I. `TensorReduce` DEMONSTRATION

In Section III, we motivate and expound upon the tensor reduction procedure, but let us first demonstrate how the `TensorReduce` function is used (for those in the know). Consider the following four-point, three-loop integral in $D = 4 - 2\epsilon$ dimensions, i.e., four space-time dimensions with dimensional regularization:

$$\int \frac{d^{4-2\epsilon}\ell_1}{(2\pi)^{4-2\epsilon}} \frac{d^{4-2\epsilon}\ell_2}{(2\pi)^{4-2\epsilon}} \frac{d^{4-2\epsilon}\ell_3}{(2\pi)^{4-2\epsilon}} \; \frac{(\ell_1 \cdot k_1)(\ell_1 \cdot \varepsilon_2)(\ell_1 \cdot \varepsilon_3)(\ell_2 \cdot k_2)(\ell_2 \cdot \varepsilon_1)(\ell_2 \cdot \varepsilon_4)(\ell_3 \cdot k_1)(\ell_3 \cdot k_3)}{(\ell_1^2 - m^2)^4(\ell_2^2 - m^2)^4(\ell_3^2 - m^2)^2}, \tag{1.1}$$

where ℓ_i are independent loop momenta, k_i are external momenta, ε_i are external polarization vectors and m is a uniform mass regulator, used to regulate infrared divergences. We can determine the tensor reduction of the numerator using the `TensorReduce` function from our `Mathematica` package `MasterFunctions`. In the `Wolfram Language` (`Mathematica`), we reduce the numerator of Eq. (1.1) as follows:

```
numerator = lk[1,1]le[1,2]le[1,3]lk[2,2]le[2,1]le[2,4]lk[3,1]lk[3,3];
reduction = TensorReduce[ numerator ];
reducedNumerator = Plus@@( Times@@@reduction );                    (1.2)
```

For this example, `reduction` is a 6×2 matrix, where the first column corresponds to the six unique scalar products of loop momenta,

$$\begin{aligned} &\ell_1^2\,(\ell_1 \cdot \ell_2)\,\ell_2^2\,\ell_3^2\,, && (\ell_1 \cdot \ell_2)\,(\ell_1 \cdot \ell_3)^2\,\ell_2^2\,, && \ell_1^2\,(\ell_1 \cdot \ell_3)\,\ell_2^2\,\ell_3^2\,, \\ &(\ell_1 \cdot \ell_2)^3\,\ell_3^2\,, && (\ell_1 \cdot \ell_2)^2\,(\ell_1 \cdot \ell_3)\,(\ell_2 \cdot \ell_3)\,, && \ell_1^2\,(\ell_1 \cdot \ell_2)\,(\ell_2 \cdot \ell_3)^2\,, \end{aligned} \tag{1.3}$$

and the second column of `reduction` corresponds to their coefficients after tensor reduction.The last line of Eq. (1.2) multiples the scalar products by their appropriate coefficients before summing them. (This explicit step can be circumvented in practice.) The final reduced numerator will integrate to the same result as the original numerator of Eq. (1.1).

II. `TensorReduce` PERFORMANCE

Often we will encounter not just a single integral but millions of integrals like Eq. (1.1). To compound the problem, at high loop orders, each integral can involve millions of calls to a metric contraction subroutine in order to generate systems of linear equations. Then, these often-complicated systems must be solved analytically. While one may need to trade the `Wolfram Language` for `C++` once the problem becomes sufficiently large, we push the `Wolfram Language` as far as we can here. This allows for the convenience of a seamless interface with our existing `MasterFunctions` tools.

First, when a sum of many numerator expressions (as opposed to the single term of Eq. (1.2)) is passed into `TensorReduce`, the function gathers like terms to avoid performing duplicate work. Furthermore, it distills and canonicalizes the input expressions into the minimal amount of necessary data. For instance, Eq. (1.2) is distilled into `{3,3,2}`. This reflects the fact that there are three, three and two occurrences of different loop momenta in the numerator. We are always able to sort these lists as well without losing information. For example, `le[2,1]lk[1,2]le[1,3]le[1,4]le[2,2]lk[1,1]` and `lk[2,1]le[1,2]le[2,3]le[1,4]le[2,1]lk[2,3]` would both be processed as `{4,2}`.

Once a reduction is processed, the final and intermediate results are saved in `MasterFunctions/aux/tensorReduction/` for later retrieval. We will discuss the intermediate steps below, but we note here that the example of Eq. (1.2) saves six results: `12.m` will be stored in the `splitByTwos` subdirectory (since we deal with a total of twelve powers of loop momenta), and different files named `3.3.2.m` will be stored in the subdirectories `tensorStructs`, `scalarProducts`, `tensorContractions`, `equations` and `solutions`.

Nearly all of our subroutines harness the parallel computing capabilities of the `Wolfram Language`. Namely, the `ParallelTable` function is used generously. Furthermore, we devote special care to the subroutine `ContractionCycles`, the real workhorse of the tensor reduction. This subroutine handles metric tensor contractions and might be called millions of times per integral for millions of integrals. We reduce the problem of contracting metric tensors to that of finding the number of connected components in an undirected graph. By writing an efficient algorithm with good constants and using the `Compile` feature to generate precompiled `C` code, we can speed up the computation by at least a factor of ten compared to native functions in the `Wolfram Language`.

III. TENSOR REDUCTION EXPLAINED

Let us now motivate and expound upon the tensor reduction procedure. Eq. (1.1) will again be used as the working example. Note that all of the functions that we mention below are private; `TensorReduce` is the only public function.

First, to see the tensor nature of Eq. (1.1), we strip off the external momenta, k_i, and polarization vectors, ε_i. This leaves a tensor integral depending on only the loop momenta and a uniform mass regulator, m,

$$\mathcal{I}^{\mu_1\mu_2\mu_3|\nu_1\nu_2\nu_3|\rho_1\rho_2} \equiv \int \frac{d^{4-2\epsilon}\ell_1}{(2\pi)^{4-2\epsilon}} \frac{d^{4-2\epsilon}\ell_2}{(2\pi)^{4-2\epsilon}} \frac{d^{4-2\epsilon}\ell_3}{(2\pi)^{4-2\epsilon}} \, \frac{\ell_1^{\mu_1}\,\ell_1^{\mu_2}\,\ell_1^{\mu_3}\,\ell_2^{\nu_1}\,\ell_2^{\nu_2}\,\ell_2^{\nu_3}\,\ell_3^{\rho_1}\,\ell_3^{\rho_2}}{(\ell_1^2-m^2)^4(\ell_2^2-m^2)^4(\ell_3^2-m^2)^2} \,. \qquad (3.1)$$

We can decompose this integral into a sum of scalar integrals (with tensor coefficients), which are much easier to evaluate. By exploiting Lorentz invariance, we know that the only possible tensors present after integration are Minkowski metric tensors, $\eta^{\alpha\beta}$.

Therefore, we need to find the possible products of metric tensors that pair all of the Lorentz indices of Eq. (3.1). The function `SplitByTwos`—called by `GetSBT` if a saved result cannot be retrieved—is a simple recursive function (taken from Tristan Dennen) that constructs all possible pairings of indices. In our example, the indices of Eq. (3.1) can be paired 105 ways as,

$$\begin{aligned} &\mu_1,\mu_2,\mu_3,\nu_1,\nu_2,\nu_3,\rho_1,\rho_2 \\ &\qquad \to \eta^{\mu_1\mu_2}\eta^{\mu_3\nu_1}\eta^{\nu_2\nu_3}\eta^{\rho_1\rho_2},\ \eta^{\mu_1\mu_2}\eta^{\mu_3\nu_1}\eta^{\nu_2\rho_1}\eta^{\nu_3\rho_2},\ \ldots,\ \eta^{\mu_1\rho_2}\eta^{\mu_2\rho_1}\eta^{\mu_3\nu_3}\eta^{\nu_1\nu_2} \,. \end{aligned} \qquad (3.2)$$

However, we also see that the final tensor expressions must be totally symmetric under the interchanges,

$$\mu_1 \leftrightarrow \mu_2 \leftrightarrow \mu_3 \,, \qquad \nu_1 \leftrightarrow \nu_2 \leftrightarrow \nu_3 \,, \qquad \rho_1 \leftrightarrow \rho_2 \,. \qquad (3.3)$$

The function GetTensorStructs handles this constraint. It first searches for a saved result, and if one cannot be found, generates and saves the appropriate tensors. For the example of Eq. (3.1), it returns six tensors, $T_i^{\mu_1\mu_2\mu_3|\nu_1\nu_2\nu_3|\rho_1\rho_2}$, that are sums of products of metric tensors that obey Eq. (3.3). For instance,

$$\begin{aligned} T_1^{\mu_1\mu_2\mu_3|\nu_1\nu_2\nu_3|\rho_1\rho_2} \equiv\ & \eta^{\mu_1\mu_2}\eta^{\mu_3\nu_1}\eta^{\nu_2\nu_3}\eta^{\rho_1\rho_2} + \eta^{\mu_1\mu_2}\eta^{\mu_3\nu_2}\eta^{\nu_1\nu_3}\eta^{\rho_1\rho_2} + \eta^{\mu_1\mu_2}\eta^{\mu_3\nu_3}\eta^{\nu_1\nu_2}\eta^{\rho_1\rho_2} \\ & + \eta^{\mu_1\mu_3}\eta^{\mu_2\nu_1}\eta^{\nu_2\nu_3}\eta^{\rho_1\rho_2} + \eta^{\mu_1\mu_3}\eta^{\mu_2\nu_2}\eta^{\nu_1\nu_3}\eta^{\rho_1\rho_2} + \eta^{\mu_1\mu_3}\eta^{\mu_2\nu_3}\eta^{\nu_1\nu_2}\eta^{\rho_1\rho_2} \\ & + \eta^{\mu_2\mu_3}\eta^{\mu_1\nu_1}\eta^{\nu_2\nu_3}\eta^{\rho_1\rho_2} + \eta^{\mu_2\mu_3}\eta^{\mu_1\nu_2}\eta^{\nu_1\nu_3}\eta^{\rho_1\rho_2} + \eta^{\mu_2\mu_3}\eta^{\mu_1\nu_3}\eta^{\nu_1\nu_3}\eta^{\rho_1\rho_2}\,. \end{aligned} \tag{3.4}$$

Notice that every summand of Eq. (3.4) contracts the tensor numerator into the same scalar product. So, we can think of $T_1^{\mu_1\mu_2\mu_3|\nu_1\nu_2\nu_3|\rho_1\rho_2}$ as the sum of all elements of an equivalence class that contracts $\ell_1^{\mu_1}\,\ell_1^{\mu_2}\,\ell_1^{\mu_3}\,\ell_2^{\nu_1}\,\ell_2^{\nu_2}\,\ell_2^{\nu_3}\,\ell_3^{\rho_1}\,\ell_3^{\rho_2}$ into $\ell_1^2\,(\ell_1\cdot\ell_2)\,\ell_2^2\,\ell_3^2$. In other words, for any representative of $T_1^{\mu_1\mu_2\mu_3|\nu_1\nu_2\nu_3|\rho_1\rho_2}$,

$$T_{1\ \mu_1\mu_2\mu_3|\nu_1\nu_2\nu_3|\rho_1\rho_2}\Big|_{\text{representative}}\ \ell_1^{\mu_1}\,\ell_1^{\mu_2}\,\ell_1^{\mu_3}\,\ell_2^{\nu_1}\,\ell_2^{\nu_2}\,\ell_2^{\nu_3}\,\ell_3^{\rho_1}\,\ell_3^{\rho_2}\ =\ \ell_1^2\,(\ell_1\cdot\ell_2)\,\ell_2^2\,\ell_3^2\,. \tag{3.5}$$

From the tensor structures $T_i^{\mu_1\mu_2\mu_3|\nu_1\nu_2\nu_3|\rho_1\rho_2}$, we can then find all possible scalar product structures (also shown in Eq. (1.3)),

$$\begin{aligned} & S_1 \equiv \ell_1^2\,(\ell_1\cdot\ell_2)\,\ell_2^2\,\ell_3^2\,, && S_2 \equiv (\ell_1\cdot\ell_2)\,(\ell_1\cdot\ell_3)^2\,\ell_2^2\,, && S_3 \equiv \ell_1^2\,(\ell_1\cdot\ell_3)\,\ell_2^2\,\ell_3^2\,, \\ & S_4 \equiv (\ell_1\cdot\ell_2)^3\,\ell_3^2\,, && S_5 \equiv (\ell_1\cdot\ell_2)^2\,(\ell_1\cdot\ell_3)\,(\ell_2\cdot\ell_3)\,, && S_6 \equiv \ell_1^2\,(\ell_1\cdot\ell_2)\,(\ell_2\cdot\ell_3)^2\,, \end{aligned} \tag{3.6}$$

where we defined,

$$S_i\ \equiv\ T_{i\ \mu_1\mu_2\mu_3|\nu_1\nu_2\nu_3|\rho_1\rho_2}\Big|_{\text{rep.}}\ \ell_1^{\mu_1}\,\ell_1^{\mu_2}\,\ell_1^{\mu_3}\,\ell_2^{\nu_1}\,\ell_2^{\nu_2}\,\ell_2^{\nu_3}\,\ell_3^{\rho_1}\,\ell_3^{\rho_2}\,. \tag{3.7}$$

The function GetScalarProducts returns all such scalar products, again saving the results for later retrieval.

Now that we have the tensor and scalar building blocks, which are respectively independent of loop momenta and solely dependent on loop momenta, we are prepared to construct an ansatz for the form of the tensor-reduced numerator:

$$\begin{aligned} \ell_1^{\mu_1}\,\ell_1^{\mu_2}\,\ell_1^{\mu_3}\,\ell_2^{\nu_1}\,\ell_2^{\nu_2}\,\ell_2^{\nu_3}\,\ell_3^{\rho_1}\,\ell_3^{\rho_2} \to\ & C_1^{\mu_1\mu_2\mu_3|\nu_1\nu_2\nu_3|\rho_1\rho_2}\,S_1 + C_2^{\mu_1\mu_2\mu_3|\nu_1\nu_2\nu_3|\rho_1\rho_2}\,S_2 \\ & + C_3^{\mu_1\mu_2\mu_3|\nu_1\nu_2\nu_3|\rho_1\rho_2}\,S_3 + C_4^{\mu_1\mu_2\mu_3|\nu_1\nu_2\nu_3|\rho_1\rho_2}\,S_4 \\ & + C_5^{\mu_1\mu_2\mu_3|\nu_1\nu_2\nu_3|\rho_1\rho_2}\,S_5 + C_6^{\mu_1\mu_2\mu_3|\nu_1\nu_2\nu_3|\rho_1\rho_2}\,S_6\,, \end{aligned} \tag{3.8}$$

where the tensor coefficients $C_i^{\mu_1\mu_2\mu_3|\nu_1\nu_2\nu_3|\rho_1\rho_2}$ are linear combinations of the building blocks $T_j^{\mu_1\mu_2\mu_3|\nu_1\nu_2\nu_3|\rho_1\rho_2}$. More specifically, we define $C_i^{\mu_1\mu_2\mu_3|\nu_1\nu_2\nu_3|\rho_1\rho_2}$ as,

$$C_i^{\mu_1\mu_2\mu_3|\nu_1\nu_2\nu_3|\rho_1\rho_2} \equiv \sum_{j=1}^{6} a_{ij}\,T_j^{\mu_1\mu_2\mu_3|\nu_1\nu_2\nu_3|\rho_1\rho_2}\,, \tag{3.9}$$

where a_{ij} can only depend on rational numbers and $D_s \equiv \eta^\alpha{}_\alpha$, the trace of the metric tensor. We can think of D_s as related to the dimension D, but it is important to leave this quantity symbolic because of dimensional regularization.

Be careful with the ansatz of Eq. (3.8). There are a lot of 6's floating around. Whereas T_i is directly related to S_i (see Eq. (3.7)), C_i is a linear combination of all T_j's. Our final, and most computationally-intensive, step it to find the exact relationship between C_i's and T_j's, i.e., to uniquely fix a_{ij} for all $i, j = 1, 2, \ldots, 6$. This requires us to construct and solve a system of 36 linearly-independent equations.

The system of equations can be constructed from Eq. (3.8) by contracting both sides by a representative from each T_i, noting the definition of S_i in Eq. (3.7). Thus, to match the RHS of Eq. (3.8) with the LHS, we demand that (denoting the lengthy index contraction by $\cdot$),

$$T_i|_{\text{rep.}} \cdot C_j = \delta_{ij} \,, \tag{3.10}$$

where δ_{ij} is the Kronecker delta. Substituting the definition of C_j from Eq. (3.9), this constraint reads,

$$\sum_{k=1}^{6} a_{jk} \left(T_i|_{\text{rep.}} \cdot T_k \right) = \delta_{ij} \,, \quad \forall\; i, j = 1, 2, \ldots, 6 \,. \tag{3.11}$$

Note that we could equivalently choose to contract with the full T_i tensor instead of a representative, but this would be unnecessary work and would require a symmetry factor. If we use one representative for T_i, then we must contract with all summands of T_k or vice versa. The tensor contraction of Eq. (3.11) is carried out by the function `GetContractions`, with the subroutine `ContractionCycles` performing the heavy lifting, as discussed in Section II. The function `GetEqs` fully assembles the equations. Finally, `GetSoln` solves the equations for a_{ij}, uniquely defining the tensor-reduction replacement given in Eq. (3.8). Because of the need to keep D_s symbolic in the equations instead of numeric, the equation solving can be cumbersome and appears to currently be the bottleneck at high loop orders.

After the tensor reduction, industrial-grade machinery such as `FIRE5` (arXiv:1408.2372) can be used to simplify the scalar integrals into a basis of integrals using integration-by-parts (IBP) relations.