z)/2\epsilon^2 + (s/2 - x - y - z)/2\epsilon \tag{2.20}$$

are the contributions from one-loop subdivergences and from the remaining two-loop divergences, respectively. In addition,

$$I(x,y,z) = S(x,y,z)|_{s=0} \tag{2.21}$$

$$T(x,y,z) = -S(x',y,z). \tag{2.22}$$

Similarly, we define

$$U(x,y,z,u) = \lim_{\epsilon\to 0} \left[\mathbf{U}(x,y,z,u) - U^{(1)}_{\rm div}(x,y) - U^{(2)}_{\rm div}\right] \tag{2.23}$$

where

$$U^{(1)}_{\rm div}(x,y) = \mathbf{B}(x,y)/\epsilon \tag{2.24}$$

$$U^{(2)}_{\rm div} = -1/2\epsilon^2 + 1/2\epsilon \tag{2.25}$$

are again the contributions from one-loop sub-divergences and the remaining two-loop divergences. Also, we define

$$V(x,y,z,u) \;=\; -U(x,y',z,u). \tag{2.26}$$

The master integral is free of divergences, so we define

$$M(x,y,z,u,v) \;=\; \lim_{\epsilon\to 0}\mathbf{M}(x,y,z,u,v). \tag{2.27}$$

Thus, bold-faced letters $\mathbf{A},\mathbf{B},\mathbf{I},\mathbf{S},\mathbf{T},\mathbf{U},\mathbf{V}$ represent the original regularized integrals that diverge as $\epsilon\to 0$, while the ordinary letters A,B,I,S,T,U,V,M are used to represent functions that are finite and independent of ϵ by definition. Note, however, that as defined above I,S,T,U,V are not simply the ϵ-independent terms in expansions in small ϵ. The following expansions are useful for converting between $\mathbf{I},\mathbf{S},\mathbf{T},\mathbf{U},\mathbf{V}$ and I,S,T,U,V:

$$\mathbf{A}(x) \;=\; -x/\epsilon + A(x) + \epsilon A_\epsilon(x) + \mathcal{O}(\epsilon^2) \tag{2.28}$$

$$\mathbf{B}(x,y) \;=\; 1/\epsilon + B(x,y) + \epsilon B_\epsilon(x,y) + \mathcal{O}(\epsilon^2), \tag{2.29}$$

where

$$A_\epsilon(x) \;=\; x[-1-\zeta(2)/2 + \overline{\ln}x - (\overline{\ln}x)^2/2] \tag{2.30}$$

$$B_\epsilon(x,y) \;=\; \zeta(2)/2 + \frac{1}{2}\int_0^1 dt\,(\overline{\ln}[tx+(1-t)y - t(1-t)s])^2. \tag{2.31}$$

Here ζ is the Riemann zeta function. The function $B_\epsilon(x,y)$ can be expressed in terms of dilogarithms [8], and is given by the coefficient of δ in eq. (83) of ref. [9]. Also,

$$\mathbf{B}(x',y) = \Big[(3-d)(s-x+y)\mathbf{B}(x,y) + (2-d)\{\mathbf{A}(y) + (s-x-y)\mathbf{A}(x)/2x\}\Big]/\Delta_{sxy} \tag{2.32}$$

where

$$\Delta_{abc} \equiv a^2+b^2+c^2-2ab-2ac-2bc. \tag{2.33}$$

From the preceding equations it follows that

$$\begin{aligned}
\mathbf{I}(x,y,z) \;=\;& -(x+y+z)/2\epsilon^2 + [A(x)+A(y)+A(z)-(x+y+z)/2]/\epsilon \\
&+I(x,y,z) + A_\epsilon(x) + A_\epsilon(y) + A_\epsilon(z) + \mathcal{O}(\epsilon)
\end{aligned} \tag{2.34}$$

$$\begin{aligned}
\mathbf{S}(x,y,z) \;=\;& -(x+y+z)/2\epsilon^2 + [A(x)+A(y)+A(z)-(x+y+z)/2+s/4]/\epsilon \\
&+S(x,y,z) + A_\epsilon(x) + A_\epsilon(y) + A_\epsilon(z) + \mathcal{O}(\epsilon)
\end{aligned} \tag{2.35}$$

$$\mathbf{T}(x,y,z) \;=\; 1/2\epsilon^2 - [A(x)/x + 1/2]/\epsilon + T(x,y,z) + [A(x)-A_\epsilon(x)]/x + \mathcal{O}(\epsilon) \tag{2.36}$$

$$\mathbf{U}(x,y,z,u) \;=\; 1/2\epsilon^2 + [1/2 + B(x,y)]/\epsilon + U(x,y,z,u) + B_\epsilon(x,y) + \mathcal{O}(\epsilon) \tag{2.37}$$

$$\begin{aligned}
\mathbf{V}(x,y,z,u) \;=\;& \frac{1}{\epsilon}[(s+x-y)(B(x,y)-1) + 2A(x) + (s-x-y)A(y)/y]/\Delta_{sxy} \\
&+V(x,y,z,u) + \Big\{(s+x-y)[B_\epsilon(x,y) - 2B(x,y)] + 2A_\epsilon(x) - 2A(x) \\
&+(s-x-y)[A_\epsilon(y)-A(y)]/y\Big\}/\Delta_{sxy} + \mathcal{O}(\epsilon)
\end{aligned} \tag{2.38}$$

$$\mathbf{M}(x,y,z,u,v) \;=\; M(x,y,z,u,v) + \mathcal{O}(\epsilon). \tag{2.39}$$

The internal workings of the TSIL code use the functions A, B, I, S, T, U, V, M rather than their bold-faced counterparts. This is because we find that renormalized expressions for physical quantities are more compactly written in terms of the non-bold-faced integrals. However, both types of functions are available as outputs, with the proviso that for $\mathbf{I}, \mathbf{S}, \mathbf{T}, \mathbf{U}, \mathbf{V}, \mathbf{M}$ we keep only the pole and finite terms as $\epsilon \to 0$, and for $\mathbf{A}, \mathbf{B}$ only terms through order ϵ.

Tarasov's algorithm [1] allows any integral of the form of eq. (2.1) to be expressed[§] as a linear combination of the following two-loop basis integrals:

$$\begin{aligned} &M(x,y,z,u,v), \quad U(z,x,y,v), \quad U(u,y,x,v), \quad U(x,z,u,v), \quad U(y,u,z,v), \quad T(v,y,z), \\ &T(u,x,v), \quad T(y,z,v), \quad T(x,u,v), \quad T(z,y,v), \quad T(v,x,u), \quad S(v,y,z), \quad S(u,x,v), \end{aligned} \tag{2.40}$$

plus terms involving the two-loop vacuum integrals $I(x,y,v)$ or $I(z,u,v)$, or quadratic in the one-loop integrals.

In particular, the V and $\mathbf{V}$ integrals can be expressed as linear combinations of the others (see eqs. (3.22)–(3.28) and (6.21) of ref. [4]), and so are not included in the basis. However, they are included as outputs, because some results are more compactly written in terms of them. Because $T(x,y,z)$ is divergent in the limit $x \to 0$, it is also sometimes useful to define the function:

$$\overline{T}(x,y,z) = T(x,y,z) + B(y,z)\,\overline{\ln} x. \tag{2.41}$$

For $x = 0$, this function is well-defined and can be written in terms of dilogarithms (see eqs. (6.18)–(6.19) of ref. [4]). In that limit it can also be rewritten in terms of the other basis functions, see eqs. (A.15)–(A.16) of ref. [5], but is still available as an output of TSIL for convenience.

It remains to provide a means for the numerical computation of the basis integrals. For special values of masses and external momentum, it is possible to compute the two-loop integrals analytically in terms of polylogarithms [10]. This requires [11] that there is no three-particle cut of the diagram for which the cut propagator masses m_1, m_2, m_3 and the five quantities

$$s_{\rm cut}, \quad s_{\rm cut} - (m_1 \pm m_2 \pm m_3)^2, \tag{2.42}$$

(where $s_{\rm cut} = -p_{\rm cut}^2$ is the momentum invariant for the total momentum flowing across the cut) are all non-zero. Many analytical results for various such special cases have been worked out in refs. [12]–[20], [9], [4], and Appendix B of the present paper. There are also expansions [21]–[25] in large and small values of the external momentum invariant, and near the thresholds and pseudo-thresholds [26]–[35]. Analytical results in terms of polylogarithms for the A, B, I, S, T, U, M functions at generic values of s are reviewed in section VI of ref. [4]. These and some other analytical formulas for special cases have been implemented in the TSIL code. They consist of the master

[§] Actually, Tarasov's algorithm relates the general integral to the bold-faced versions of the basis functions, and holds for general d. By neglecting contributions that vanish for $\epsilon \to 0$, one can write the results in terms of the non-bold-faced functions.

integral cases:

$M(x,x,y,y,0)$,	$M(0,0,0,x,0)$,	$M(0,x,0,x,x)$,	ref. [16]
$M(0,0,0,0,x)$,	$M(0,x,0,y,0)$,		ref. [9]
$M(0,0,0,x,x)$,	$M(0,x,x,0,0)$,	$M(0,x,x,x,0)$,	ref. [19]
$M(0,0,x,y,0)$,			Appendix B
$M(0,x,x,0,x)\|_{s=x}$,			ref. [16]
$M(0,y,y,0,x)\|_{s=x}$,			Appendix B
$M(0,x,y,0,y)\|_{s=x}$,			refs. [36, 37] (see also Appendix B)

and functions obtained by permutations of the arguments, and all subordinate integrals A, B, I, S, T, U, V, $\overline{T}$ that have topologies obtained by removing one or more propagator lines from the cases above. These include:

$U(x,y,0,y)$,	ref. [15]
$U(x,y,0,0)$, $U(0,0,0,x)$,	ref. [9]
$U(0,x,y,z)$,	ref. [4]
$U(x,0,0,y)$,	ref. [5]
$U(x,0,y,y)\|_{s=x}$, $U(y,0,y,x)\|_{s=x}$,	refs. [38, 39] (see also Appendix B)

and

$S(0,x,y)$, $T(x,0,y)$, $\overline{T}(0,x,y)$,	ref. [18]
$S(x,y,y)\|_{s=x}$, $T(x,y,y)\|_{s=x}$, $T(y,x,y)\|_{s=x}$.	refs. [38, 29]

Also included are all of the functions at $s = 0$, which can be easily expressed in terms of the A and I functions and derivatives of them, which are in turn known [14, 40] analytically in terms of logarithms and dilogarithms.

For the case of generic masses and s, another method is needed. Integral representations have been studied in refs. [41]–[50]. For `TSIL`, we instead use the differential equations method [51]–[52] to evaluate the integrals numerically. For the **S**, **T** and **U** integrals, this strategy was proposed and implemented in [53]–[57]. The method was rewritten in terms of the S, T, U integrals and extended to M in ref. [4]. To see how the method works, consider the functions listed in eq. (2.40) and also $B(x,z)$ and $B(y,u)$ and the product $B(x,z)B(y,u)$. Let us denote these sixteen quantities by F_i where $i = 1, \ldots, 16$. Considered as functions of s for fixed x, y, z, u, v, Q^2 they can be shown to satisfy a set of coupled first-order differential equations of the form

$$\frac{d}{ds}F_i = \sum_j C_{ij}F_j + C_i, \tag{2.43}$$

where the coefficients C_{ij} and C_i are ratios of polynomials in the squared masses and s (and of the analytically known A and I functions, for some of the coefficients not multiplying two-loop

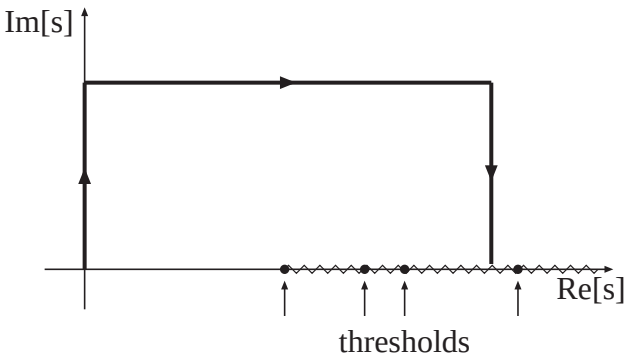

Figure 2: When s is greater than or equal to the smallest non-zero threshold or pseudo-threshold, the Runge-Kutta integration proceeds along a contour with the shape shown here, as suggested in ref. [56].

functions). The coefficients can be evaluated by using Tarasov's recurrence relations, and were listed in [4].

At $s = 0$ the values of all of the functions and their derivatives with respect to s (and/or their expansions in small s) are known analytically in terms of dilogarithms. Therefore, one can integrate the functions[¶] by the Runge-Kutta method to any other value of s. In order to avoid numerical problems from integrating through thresholds and pseudo-thresholds, we use the suggestion of ref. [56] by following a displaced contour in the upper-half complex s plane, as shown in fig. 2, whenever s is greater than the smallest non-zero threshold or pseudo-threshold. This contour starts from $s = 0$ (or, in some cases, a point very close by, as explained in the next section) and proceeds to a point $is_{\rm im}$, from there to $s + is_{\rm im}$, and from there to the desired value s. Here $s_{\rm im}$ is real and positive. Since s has an infinitesimal positive imaginary part on the physical sheet, this procedure also automatically produces the correct branch cut behavior for functions when s is above thresholds. When s is less than or equal to the smallest non-zero threshold or pseudo-threshold, the Runge-Kutta integration instead proceeds directly along the real s axis. In typical physical problems, if the master integral is needed, then so will be all of its subordinate B, S, T, U integrals. These are all obtained simultaneously as a result of the Runge-Kutta method. Furthermore, checks on the numerical accuracy can be made by varying the Runge-Kutta step size parameters and the choice of contour in the upper-half complex s plane.

Most of the practical difficulties in realizing this program have to do with numerical instabilities when the final value of s is at or near a threshold or pseudo-threshold, or when the starting point $s = 0$ is itself a threshold. We describe these issues and the methods used by `TSIL` to successfully evade them in the next section.

[¶]For the master integral, we actually run $sM(x, y, z, u, v)$.

3 Numerical considerations near thresholds and pseudo-thresholds

The two-loop master integral function $M(x,y,z,u,v)$, and its subordinates listed in eq. (2.40), have two-particle and three-particle thresholds at s equal to:

$$t_{xz} = (\sqrt{x}+\sqrt{z})^2, \qquad t_{yu} = (\sqrt{y}+\sqrt{u})^2, \tag{3.1}$$

$$t_{xuv} = (\sqrt{x}+\sqrt{u}+\sqrt{v})^2, \qquad t_{yzv} = (\sqrt{y}+\sqrt{z}+\sqrt{v})^2. \tag{3.2}$$

At the thresholds, the integral functions have branch cuts, and they therefore develop imaginary part contributions for $s > s_{\rm thresh}$. The pseudo-thresholds occur at s equal to:

$$p_{xz} = (\sqrt{x}-\sqrt{z})^2, \qquad p_{yu} = (\sqrt{y}-\sqrt{u})^2, \tag{3.3}$$

$$p_{xuv} = (-\sqrt{x}+\sqrt{u}+\sqrt{v})^2, \qquad p_{uxv} = (\sqrt{x}-\sqrt{u}+\sqrt{v})^2, \tag{3.4}$$

$$p_{vxu} = (\sqrt{x}+\sqrt{u}-\sqrt{v})^2, \qquad p_{yzv} = (-\sqrt{y}+\sqrt{z}+\sqrt{v})^2, \tag{3.5}$$

$$p_{zyv} = (\sqrt{y}-\sqrt{z}+\sqrt{v})^2, \qquad p_{vyz} = (\sqrt{y}+\sqrt{z}-\sqrt{v})^2. \tag{3.6}$$

The integral functions are analytic at the pseudo-threshold points (unless they coincide with a threshold), but the coefficients in the differential equation have pole singularities at both the thresholds and pseudo-thresholds. For values of s close to both types of special points, one must therefore be careful in numerical evaluation to avoid undefined quantities and round-off errors. In this section, we discuss the ways in which `TSIL` avoids these problems.

First, we consider the case that the initial point of the Runge-Kutta integration, $s = 0$, is actually a threshold. This occurs for master integrals $M(0,y,z,0,0)$ and $M(0,y,0,u,v)$, and permutations thereof. In these cases, some of the basis integral functions have logarithmic singularities and/or branch cuts at $s = 0$. To deal with this, we make a change of independent variable to

$$r = \overline{\ln}(-s), \tag{3.7}$$

and start the integration at a point $r_0 = \ln(-i\delta) = \ln(\delta) - i\pi/2$, with δ real, positive, and extremely small (of order the relative error of the computer arithmetic). The initial values of the integrals are obtained from the expansions in small s, given in section V of ref. [4]. This change of variables is also used when $s = 0$ is close to, but not exactly a threshold (with the exact criteria adjustable by the user). This variable r is used for the first leg of the contour in the Runge-Kutta integration.

Next, consider the case that the final value of s is at, or very near, a threshold $s_{\rm thresh}$. In this case, we find that the change of variable

$$r = \ln(1 - s/s_{\rm thresh}) \tag{3.8}$$

is effective, and so is used by `TSIL` for the final part of the Runge-Kutta integration. When the final value of s is exactly equal to a threshold, then the endpoint of the running is taken to be $r = \ln(\delta) - i\pi/2$, with δ again taken to be real, positive, and extremely small.

We next describe the Runge-Kutta algorithms used, since they have some slightly unusual properties dictated by the need for control of precision near thresholds and pseudo-thresholds.

Consider a vector of quantities $\vec{F}$ that satisfy coupled first-order differential equations $d\vec{F}/dt = \vec{f}(t, \vec{F})$. The general form for an explicit m-stage Runge-Kutta routine advancing the solution from t to $t + h$ is given by:

$$\vec{F}(t+h) = \vec{F}(t) + h \sum_{i=1}^{m} b_i \vec{k}_i, \tag{3.9}$$

where

$$\vec{k}_i = \vec{f}(t + c_i h, \vec{F}(t) + h \sum_{j=1}^{i-1} a_{ij} \vec{k}_j) \qquad (\text{for } i = 1, \ldots, m). \tag{3.10}$$

Here a_{ij}, b_i, and c_i are known as Butcher coefficients, and satisfy $c_1 = 0$, $a_{10} = 0$, and

$$c_i = \sum_{j=1}^{i-1} a_{ij} \qquad (\text{for } i = 2, \ldots, m) \tag{3.11}$$

$$\sum_{i=1}^{m} b_i = 1 \tag{3.12}$$

plus other, non-linear, constraints [58]. The algorithm is said to be of nth order if the error is proportional to h^{n+1} for sufficiently small h. In order to implement automatic step-size adjustment, we use a 6-stage embedded Runge-Kutta [59] with coefficients given by Cash and Karp [60]. These give not only a 5th-order step as in eq. (3.9), but a 4th-order step estimate of the same form with different coefficients b_i^*. This gives an error estimate for each dependent variable, for each step:

$$\Delta\vec{F}(t+h) = h \sum_{i=1}^{m} (b_i - b_i^*) \vec{k}_i. \tag{3.13}$$

The theoretical estimate of the step size needed so that the error for each dependent variable is less than δ_P times the increment of that variable is then given by:

$$h_{\text{new}} = hS \left[\frac{\delta_P}{\text{Max}(|\Delta\vec{F}|)} \right]^{1/4}, \tag{3.14}$$

where S is a safety factor less than unity.

However, in the present application there is a special problem because the final destination point might be equal to (or close to) a threshold or pseudo-threshold. There, the individual coefficients in the derivatives of the functions might be ill-defined (or subject to large numerical round-off errors, because of small denominators in individual terms), even though the basis functions themselves are well-defined. To avoid this problem, we instead need to use an m-stage Runge-Kutta with the crucial property $c_i < 1$ for all i, so that no derivatives are ever evaluated at the endpoint. There are no 4-stage, 4th-order solutions to the Butcher coefficient conditions with this property, but there are many 5-stage, 4th-order solutions. We chose, somewhat arbitrarily, the set:

$$c_i = (0,\ 1/4,\ 3/8,\ 1/2,\ 5/8) \tag{3.15}$$

$$b_i = (-1/15,\ 2/3,\ 4/3,\ -10/3,\ 12/5) \tag{3.16}$$

$$a_{21} = 1/4,\ a_{32} = 3/8,\ a_{42} = 1/2,\ a_{52} = 35/72,\ a_{54} = 5/36, \tag{3.17}$$

$$a_{ij} = 0 \qquad \text{for other } i, j. \tag{3.18}$$

This procedure is not as efficient as the Cash-Karp 6-stage, 5th-order algorithm under normal circumstances, and does not provide an error estimate, so it is used only where needed for the very last Runge-Kutta step.

The `TSIL` implementation of the coefficients C_{ij} and C_i in eqs. (2.43) is also done in a special way to avoid roundoff errors near thresholds and pseudo-thresholds. The expressions as given in [4] for these coefficients appear to have double (or higher) poles in s for some special values of the masses with degenerate thresholds and pseudo-thresholds. The presence of such higher-order poles can lead to large round-off errors, due to incomplete cancellations in computer arithmetic with finite precision. Fortunately, this is avoided in most cases by applying the partial fractions technique to rewrite the coefficients in the derivatives with respect to s, so that all poles in s are at most simple poles. This can always be done for the B, S, and T functions.

For the U functions, double poles in the coefficient functions for the derivative with respect to s remain only if the second argument vanishes. Here, we take advantage of the facts that $U(x, 0, y, z)$ does not enter into the differential equations that govern the other basis functions, and it can always be written algebraically in terms of them (see eqs. (A.15), (A.17) and (A.18) of ref. [5]). Therefore, when the second argument of a U function vanishes, the result obtained for it from the Runge-Kutta running is irrelevant; it is simply replaced by the algebraic result before it is returned by the program, eliminating the roundoff error problem.

In most cases for which double poles in the coefficient functions of the derivative of the master integral cannot be eliminated, it can be evaluated in terms of polylogarithms, so the Runge-Kutta technique is not needed anyway. A special case in which this does not occur is $M(x, y, z, u, v)$ for $v = (\sqrt{x} + \sqrt{y})^2$; then $d(sM)/ds$ in ref. [4] has coefficients with double poles at $s_0 = [\sqrt{x}(u - y) + \sqrt{y}(z - x)]/\sqrt{v}$. However, here we can use the identity:

$$\begin{aligned} 0 \;=\; & \sqrt{x}[U(z, x, y, v) - T(x, u, v)] + \sqrt{y}[U(u, y, x, v) - T(y, z, v)] \\ & + \sqrt{v}[T(v, u, x) + T(v, y, z) - 1] + [A(v)/\sqrt{v} - A(x)/\sqrt{x} - \sqrt{y}]B(y, u) \\ & + [A(v)/\sqrt{v} - A(y)/\sqrt{y} - \sqrt{x}]B(x, z), \end{aligned} \tag{3.19}$$

valid in general for $v = (\sqrt{x} + \sqrt{y})^2$. When the right side of equation (3.19) is multiplied by $[\sqrt{z}(u - y) + \sqrt{u}(z - x)]/v(s - s_0)^2$ and added to the expression for $d(sM)/ds$ from ref. [4], the result generically has no double poles in s. That is the form used by the program in this special case (and others related by permutations of the masses).

A non-generic sub-case of the preceding, for which double poles in the coefficient functions of $d(sM)/ds$ are not eliminated, is $M(0, y, z, u, y)$ with $y = (\sqrt{z} \pm \sqrt{u})^2$. Because the coefficients involve double poles at $s = z$, there is some loss of accuracy at, and very near, that threshold. Fortunately, there is no good reason why a relation like $y = (\sqrt{z} \pm \sqrt{u})^2$ should hold exactly in a quantum field theory, unless a symmetry makes y or z or u equal to 0, and in each of these cases the master integral and all of its subordinates are given in terms of polylogarithms. More generally, we have checked that the coefficients in the derivatives as implemented in `TSIL` always have only simple poles, except in "unnatural" cases of this type (where no symmetry of a quantum

field theory can cause the necessary coincidence), or when the integrals are already analytically computed.

The measures detailed above are generally sufficient to give good numerical accuracy near thresholds and pseudo-thresholds (except in the unnatural coincidence case just mentioned) without need for interpolation techniques or special expansions.

4 Description of the program

`TSIL` is a library of functions written in C, which can be (statically) linked from C, C++, or Fortran code.[†] In addition to the main evaluation functions, it contains a variety of routines for I/O and other utilities. A complete list of functions in the user API is given in Appendix C.

The principal data object in the code is a C `struct` with type name `TSIL_DATA` that contains values of the parameters x, y, z, u, v and Q^2 as well as the 15 basis functions of types B, S, T, U, M. Each integral function is itself a `struct` containing its value, arguments, and various unchanging coefficients used in computing its derivative. These coefficients are functions of x, y, z, u, v and are computed when the parameter values are set. In addition, each basis function contains a set of pointers to the other functions needed in evaluating its derivative. Also contained in the data struct are values of the integrals $\overline{T}$, V, and $\mathbf{S}$, $\mathbf{T}$, $\mathbf{U}$, and $\mathbf{V}$. Definitions of all datatypes are contained in the header file `tsil.h`, which must be included in all application programs.

In any program that calls `TSIL` functions requiring Runge-Kutta evaluation, at least one of these high-level data objects must be declared:

```
TSIL_DATA foo;
```

(More than one such object, and arrays of such objects, are allowed. See subsection 5.3 for an example.) Users can of course access the items in the `struct` directly, though it is recommended that the provided user interface routines be used. These allow one to extract values of individual functions (or all of them), set the values of the external parameters, and so on.

The size of the basic datatypes used for floating point values is controlled by the user (at compile time) with the choice of compiler flag `-DTSIL_SIZE_LONG` or `-DTSIL_SIZE_DOUBLE`. Then the type `TSIL_REAL` is accordingly synonymous with `long double` or `double`, while the type `TSIL_COMPLEX` is synonymous with `long double complex` or `double complex`. The recommended default size is `long double` on systems where it is available. For most physics applications (taking into account the natural suppression of two-loop effects), `double` should easily give sufficient accuracy. However, the use of `long double` provides a nice safety margin, and execution times are typically short (of order tenths or hundredths of a second on a modern workstation for generic inputs) in any case. Generally, `long double` data (typically with 63 or more bits of relative precision) gives results with relative accuracies better than 10^{-10} for generic cases, but sometimes somewhat worse in cases with large mass hierarchies, and in some particularly difficult cases significantly worse. [The function

[†] A wrapper routine is included that provides the interface to Fortran; see section 5.5 below.

$V(x, y, z, u)$ for very small but non-zero y can be particularly sensitive to roundoff errors, since the individual terms in its evaluation are proportional to $1/y$ and yet it is only logarithmically divergent as $y \to 0$.] The user should consider modifying the default parameters of the program if significant sensitivity to parameters is expected (or observed), or if speed is an overriding concern.

In a typical application the user will initialize the data object and set values for the external parameters x, y, z, u, v, Q^2 by calling the function `TSIL_SetParameters`. (Parameter values may be changed in a data object with subsequent calls to this function.) Then the basis integrals are evaluated at any desired value of s by calling the master evaluation function `TSIL_Evaluate`. Additional calls to `TSIL_Evaluate` can be used to re-compute the basis functions for other values of s.

`TSIL_Evaluate` first decides whether the case at hand is known analytically; if so, the basis functions are computed directly. If not, numerical integration is required. In this case `TSIL_Evaluate` begins by rescaling all dimensionful quantities by the largest of $x, y, z, u, v, |s|$, so that all parameters are rendered dimensionless. It then locates all thresholds and pseudo-thresholds and decides whether special handling is needed, that is, if one of these special points is near $s = 0$ or the final value of s. The nearness criterion is controlled by a constant `THRESH_CUTOFF`, defined in `tsil_params.h`. If $s = 0$ is a threshold (or there is a threshold very near $s = 0$), evaluation proceeds as described above by making the change of integration variable (3.7). If the final value of s is at or near a threshold, the change of variable (3.8) is enabled for the final leg of the integration contour.

Next, `TSIL_Evaluate` checks to see if the final value of s is smaller than the smallest non-zero threshold or pseudo-threshold; if so, then integration proceeds directly along the real s axis. If not, the generic displaced integration contour (fig. 2) is used. In cases where the final destination s is near a threshold or pseudo-threshold, the 5-stage Runge-Kutta routine described by eqs. (3.15)–(3.18) is used for the very last step of the integration, to avoid evaluation of any derivatives at the endpoint.

After the Runge-Kutta integration, the program checks to see if this was a case with uncanceled double pole terms in the derivatives of the U functions, due to the second argument vanishing. If so, these values are corrected. (Recall that in such cases the incorrect values obtained by the Runge-Kutta integration have no effect on other basis functions.) The program also replaces any subordinate integrals (B, S, T, U) that can be computed in terms of polylogarithms by their analytical values. Finally, the program computes the values of all $\overline{T}$ and V functions, and the "bold" variants of all functions defined in section 2, using eqs. (2.35)–(2.39).

The function `TSIL_Evaluate` returns 1 (`TRUE`) for successful execution or 0 (`FALSE`) for error execution. A warning message is printed if the external parameters correspond to the unnatural threshold case discussed at the end of section 3. The data object further contains a status parameter, accessible via the function `TSIL_GetStatus`, which indicates how the master integral evaluation was performed: either analytic, numerical integration along real axis, or numerical integration along the contour of fig. 2.

The standard output function is `TSIL_PrintData`, which prints all function values on `stdout`. An alternate format, designed so that captured output can serve as valid input files for Mathematica, is given by `TSIL_PrintDataM`. Additional utilities allow the user to extract individual basis functions or sets of functions to arrays. Note that warning and error messages appear on `stderr` so they may be redirected by the shell and examined separately.

Along with the size of intrinsic datatypes, the parameters associated with the numerical integration adaptive step-size control exert the main influences on execution speed and accuracy.[‡] They are realized as members of the `TSIL_DATA` struct, with names:

- `precisionGoal`: This is δ_P in eq. (3.14). (We use a safety factor $S = 0.9$.) If the maximum estimated error for any dependent variable exceeds δ_P multiplied by the increment of that variable for that step, and also exceeds the relative precision of the computer arithmetic times the absolute value of that variable, then the step is retried with a smaller step size, unless the step size would become smaller than specified below. Also, after a successful step, the size for the next step is chosen according to eq. (3.14), unless it would exceed the amount specified below. (Defaults: 10^{-12} for `long double`, 5×10^{-11} for `double`.)

- `nStepsStart`: For each leg of the contour, the initial step size is chosen so that there would be this many steps if the step size did not change. (Default: 500)

- `nStepsMin`: The maximum allowed step size on a leg of the contour with dimensionless (rescaled) independent variable length `L` is given by `L/nStepsMin`. (Default: 100)

- `nStepsMaxCon`, `nStepsMaxVar`: The minimum allowed step size on a leg of the contour with dimensionless independent variable length `L` is given by `L/(nStepsMaxCon + L*nStepsMaxVar)`. (Defaults: 10000, 10000)

The step size is not allowed to increase by more than a factor of 1.5 or decrease by more than a factor of 2 after each step or attempted step. Note that by setting `precisionGoal` to 0, one can arrange that the total number of steps on each leg tends to `nStepsMaxCon + L*nStepsMaxVar`. If instead one sets `precisionGoal` to a very large number, the number of steps will tend to `nStepsMin`. The default values have been found to give good results for a wide variety of different choices of input parameters, for the integration variables used in the program.

In addition to the evaluation for generic parameters described above, `TSIL` provides functions for direct analytical evaluation of the vacuum integrals $A(x)$ and $I(x, y, z)$, the one-loop integral $B(x, y)$, as well as $A(x')$, $B(x', y)$, $\partial B(x, y)/\partial s$, $A_\epsilon(x)$, $B_\epsilon(x, y)$, $I(x', y, z)$, $I(x'', y, z)$, $I(x', y', z)$, $I(x''', y, z)$, and all S, T, $\overline{T}$, U, V, M functions for which results in terms of polylogarithms are available in the literature.

[‡]These are always set, by `TSIL_SetParameters`, to be equal to the values specified at compile time in the file `tsil_params.h`. However, to deal with exceptional situations, they can optionally be reset at run time with the function `TSIL_ResetStepSizeParams`, after calling `TSIL_SetParameters` and before calling `TSIL_Evaluate`.

5 How to use the program

5.1 Building TSIL

Complete instructions for building the library are provided with the distribution. Typically the user edits the `Makefile` to choose the desired data size and set appropriate optimization flags for the compiler. The command `make` then produces the static archive `libtsil.a`, which may be placed in any convenient directory `<dir>`. The user then typically links to this library by passing the flags

```
-L<dir> -ltsil -lm -DTSIL_SIZE_<size>
```

to the linker, where the same flag `-DTSIL_SIZE_<size>` was used in the `Makefile` when compiling `libtsil.a`. The header file `tsil.h` must be included in any source file that makes use of the `TSIL` routines or data structures.

The command `make` also produces an executable `tsil`, which takes as command-line arguments x, y, z, u, v, s, Q^2 and prints out the values of all integral functions together with timing and other information. Also included with the package is a test program `testprog.c` (with executable `tsil.tst` produced by `make`) and a set of 320 files containing comprehensive test data. These include cases representing all known analytic results as well as cases requiring integration that have thresholds and pseudo-thresholds at $s = 0$ and at the final s. Users should run this test suite after building the library to insure that accurate results are obtained. The test program uses pass/fail/warn criteria that are controlled by macros `TSIL_PASS` and `TSIL_WARN` in `tsil_testparams.h`. The first sets the maximum relative error allowed for the test to pass; the second sets a lower error threshold below which the test is deemed to fail. A relative error between these two values results in a warning.

5.2 Essential functionality

In the simplest application of `TSIL`, the parameters $x, y, z, u, v \geq 0$ and $Q^2 > 0$ will be set using `TSIL_SetParameters`, the integrals for real s evaluated using `TSIL_Evaluate`, and the results extracted by the calling program with the command `TSIL_GetFunction`. The code for this might look like:

```
TSIL_SetParameters (&foo, x, y, z, u, v, qq);
TSIL_Evaluate (&foo, s);
integral1 = TSIL_GetFunction (&foo, <string1>);
integral2 = TSIL_GetFunction (&foo, <string2>);
...
```

where `foo` has type `TSIL_DATA`, and `x,y,z,u,v,s,qq` all have type `TSIL_REAL`, and `integral1`, `integral2`, ... have type `TSIL_COMPLEX`, and `<string1>`, `<string2>`, ... can each be one of

```
"M",   "Uzxyv",   "Uuyxv",   "Uxzuv",   "Uyuzv",   "Tvyz",
"Tuxv",   "Tyzv",   "Txuv",   "Tzyv",   "Tvxu",   "Svyz",   "Suxv",
```

according to which of the integrals in eq. (2.40) is desired. In addition, permutations of the above, matching the symmetries of the basis functions, are permitted. Thus, for example, one can access $U(z, x, y, v)$ by specifying either `"Uzxyv"` or `"Uzxvy"` in a call to `TSIL_GetFunction`, since the U functions are symmetric under interchange of their last two arguments. Likewise, any of `"Suxv"`, `"Sxuv"`, `"Suvx"`, `"Svux"`, `"Sxvu"`, or `"Svxu"` will return the function $S(u, x, v)$ (symmetric under interchange of any of its arguments), etc.

Identifier strings can also be one of

```
"Vzxyv",   "Vuyxv",   "Vxzuv",   "Vyuzv",   "Bxz",   "Byu",
```

(and allowed permutations thereof) to access the functions V and the one-loop B functions. Examples are given in subsection 5.3. The functions **S**, **T**, **U** and **V** can be accessed in a similar way, e.g.:

```
integral3 = TSIL_GetBoldFunction (&foo, <string3>, n);
```

would return the coefficient of $1/\epsilon^n$ (for $n = 0, 1,$ or 2) in the bold-faced function corresponding to an appropriate `<string3>` from the list above.

All integrals that are analytically known in terms of polylogarithms can also be evaluated directly, without `TSIL_SetParameters` or `TSIL_Evaluate` or `TSIL_GetFunction`. For example,

```
TSIL_Manalytic (x,y,z,u,v,s,&result);
```

will return the `int` value 1 and set the variable `result` equal to $M(x, y, z, u, v)$ for the appropriate s if it is analytically available, and otherwise will return 0. Here `x,y,z,u,v` are of type `TSIL_REAL`, and `s, result` must be of type `TSIL_COMPLEX`. The functions `TSIL_Sanalytic`, `TSIL_Tanalytic`, `TSIL_Tbaranalytic`, `TSIL_Uanalytic`, and `TSIL_Vanalytic` have analogous behavior, except that they carry an additional argument `qq` of type `TSIL_REAL` for the renormalization scale squared Q^2. For example,

```
TSIL_Uanalytic (x,y,z,u,s,qq,&result)
```

will return the `int` value 1 and set the variable `result` equal to $U(x, y, z, u)$ for the appropriate s and Q^2, if it is analytically available, and otherwise will return 0. The other analytic functions assign without pointers, for example

```
result = TSIL_Bp (x,y,s,qq);
```

will set `result` equal to $B(x', y)$ computed analytically for the appropriate s and Q^2. Some examples are given in the next subsection, and a complete list of the TSIL user API is given in Appendix 6 and the program documentation and header files.

In some applications, it could be that rather than a master integral M and all of its subordinates, one may only need integral functions corresponding to S, T, U or only S, T. Those cases can be evaluated simply by choosing any convenient numbers for the irrelevant squared-mass arguments.

However, this is clearly not optimally efficient. In version 1.1 of TSIL, we have added the capability to only compute the integrals in the S,T,U system, or only those in the S,T system. This has been accomplished by adding functions `TSIL_SetParametersSTU` and `TSIL_SetParametersST`, each of which sets appropriate subsets of the squared mass parameters. A subsequent call of the function `TSIL_Evaluate` will only evaluate the relevant subset of integral functions. So, the user code might include for example:

```
TSIL_SetParametersSTU (&foo, x, z, u, v, qq);
TSIL_Evaluate (&foo, s);
integral1 = TSIL_GetFunction (&foo, <string1>);
integral2 = TSIL_GetFunction (&foo, <string2>);
...
```

where `<string1>`, `<string2>`, ... can each be one of

`"Uxzuv", "Tuxv", "Txuv", "Tvxu", "Suxv", "Bxz".`

Or, if $U(x,z,u,v)$ is not needed, then one can use `TSIL_SetParametersST (&foo, x, u, v, qq);` instead. For generic cases, the S,T,U evaluation is a factor of 4 or 5 faster than full evaluation, and the S,T case gains a further 20% in evaluation speed. Note that the choice of which integral functions are evaluated by `TSIL_Evaluate` is controlled by the most recent call of `TSIL_SetParameters` or `TSIL_SetParametersSTU` or `TSIL_SetParametersST` for the relevant data struct. Also, if S,T,U or S,T evaluation is used, then `TSIL_GetData` and `TSIL_GetBoldData` will generate an error message; only `TSIL_GetFunction` and `TSIL_GetBoldFunction` should be used to extract the results in those cases. Note that in subset evaluation cases where there is only a single function of a given type (U, V, S, or B), the specification string may be abbreviated to only the first character, e.g. "`U`" in place of "`Uxzuv`" above.

In version 1.2 of TSIL, we have introduced a new struct type `TSIL_RESULT`, which contains only the subset of the information found in `TSIL_DATA` that is essential in typical applications. This information consists of: the squared masses x,y,z,u,v, the external momentum invariant s, the renormalization scale Q^2 and the integral results listed in eq. (2.40) as well as $\overline{T}(v,y,z)$, $\overline{T}(u,x,v)$, $\overline{T}(y,z,v)$, $\overline{T}(z,y,v)$, $B(x,z)$ and $B(y,u)$. The `TSIL_RESULT` struct is useful for more efficient storage of results and for performing permutations of the squared masses. The new function `TSIL_CopyResult` can be used to copy data from a full `TSIL_DATA` struct to a `TSIL_RESULT` struct. (This should be done only after `TSIL_Evaluate` has been called on the `TSIL_DATA` struct.) The new function `TSIL_PermuteResult` can then be used to copy the data from one `TSIL_RESULT` struct to another, with the option of permuting the squared masses according to either $(x,z) \leftrightarrow (y,u)$, or $(x,y) \leftrightarrow (z,u)$, or $(x,y) \leftrightarrow (u,z)$, by using the symmetries of the master topology rather than needlessly redoing the integrals. Such permutations occur often in practical applications such as the supersymmetric Standard Model. The elements of the `TSIL_DATA` and `TSIL_RESULT` struct types can be found in the file `tsil.h`.

5.3 Sample applications

As a sample application of `TSIL`, let us calculate the two-loop self energy and pole squared mass for a single scalar field with interaction Lagrangian

$$\mathcal{L} = -\frac{1}{2}m^2\phi^2 - \frac{g}{3!}\phi^3 - \frac{\lambda}{4!}\phi^4 \; . \tag{5.1}$$

Here m^2, g and λ are the tree-level renormalized parameters. The self-energy

$$\Pi(s) = \frac{1}{16\pi^2}\Pi^{(1)}(s) + \frac{1}{(16\pi^2)^2}\Pi^{(2)}(s) + \cdots \tag{5.2}$$

is a function of $s = -p^2$, with p^μ the external momentum, as well as the parameters m^2, g and λ. Note that the metric is either of signature $(-+++)$ or Euclidean. Furthermore, s must be taken to be real with an infinitesimal positive imaginary part to properly resolve the branch cuts.

The pole squared mass

$$s = M^2 - i\Gamma M \tag{5.3}$$

is defined as the position of the pole, with non-positive imaginary part, in the propagator obtained from the perturbative Taylor expansion of the self-energy about the tree-level squared mass.[†] (Note that in the present case the width $\Gamma = 0$ identically.) This leads to the following iterative scheme for computing the two-loop pole squared mass. First, the one-loop approximation $s^{(1)}$ to the pole squared mass is obtained as

$$s^{(1)} = m^2 + \frac{1}{16\pi^2}\Pi^{(1)}(m^2) \; . \tag{5.4}$$

Then, defining

$$\tilde{\Pi} = \frac{1}{16\pi^2}\left[\Pi^{(1)}(m^2) + (s^{(1)} - m^2)\Pi^{(1)\prime}(m^2)\right] + \frac{1}{(16\pi^2)^2}\Pi^{(2)}(m^2) \; , \tag{5.5}$$

in which the prime indicates a derivative with respect to s, we obtain the two-loop approximation to the pole squared mass as

$$s^{(2)} = m^2 + \tilde{\Pi} \; . \tag{5.6}$$

For the scalar theory described by eq. (5.1) we find, including the $\overline{\rm MS}$ counterterms:

$$\Pi^{(1)}(s) = \frac{1}{2}\lambda A(x) - \frac{1}{2}g^2 B(x,x) \tag{5.7}$$

$$\begin{aligned}\Pi^{(2)}(s) = {} & -\frac{1}{2}g^4 M(x,x,x,x,x) - \frac{1}{2}g^4 V(x,x,x,x) + g^3 U(x,x,x,x) \\ & -\frac{1}{6}\lambda^2 S(x,x,x) + \frac{1}{4}\lambda g^2 B(x,x)B(x,x) + \frac{1}{4}\lambda^2 A(x)\left[A(x)/x + 1\right] \\ & -\frac{1}{2}\lambda g^2 A(x)B(x',x) - \frac{1}{4}\lambda g^2 I(x',x,x) \; ,\end{aligned} \tag{5.8}$$

[†] In a theory with gauge fields the self-energy is gauge-dependent, but the pole squared mass defined in this way is properly gauge invariant.

where $x = m^2$, and

$$\Pi^{(1)\prime}(s) = -\frac{1}{2}g^2\frac{\partial}{\partial s}B(x,x)\ . \tag{5.9}$$

Note that both $\Pi^{(1)\prime}$ and $\Pi^{(2)}$ have $(1-s/4x)^{-1/2}$ singularities at the threshold $s = 4x$, though this is not manifest in the above formulas. The `TSIL` code handles such true singularities by returning a value that is interpreted and displayed as the string `ComplexInfinity`.

Below is a sample C code that uses `TSIL` to calculate the pole squared mass for parameter values m^2, g, λ and Q^2 obtained as command-line inputs in that order. It first computes the required basis functions at $s = m^2$, then assembles $\Pi^{(1)}(m^2)$, $\Pi^{(2)}(m^2)$ and $\Pi^{(1)\prime}(m^2)$, and finally outputs the pole squared mass:

```
/*   === scalarpole.c ===
 *
 *   Command-line arguments are:
 *          scalar mass squared = x,
 *          cubic coupling = g,
 *          quartic coupling = lambda,
 *          renormalization scale squared = qq.
 *
 *    Run as, for example: ./spole 1 2 3 1
 */

#include <stdio.h>
#include "tsil.h"  /* Required TSIL header file */

#define PI 4.0L*TSIL_ATAN(1.0L) /* Uses arctan function defined in tsil.h */

int main (int argc, char *argv[])
{
  TSIL_DATA    result; /* Top-level TSIL data object */
  TSIL_REAL    qq;     /* Ensures correct basic type; see also TSIL_COMPLEX */
  TSIL_REAL    x, g, lambda;
  TSIL_COMPLEX pi1, pi1prime, pi2, s1, s2;
  TSIL_REAL    factor = 1.0L/(16.0L*PI*PI);

  /* If incorrect number of args, print message on stderr and exit: */
  if (argc != 5)
    TSIL_Error("main", "Expected 4 arguments: m^2, g, lambda, and Q^2", 1);

  /* Note cast to appropriate floating-point type for safety */
  x      = (TSIL_REAL) strtold(argv[1], (char **) NULL);
  g      = (TSIL_REAL) strtold(argv[2], (char **) NULL);
  lambda = (TSIL_REAL) strtold(argv[3], (char **) NULL);
  qq     = (TSIL_REAL) strtold(argv[4], (char **) NULL);

  /* All loop integrals have a common squared-mass argument x: */
  TSIL_SetParameters (&result, x, x, x, x, x, qq);

  /* For the pole mass calculation, evaluate two-loop integrals at s = x: */
  TSIL_Evaluate (&result, x);
```

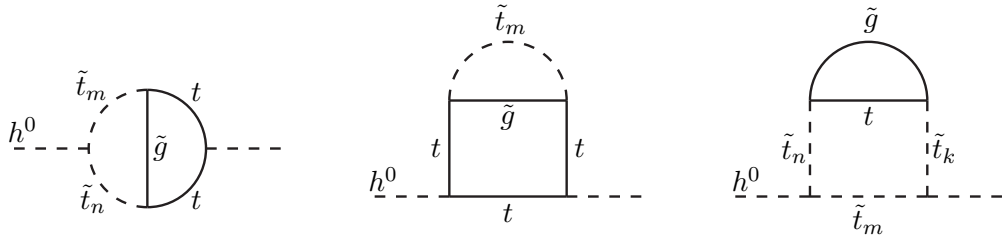

Figure 3: Feynman diagram topologies for some two-loop corrections to the neutral Higgs scalar boson self-energies in supersymmetry.

```
  /* Assemble one- and two-loop mass squared results: */
  pi1 = 0.5L*lambda*TSIL_A(x,qq) - 0.5L*g*g*TSIL_B(x,x,x,qq);

  pi1prime = -0.5L*g*g*TSIL_dBds(x, x, x, qq);

  pi2 = - 0.5L*g*g*g*g*TSIL_GetFunction(&result, "M")
        - 0.5L*g*g*g*g*TSIL_GetFunction(&result, "Vzxyv")
                + g*g*g*TSIL_GetFunction(&result, "Uzxyv")
    - (1.0L/6.0L)*lambda*lambda*TSIL_GetFunction(&result, "Svyz")
    + 0.25L*lambda*g*g*TSIL_POW(TSIL_GetFunction(&result, "Bxz"), 2)
    + 0.25L*lambda*lambda*TSIL_A(x,qq)*(TSIL_A(x,qq)/x + 1.0L)
    - 0.5L*lambda*g*g*TSIL_A(x,qq)*TSIL_Bp(x, x, x, qq)
    - 0.25L*lambda*g*g*TSIL_I2p(x,x,x,qq);

  s1 = x + factor*pi1;
  s2 = x + factor*pi1 + factor*factor*(pi2 + pi1*pi1prime);

  printf("Tree-level squared mass:    %lf\n", (double) x);
  printf("One-loop pole squared mass: %lf\n", (double) s1);
  printf("Two-loop pole squared mass: %lf\n", (double) s2);

  return 0;
}
```

Note the use of `TSIL_A`, `TSIL_I2p`, `TSIL_B`, `TSIL_Bp`, and `TSIL_dBds` to evaluate the functions $A(x)$, $I(x', x, x)$, $B(x, x)$, $B(x', x)$, and $\partial B(x, x)/\partial s$, respectively. [In the evaluation of `pi2`, we arbitrarily chose to use `TSIL_GetFunction(&result, "Bxz")` where `TSIL_B` could have been used.] The compile command for this program to produce the executable `spole` would typically be

```
cc -o spole scalarpole.c -L<dir> -ltsil -lm -DTSIL_SIZE_<size>
```

as discussed in subsection 5.1.

Other situations can be treated with appropriate generalizations. For example, in the Minimal Supersymmetric Standard Model correction to the neutral Higgs scalar boson self-energies [61], there are graphs with the topology shown in fig. 3 involving the top quark t, the top squarks $\tilde{t}_n$ for $n = 1, 2$, and the gluino $\tilde{g}$. All of the necessary one-loop and two-loop basis integrals for these diagram topologies can be computed in one fell swoop with:

```
TSIL_DATA result[2][2]; /* Declare an array of TSIL_DATA structs */
```

```
TSIL_REAL mt2, mstop2[2], mgluino2; /* Squared masses of top, stops, gluino */
TSIL_REAL s;
int i,j;
...
for (i=0; i<2; i++) {
  for (j=0; j<i; j++) {
   TSIL_SetParameters (&(result[i][j]),mstop2[i],mt2,mstop2[j],mt2,mgluino2,qq);
   TSIL_Evaluate (&(result[i][j]),s);
  }
}
```

The index of `mstop2[]` in the code is one less than the conventional top squark mass eigenstate label, because arrays start at index 0 in C. Note that the evaluation for $j = 1$, $i = 0$ would be redundant by symmetry with that for $j = 0$, $i = 1$, and so is not performed here. Note also that the subordinate integrals of topology S, T, and U are needed for the calculation of the self-energy and the pole mass, even though there are no Feynman diagrams with the corresponding topologies since there are no four-particle couplings involving fermions. The necessary two-loop integrals can then be extracted by, for example:

```
value = TSIL_GetFunction(&(result[0][0]), "M");
```
for $M(m^2_{\tilde{t}_1}, m^2_t, m^2_{\tilde{t}_1}, m^2_t, m_{\tilde{g}})$,

```
value = TSIL_GetFunction(&(result[0][0]), "Vuyxv");
```
for $V(m^2_t, m^2_t, m^2_{\tilde{t}_1}, m_{\tilde{g}})$,

```
value = TSIL_GetFunction(&(result[1][0]), "Vzxyv");
```
for $V(m^2_{\tilde{t}_1}, m^2_{\tilde{t}_2}, m^2_t, m_{\tilde{g}})$,

```
value = TSIL_GetFunction(&(result[1][0]), "Vxzuv");
```
for $V(m^2_{\tilde{t}_2}, m^2_{\tilde{t}_1}, m^2_t, m_{\tilde{g}})$,

```
value = TSIL_GetFunction(&(result[0][0]), "Txuv");
```
for $T(m^2_{\tilde{t}_1}, m^2_t, m_{\tilde{g}})$,

etc.

5.4 Using TSIL with C++

`TSIL` functions can be called from C++ code using `libtsil.a`. The header file `tsil_cpp.h` should be included in any C++ source files that make use of `TSIL` functions. This file is equivalent to the usual `tsil.h`, but with additional definitions to streamline interoperation with C++.

In particular, `tsil_cpp.h` provides wrappers for `TSIL` functions that insure proper handling of complex values, which are of generic type `_Complex` in C and `std::complex<>` in C++. The relevant standards guarantee that pointers to these types will be interpreted correctly in any context, and the wrappers insure that complex arguments and return values are always passed between C and C++ in this way.

What this means for the user is that the `TSIL` functions all have C++-specific versions that can be called with C++ types as arguments and will return C++ types. The names of these are the same as the corresponding TSIL functions, with a trailing underscore. Thus the C function

```
TSIL_GetFunction (...)
```

becomes

```
TSIL_GetFunction_ (...)
```

when called from C++, etc. All functions in the user API have been supplied with such wrappers, even though not all functions really need them; this is so that users need not remember which functions have special names.

See the TSIL documentation file README.txt for additional details on using TSIL with C++.

5.5 Using TSIL with FORTRAN

TSIL functions can be called from Fortran programs, and utilities for this are included with the package. Basic functionality is provided by a "wrapper" function tsilfevaluate, which is called as a subroutine from a Fortran program. This subroutine implements the most general TSIL calculation: it takes as arguments x, y, z, u, v, Q^2, s and returns the values of all basis functions, including $\overline{T}$, V, and "bold" functions.

The results are returned to the calling program in a COMMON block, which corresponds to a special C struct used in tsilfevaluate. This COMMON block contains a number of pre-defined arrays that hold the various function values. Definitions of the COMMON block and subsidiary arrays are given in a header file that users include in their Fortran programs (tsil_fort.inc). In addition, this header file defines a set of integer variables that allow items in the COMMON block to be referred to by name.

A Fortran program fragment that uses these utilities is shown below:

```
      PROGRAM ftest
c     Includes array and COMMON definitions:
      INCLUDE 'tsil_fort.inc'

c     (Code setting values of x,y,z,u,v,qq,s not shown)
      ...
c     Evaluate basis integrals:
      CALL tsilfevaluate(x,y,z,u,v,qq,s)

c     Print a representative value:
      PRINT *, U(xzuv)
      ...
```

The provided wrapper code can serve as a model for users wishing to write their own interface routines with additional functionality. The TSIL documentation contains a detailed discussion of the issues that arise in using TSIL with Fortran.

6 Outlook

The TSIL library is intended to provide a convenient all-purpose solution to the problem of numerical evaluation of two-loop self-energy integrals in high-energy physics. The functions provided should work for arbitrary input parameters, without relying on special mass hierarchies or other

simplifications that can arise in special cases. The library does take advantage of simplifications when they allow evaluation in terms of polylogarithms.

The library is organized around the calculation of the basis integrals to which all other self-energy contributions can be reduced by known algorithms. It is therefore the responsibility of the user to perform the non-trivial calculations necessary to assemble the basis functions into physical observables. This involves the reduction of the Feynman diagrams to the basis integrals and the inclusion of counterterms, tasks that can always be automated using purely symbolic computer algebra manipulations or even performed by hand in favorable situations. We believe that keeping these separate from the problem of numerical evaluation is advantageous, and that the modular nature of this approach will afford the flexibility to deal with the surprises that hopefully await us as we explore physics at the TeV scale.

Appendix A: Comparison of conventions with other sources

In this Appendix, we list the correspondence between our notation for the loop integrals and those found in several other references. In the following, the functions as defined in this paper and in refs. [4],[5] always appear on the left-hand side, and equivalent notations for them in other papers appear on the right.

The definition of the master integral in ref. [16] is given by:

$$M(x,y,z,u,v) \;=\; -I(s)/s, \tag{A.1}$$

with $m_1^2, m_2^2, m_3^2, m_4^2, m_5^2 = x, z, v, y, u$, and the integral on the right-hand side defined in 4 dimensions.

The notation used in refs. [3, 9, 18, 45, 47] is:

$$\begin{aligned}
\mathbf{A}(x) &= -A_0(x) && \text{(A.2)}\\
\mathbf{B}(x,y) &= B_0(s;x,y) && \text{(A.3)}\\
\mathbf{I}(x,y,z) &= -T_{135}(x,y,z) && \text{(A.4)}\\
\mathbf{S}(x,y,z) &= -T_{234}(s;x,y,z) && \text{(A.5)}\\
\mathbf{T}(x,y,z) &= T_{2234}(s;x,x,y,z) && \text{(A.6)}\\
\mathbf{U}(x,y,z,u) &= T_{1234}(s;y,x,z,u) && \text{(A.7)}\\
\mathbf{V}(x,y,z,u) &= -T_{11234}(s;y,y,x,z,u) && \text{(A.8)}\\
\mathbf{M}(x,y,z,u,v) &= -T_{12345}(s;x,z,v,u,y). && \text{(A.9)}
\end{aligned}$$

Ref. [46] used the same notation, with the exception that $\mathbf{S}(x,y,z) = -T_{123}(s;x,y,z)$ there.

The notation in ref. [1] is:

$$\begin{aligned}
\mathbf{A}(x) &= iaT_1^{(d)} \quad \text{with } m_1^2 = x, && \text{(A.10)}\\
\mathbf{B}(x,y) &= -iaG_{11}^{(d)}(s) \quad \text{with } m_1^2, m_2^2 = x, y, && \text{(A.11)}
\end{aligned}$$

$$\mathbf{I}(x,y,z) = a^2 J^{(d)}_{111}(0) \quad \text{with } m_1^2, m_2^2, m_3^2 = x, y, z, \tag{A.12}$$

$$\mathbf{S}(x,y,z) = a^2 J^{(d)}_{111}(s) \quad \text{with } m_1^2, m_2^2, m_3^2 = x, y, z, \tag{A.13}$$

$$\mathbf{T}(x,y,z) = -a^2 J^{(d)}_{211}(s) \quad \text{with } m_1^2, m_2^2, m_3^2 = x, y, z, \tag{A.14}$$

$$\mathbf{U}(x,y,z,u) = -a^2 V^{(d)}_{1111}(s) \quad \text{with } m_1^2, m_2^2, m_3^2, m_4^2 = u, x, z, y, \tag{A.15}$$

$$\mathbf{V}(x,y,z,u) = a^2 V^{(d)}_{1112}(s) \quad \text{with } m_1^2, m_2^2, m_3^2, m_4^2 = u, x, z, y, \tag{A.16}$$

$$\mathbf{M}(x,y,z,u,v) = a^2 F^{(d)}_{11111}(s) \quad \text{with } m_1^2, m_2^2, m_3^2, m_4^2, m_5^2 = x, y, z, u, v, \tag{A.17}$$

and the closely related notation of ref. [2] is

$$\mathbf{A}(x) = ia\texttt{TAI}[\texttt{d,s,}\{\{1,\sqrt{x}\}\}], \tag{A.18}$$

$$\mathbf{B}(x,y) = -ia\texttt{TBI}[\texttt{d,s,}\{\{1,\sqrt{x}\},\{1,\sqrt{y}\}\}] \tag{A.19}$$

$$\mathbf{I}(x,y,z) = a^2\texttt{TJI}[\texttt{d,0,}\{\{1,\sqrt{x}\},\{1,\sqrt{y}\},\{1,\sqrt{z}\}\}] \tag{A.20}$$

$$\mathbf{S}(x,y,z) = a^2\texttt{TJI}[\texttt{d,s,}\{\{1,\sqrt{x}\},\{1,\sqrt{y}\},\{1,\sqrt{z}\}\}] \tag{A.21}$$

$$\mathbf{T}(x,y,z) = -a^2\texttt{TJI}[\texttt{d,s,}\{\{2,\sqrt{x}\},\{1,\sqrt{y}\},\{1,\sqrt{z}\}\}] \tag{A.22}$$

$$\mathbf{U}(x,y,z,u) = -a^2\texttt{TVI}[\texttt{d,s,}\{\{1,\sqrt{u}\},\{1,\sqrt{x}\},\{1,\sqrt{z}\},\{1,\sqrt{y}\}\}] \tag{A.23}$$

$$\mathbf{V}(x,y,z,u) = a^2\texttt{TVI}[\texttt{d,s,}\{\{1,\sqrt{u}\},\{1,\sqrt{x}\},\{1,\sqrt{z}\},\{2,\sqrt{y}\}\}] \tag{A.24}$$

$$\mathbf{M}(x,y,z,u,v) = a^2\texttt{TFI}[\texttt{d,s,}\{\{1,\sqrt{x}\},\{1,\sqrt{y}\},\{1,\sqrt{z}\},\{1,\sqrt{u}\},\{1,\sqrt{v}\}\}] \tag{A.25}$$

where $a = (4\pi\mu^2)^{2-d/2}$.

The notation of [32, 53, 55, 56, 57] (ref. [54] used a slightly different normalization) is:

$$\mathbf{A}(x) = bT(d,x) \tag{A.26}$$

$$\mathbf{B}(x,y) = bS(d,x,y,-s) \tag{A.27}$$

$$\mathbf{I}(x,y,z) = b^2 V(d,x,y,z) \tag{A.28}$$

$$\mathbf{S}(x,y,z) = b^2 F_0(d,x,y,z,-s) \tag{A.29}$$

$$\mathbf{T}(x,y,z) = b^2 F_1(d,x,y,z,-s) = b^2 F_2(d,z,x,y,-s) = b^2 F_3(d,y,z,x,-s) \tag{A.30}$$

$$\mathbf{U}(x,y,z,u) = b^2 F_4(d,x,u,y,z,-s), \tag{A.31}$$

where $b = 4\mu^{4-d}$.

The notation of [50] is:

$$\mathbf{S}(x,y,z) = c^2 S^{111} \quad \text{with } m_1^2, m_2^2, m_3^2 = x, y, z, \tag{A.32}$$

$$\mathbf{U}(x,y,z,u) = c^2 S^{121} \quad \text{with } m_1^2, m_2^2, m_3^2, m_4^2 = z, u, y, x, \tag{A.33}$$

$$\mathbf{M}(x,y,z,u,v) = c^2 S^{221} \quad \text{with } m_1^2, m_2^2, m_3^2, m_4^2, m_5^2 = x, z, v, y, u, \tag{A.34}$$

with $c = (2\pi)^{d-4}$.

Appendix B: Analytic expressions for some special cases

In this Appendix, we present some analytic formulas for some important two-loop basis integrals special cases, two of which do not seem to have appeared before in the literature, and others that

are equivalent to known results.

As a generalization of the integral $M(0,0,x,x,0)$ computed in ref. [16], we find that for $x \geq y$:

$$\begin{aligned} M(0,0,x,y,0) &= \Big[3\mathrm{Li}_3(r_x) + 3\mathrm{Li}_3(r_y) - 3\mathrm{Li}_3(r_y/r_x) - 3\mathrm{Li}_3(yr_y/xr_x) \\ &+3\mathrm{Li}_3(y/x) - 3\zeta(3) + [\ln(r_y) - \ln(r_x) + 2\ln(y/x)]\mathrm{Li}_2(yr_y/xr_x) \\ &+[\ln(r_y) - \ln(r_x) - 2\ln(y/x)]\mathrm{Li}_2(y/x) + [\ln(r_y) - 3\ln(r_x)]\mathrm{Li}_2(r_x) \\ &+[\ln(r_x) - 3\ln(r_y)]\mathrm{Li}_2(r_y) + 3[\ln(r_y) - \ln(r_x)]\mathrm{Li}_2(r_y/r_x) \\ &+[\ln(r_x) - \ln(r_y)][\ln(r_x) - \ln(r_y) + \ln(x/y)]\ln(1 - y/x) \\ &- \ln^3(r_x) + 2\ln^2(r_x)\ln(r_y) - \ln(r_x)\ln^2(r_y) - \ln(r_x)\ln(r_y)\ln(1 - r_x) \\ &+ \ln(r_x)\ln(r_y)\ln(y/x) - \ln(r_x)\ln(y/x)[\ln(r_x) + \ln(y/x)]/2 \\ &-\zeta(2)[3\ln(r_x) + \ln(r_y)]\Big]/s \end{aligned} \tag{B.1}$$

where $r_x = 1 - s/x$ and $r_y = 1 - s/y$ are each given an infinitesimal negative imaginary part. The result for $x < y$ is obtained by interchanging the squared masses.

Generalizing the result $M(0,x,x,0,x)|_{s=x}$ found in ref. [16], we obtain threshold cases:

$$\begin{aligned} M(0,y,y,0,x)|_{s=x} &= \Big[2\mathrm{Li}_3([r-1]/[r+1]) - 2\mathrm{Li}_3([1-r]/[r+1]) - 2\mathrm{Li}_3(r/[r+1]) \\ &-2\mathrm{Li}_3(1 - 1/r) - \mathrm{Li}_3(1/r^2)/4 + 2[\ln(2r) - \ln(r+1)]\mathrm{Li}_2(r/[r+1]) \\ &+[4\ln(r-1) - 2\ln(2r) - 2\ln(r+1)]\mathrm{Li}_2(1 - 1/r) \\ &+2[\ln(r-1) - \ln(r+1)][\mathrm{Li}_2([1-r]/2) - \mathrm{Li}_2([r-1]/2r) \\ &- \ln(4r)\mathrm{Li}_2(1/r^2)/2 + \ln^2(r)[3\ln(r-1) - \ln(r+1) - 6\ln 2]/2 \\ &+ \ln(r)[2\ln(r+1)\ln(2[r-1]) - \ln^2(r-1) - 6\zeta(2)] - (4/3)\ln^3(r) \\ &-(2/3)\ln^3(r+1) + \ln 2\ln^2(r+1) + \zeta(3)/2 + 6\zeta(2)\ln(1+r)\Big]/x, \end{aligned} \tag{B.2}$$

$$\begin{aligned} M(0,x,y,0,y)|_{s=x} &= \Big[2\mathrm{Li}_3([\sqrt{r}-1]/[\sqrt{r}+1]) - 2\mathrm{Li}_3([1-\sqrt{r}]/[\sqrt{r}+1]) \\ &+2\mathrm{Li}_3(1-r) - 2\mathrm{Li}_3([r-1]/r) - 3\zeta(3)/2 \\ &+[\ln(r) - 2\ln(r-1)]\mathrm{Li}_2(1-r) - [\ln^2(r-1)\ln(r)]/2 \\ &+[\ln^3(r)]/3 - 2\zeta(2)\ln(r) + 6\zeta(2)\ln(1+\sqrt{r})\Big]/x, \end{aligned} \tag{B.3}$$

where $r = y/x$ is given an infinitesimal negative imaginary part to get the correct branches. Eq. (B.3) is equivalent, by use of the recurrence relations of [1], to results already obtained in [36, 37].

We also note the following threshold and pseudo-threshold values (equivalent to results obtained in [38]; see also [39]):

$$\begin{aligned} U(x,0,y,y)|_{s=x} &= 11/2 - 3\overline{\ln}x + \overline{\ln}x\overline{\ln}y - (\overline{\ln}y)^2/2 + (1+y/x)[\zeta(2) - \mathrm{Li}_2(1-x/y)] \\ &-4r\{\mathrm{Li}_2([1-r]/[1+r]) - \mathrm{Li}_2([r-1]/[r+1]) + 3\zeta(2)/2\} \end{aligned} \tag{B.4}$$

where $r = \sqrt{y/x}$, and

$$\begin{aligned} U(y,0,y,x)|_{s=x} &= 11/2 - 2\overline{\ln}x - \overline{\ln}y + (\overline{\ln}y)^2/2 - 2\zeta(2)(1+y/x) \\ &+(\overline{\ln}x - 1 + y/x[1 - \overline{\ln}y])\ln(1 - x/y - i\varepsilon) + (1+2y/x)\mathrm{Li}_2(1-x/y). \end{aligned} \tag{B.5}$$

Appendix C: The TSIL Application Programmer Interface

This Appendix lists the functions in the TSIL package, and their basic functionality. Complete details may be found in the TSIL documentation and header files.

Basic evaluation functions:

TSIL_SetParameters	Sets parameters x, y, z, u, v, Q^2 and selects evaluation of all integral functions
TSIL_SetParametersSTU	Sets parameters x, z, u, v, Q^2 and selects evaluation of S, T, U functions only. (New in v1.1 November 2006; see section 5.2.)
TSIL_SetParametersST	Sets parameters x, u, v, Q^2 and selects evaluation of S, T functions only. (New in v1.1 November 2006; see section 5.2.)
TSIL_Evaluate	Evaluate integral functions for a specified s
TSIL_GetStatus	Return current evaluation status
TSIL_GetData	Extract a set of integral function values to an array
TSIL_GetBoldData	Extract a set of bold integral function values to an array
TSIL_GetFunction	Return a single specified integral function value
TSIL_GetBoldFunction	Return a single specified bold integral function value

I/O related functions:

TSIL_PrintStatus	Print evaluation status to stdout
TSIL_PrintData	Print all integral function values to stdout
TSIL_WriteData	Write all integral function values to a file
TSIL_PrintDataM	As TSIL_PrintData, but format is valid Mathematica input
TSIL_WriteDataM	As TSIL_WriteData, but format is valid Mathematica input
TSIL_cprintf	Generic printing of values of TSIL_Complex type
TSIL_cprintfM	As TSIL_cprintf, but in Mathematica input form
TSIL_Error	Print a message to stderr and exit
TSIL_Warn	Print a warning message to stderr
TSIL_WarnsOff	Toggles warning messages off. (New in v1.3 June 2015.)
TSIL_WarnsOn	Toggles warning messages on. (New in v1.3 June 2015.)
TSIL_PrintInfo	Write general information on stdout

Utilities:

TSIL_ResetStepSizeParams	Sets new parameters used for Runge-Kutta step size control
TSIL_IsInfinite	Tests whether argument of type TSIL_Complex is finite
TSIL_DataSize	Returns size of intrinsic floating point data used
TSIL_PrintVersion	Prints version number of TSIL
TSIL_CopyResult	Copies data from a TSIL_DATA struct to a TSIL_RESULT struct. (New in v1.2 July 2014, see end of section 5.2.)
TSIL_PermuteResult	Copies the data from one TSIL_RESULT struct to another, with

	the option of permuting the squared masses according to either $(x,z) \leftrightarrow (y,u)$, or $(x,y) \leftrightarrow (z,u)$, or $(x,y) \leftrightarrow (u,z)$, or doing no permutation. (New in v1.2 July 2014, see end of section 5.2.)
`TSIL_NumFuncs`	Returns number of basis functions of specified type. (New in v1.3 June 2015.)

Analytic cases:

`TSIL_Dilog`	Dilogarithm function of complex argument $\mathrm{Li}_2(z)$
`TSIL_Trilog`	Trilogarithm function of complex argument $\mathrm{Li}_3(z)$
`TSIL_A`	One-loop vacuum integral $A(x)$
`TSIL_Ap`	One-loop vacuum integral $A(x') = \overline{\mathrm{ln}}(x)$ (New in v1.2 July 2014.)
`TSIL_Aeps`	One-loop vacuum integral $A_\epsilon(x)$
`TSIL_B`	One-loop self-energy integral $B(x,y)$
`TSIL_Bp`	$B(x',y)$
`TSIL_dBds`	$\partial B(x,y)/\partial s$
`TSIL_Beps`	One-loop self-energy integral $B_\epsilon(x,y)$
`TSIL_I2`	Two-loop vacuum integral $I(x,y,z)$
`TSIL_I2p`	$I(x',y,z)$
`TSIL_I2p2`	$I(x'',y,z)$
`TSIL_I2pp`	$I(x',y',z)$
`TSIL_I2p3`	$I(x''',y,z)$
`TSIL_Sanalytic`	Analytic evaluation of S if available
`TSIL_Tanalytic`	Analytic evaluation of T if available
`TSIL_Tbaranalytic`	Analytic evaluation of $\overline{T}$ if available
`TSIL_Uanalytic`	Analytic evaluation of U if available
`TSIL_Vanalytic`	Analytic evaluation of V if available
`TSIL_Manalytic`	Analytic evaluation of M if available

Fortran interface function:

`tsilfevaluate_`	Wrapper for `TSIL_Evaluate`, callable from Fortran

Acknowledgments: We are grateful to M. Kalmykov for bringing to our attention previous results on some of the cases in Appendix B. The work of SPM was supported in part by the National Science Foundation grant number PHY-0140129. DGR was supported by an award from Research Corporation, and by a grant from the Ohio Supercomputer Center.

References

[1] O. V. Tarasov, Nucl. Phys. B **502**, 455 (1997) [hep-ph/9703319]. A different algorithm was given in [3].

[2] R. Mertig and R. Scharf, Comput. Phys. Commun. **111**, 265 (1998) [hep-ph/9801383].

[3] G. Weiglein, R. Scharf and M. Bohm, Nucl. Phys. B **416**, 606 (1994) [hep-ph/9310358].

[4] S.P. Martin, Phys. Rev. D **68**, 075002 (2003) [hep-ph/0307101].

[5] S.P. Martin, Phys. Rev. D **70**, 016005 (2004) [hep-ph/0312092].

[6] G. 't Hooft and M. J. Veltman, Nucl. Phys. B **44**, 189 (1972); W. A. Bardeen, A. J. Buras, D. W. Duke and T. Muta, Phys. Rev. D **18**, 3998 (1978).

[7] W. Siegel, Phys. Lett. B **84**, 193 (1979); D. M. Capper, D.R.T. Jones and P. van Nieuwenhuizen, Nucl. Phys. B **167**, 479 (1980). I. Jack et al, Phys. Rev. D **50**, 5481 (1994) [hep-ph/9407291].

[8] U. Nierste, D. Muller and M. Bohm, Z. Phys. C **57**, 605 (1993).

[9] R. Scharf and J. B. Tausk, Nucl. Phys. B **412**, 523 (1994).

[10] L. Lewin, "Polylogarithms and associated functions" (Elsevier North Holland, New York, 1981). A formula useful for fast and accurate numerical calculation of polylogarithms appears as Proposition 1 of H. Cohen, L. Lewin and D. Zagier, Experiment. Math. **1**, 25, (1992).

[11] R. Scharf, Würzburg Diploma Thesis, as quoted in [9].

[12] J.L. Rosner, Ann. Phys. **44**, 11 (1967).

[13] K. G. Chetyrkin, A. L. Kataev and F. V. Tkachov, Nucl. Phys. B **174**, 345 (1980); K. G. Chetyrkin and F. V. Tkachov, Nucl. Phys. B **192**, 159 (1981).

[14] J. van der Bij and M. J. Veltman, Nucl. Phys. B **231**, 205 (1984).

[15] A. Djouadi, Nuovo Cim. A **100**, 357 (1988).

[16] D. J. Broadhurst, Z. Phys. C **47**, 115 (1990).

[17] B. A. Kniehl, Nucl. Phys. B **347**, 86 (1990).

[18] F. A. Berends and J. B. Tausk, Nucl. Phys. B **421**, 456 (1994).

[19] J. Fleischer, A. V. Kotikov and O. L. Veretin, Nucl. Phys. B **547**, 343 (1999) [hep-ph/9808242].

[20] S. Laporta and E. Remiddi, [hep-ph/0406160].

[21] V. A. Smirnov, Commun. Math. Phys. **134**, 109 (1990).

[22] A. I. Davydychev and J. B. Tausk, Nucl. Phys. B **397**, 123 (1993).

[23] D. J. Broadhurst, J. Fleischer and O. V. Tarasov, Z. Phys. C **60**, 287 (1993) [hep-ph/9304303].

[24] A. I. Davydychev, V. A. Smirnov and J. B. Tausk, Nucl. Phys. B **410**, 325 (1993) [hep-ph/9307371].

[25] F. A. Berends, A. I. Davydychev, V. A. Smirnov and J. B. Tausk, Nucl. Phys. B **439**, 536 (1995) [hep-ph/9410232].

[26] F. A. Berends, A. I. Davydychev and V. A. Smirnov, Nucl. Phys. B **478**, 59 (1996) [hep-ph/9602396].

[27] A. Czarnecki and V. A. Smirnov, Phys. Lett. B **394**, 211 (1997) [hep-ph/9608407].

[28] M. Beneke and V. A. Smirnov, Nucl. Phys. B **522**, 321 (1998) [hep-ph/9711391].

[29] F. A. Berends, A. I. Davydychev and N. I. Ussyukina, Phys. Lett. B **426**, 95 (1998) [hep-ph/9712209].

[30] A. I. Davydychev and V. A. Smirnov, Nucl. Phys. B **554**, 391 (1999) [hep-ph/9903328].

[31] J. Fleischer, M. Y. Kalmykov and A. V. Kotikov, Phys. Lett. B **462**, 169 (1999) [hep-ph/9905249].

[32] M. Caffo, H. Czyz and E. Remiddi, Nucl. Phys. B **581**, 274 (2000) [hep-ph/9912501], Nucl. Phys. B **611**, 503 (2001) [hep-ph/0103014].

[33] S. Groote and A. A. Pivovarov, Nucl. Phys. B **580**, 459 (2000) [hep-ph/0003115].

[34] F. Jegerlehner, M.Y. Kalmykov and O. Veretin, Nucl. Phys. B **641**, 285 (2002) [hep-ph/0105304].

[35] F. Jegerlehner, M. Y. Kalmykov and O. Veretin, Nucl. Phys. B **658**, 49 (2003) [hep-ph/0212319].

[36] J. Fleischer, F. Jegerlehner, O. V. Tarasov and O. L. Veretin, Nucl. Phys. B **539**, 671 (1999) [Erratum-ibid. B **571**, 511 (2000)] [hep-ph/9803493].

[37] F. Jegerlehner and M. Y. Kalmykov, Nucl. Phys. B **676**, 365 (2004) [hep-ph/0308216].

[38] N. Gray, D. J. Broadhurst, W. Grafe and K. Schilcher, Z. Phys. C **48**, 673 (1990).

[39] A. I. Davydychev and A. G. Grozin, Phys. Rev. D **59**, 054023 (1999) [hep-ph/9809589].

[40] C. Ford and D.R.T. Jones, Phys. Lett. B **274**, 409 (1992); C. Ford, I. Jack and D.R.T. Jones, Nucl. Phys. B **387**, 373 (1992) [hep-ph/0111190].

[41] D. Kreimer, Phys. Lett. B **273**, 277 (1991).

[42] F. A. Berends, M. Buza, M. Bohm and R. Scharf, Z. Phys. C **63**, 227 (1994).

[43] A. Ghinculov and J. J. van der Bij, Nucl. Phys. B **436**, 30 (1995) [hep-ph/9405418].

[44] A. Czarnecki, U. Kilian and D. Kreimer, Nucl. Phys. B **433**, 259 (1995) [hep-ph/9405423].

[45] S. Bauberger et al, Nucl. Phys. Proc. Suppl. **37B**, 95 (1994) [hep-ph/9406404].

[46] S. Bauberger, F. A. Berends, M. Bohm and M. Buza, Nucl. Phys. B **434**, 383 (1995) [hep-ph/9409388].

[47] S. Bauberger and M. Bohm, Nucl. Phys. B **445**, 25 (1995) [hep-ph/9501201].

[48] A. Ghinculov and Y. P. Yao, Nucl. Phys. B **516**, 385 (1998) [hep-ph/9702266]; Phys. Rev. D **63**, 054510 (2001) [hep-ph/0006314].

[49] G. Passarino, Nucl. Phys. B **619**, 257 (2001) [hep-ph/0108252].

[50] G. Passarino and S. Uccirati, Nucl. Phys. B **629**, 97 (2002) [hep-ph/0112004].

[51] A. V. Kotikov, Phys. Lett. B **254**, 158 (1991), Phys. Lett. B **259**, 314 (1991).

[52] E. Remiddi, Nuovo Cim. A **110**, 1435 (1997) hep-th/9711188.

[53] M. Caffo, H. Czyz, S. Laporta and E. Remiddi, Nuovo Cim. A **111**, 365 (1998) hep-th/9805118.

[54] M. Caffo, H. Czyz, S. Laporta and E. Remiddi, Acta Phys. Polon. B **29**, 2627 (1998) hep-th/9807119.

[55] M. Caffo, H. Czyz and E. Remiddi, Nucl. Phys. B **634**, 309 (2002) [hep-ph/0203256].

[56] M. Caffo, H. Czyz and E. Remiddi, "Numerical evaluation of master integrals from differential equations," [hep-ph/0211178].

[57] M. Caffo, H. Czyz, A. Grzelinska and E. Remiddi, Nucl. Phys. B **681**, 230 (2004) [hep-ph/0312189].

[58] J.C. Butcher, "The Numerical Analysis of Ordinary Differential Equations: Runge-Kutta and General Linear Methods," John Wiley and Sons, New York, 1987.

[59] E. Fehlberg, NASA Technical Report, NASA TR R-315, (1969).

[60] J.R. Cash and A.H. Karp, ACM Transactions on Mathematical Software, **16**, 222, (1990).

[61] S. P. Martin, "Strong and Yukawa two-loop contributions to Higgs scalar boson self-energies and pole masses in supersymmetry," Phys. Rev. D to appear, [hep-ph/0405022].