title

Thomas Young

2020 年 7 月 30 日

COLLIER is a fortran 单圈--标量和张量的数值积分程序库。
这些积分出现在微扰的相对论性量子场论中。
它具有以下 features：

1. 多粒子复杂度 scalar and tensor integrals
2. ultraviolet divergences 的维数正规化
3. soft infrared divergences 的维数正规化（对于非阿贝尔场，也支持 mass regularization）
4. 对于共线质量奇点的维数正规化或者质量正规化
5. 对于不稳定粒子，complex 内线质量完全支持（外动量和 virtualities 认作是实数）
6. 数值危险区域（小 Gram 或者其他运动学行列式），使用专用的展开处理。
7. 所有基本模块都有两种平行的实现方式，可以用作内部交叉检验
8. 缓存系统--用来加速计算 \end{itemize}

代码提供了量子场论中任意张量和标量积分的数值结果。

1. 对于张量积分，不管协变分解中的系数还是张量元本身都将给出。
2. COLLIER 支持 complex 质量，在计算不稳定粒子时会需要。
3. 采用维数正规化处理紫外和红外奇点。
4. 对于 soft 和共线奇点，有可选用的质量正规化方案。

第一章　COLLIER doc

1.1　Convention

一致性地使用 Refs. [50, 59] 中的约定。约化张量积分的方法在 Refs. [43, 50] 中有描述，已经实现 4--点函数的结果可以在 Ref. [59] 中找到。标量 1-，2-，3-点函数的结果基于 Refs. [45, 52]

D 维空间中，单圈 N 点张量积分的一般形式为：

$$T^{N,\mu_1,\cdots,\mu_P}(p_1,\cdots,p_{N-1},m_0,\cdots,m_{N-1}) = \frac{(2\pi\mu)^{4-D}}{i\pi^2}\int \mathrm{d}^D q \frac{q^{\mu_1}\cdots q^{\mu_P}}{N_0 N_1 \cdots N_{N-1}} \tag{1.1}$$

其中分母因子 $N_k = (q+p_k)^2 - m_k^2 + i\epsilon$, $k = 0,\cdots,N-1, p_0 = 0$, 其中 $i\varepsilon$ 是无穷小的虚部。

对于 $P=0$，即分子上是 1，(1.1) 定义了 N-点标量积分 T_0^N。

按照 Ref.[52]，我们令 $T^1 = A, T^2 = B, T^3 = C, T^4 = D, T^5 = E, T^6 = F, \text{and } T^7 = G$。

为了能够简洁的写出张量分解。

我们使用大括号来表示对所有洛伦兹指标进行对称化操作。比如：

$$\{p\cdots p\}_{i_1\cdots i_P}^{\mu_1\cdots\mu_P} = p_{i_1}^{\mu_1}\cdots p_{i_P}^{\mu_P}\{gp\}_{i_1}^{\mu\nu\rho} = g^{\mu\nu}p_{i_1}^{\rho} + g^{\nu\rho}p_{i_1}^{\mu} + g^{\mu\rho}p_{i_1}^{\nu} \tag{1.2}$$

$$gg^{\mu\nu\rho\sigma} = g^{\mu\nu}g^{\rho\sigma} + g^{\mu\sigma}g^{\nu\rho} + g^{\mu\rho}g^{\nu\sigma} \tag{1.3}$$

这种分解是可以递归进行的。

$$\{p\cdots p\}^{\mu_1\cdots\mu_P}_{i_1\cdots i_P} = p^{\mu_1}_{i_1}\cdots p^{\mu_P}_{i_P} \tag{1.4}$$

$$\{\underbrace{g\cdots g}p\cdots p\}^{\mu_1\cdots\mu_P}_{i_{2n+1}\cdots i_P} = \frac{1}{n}\sum_{\substack{k,l=1\\k<l}}^{P} g^{\mu_k\mu_l}\{\underbrace{g\cdots g}p\cdots p\}^{\mu_1\cdots\mu_{k-1}\mu_{k+1}\cdots\mu_{l-1}\mu_{l+1}\cdots\mu_P}_{i_{2n+1}\cdots i_P} \tag{1.5}$$

我们把一般的张量积分约化到洛伦兹协变的结构，as

$$\begin{aligned}
T^{N,\mu 1,\cdots,\mu P} &= \sum_{n=0}^{[p/2]}\sum_{i_{2n+1},\cdots,i_p=1}^{N-1}\left\{\underbrace{g\cdots g}_{n}p\cdots p\right\}^{\mu 1\cdots\mu P}_{i_{2n+1}\cdots i_P} T^N_{\underbrace{0\cdots 0}_{2n}\,i_{2n+1}\cdots i_p}\\
&= \sum_{i_1,\cdots,i_p=1}^{N-1} p^{\mu_1}_{i_1}\cdots p^{\mu_P}_{i_P} T^N_{i_1\cdots i_P} + \sum_{i_3,\cdots,i_p=1}^{N-1}\{gp\cdots p\}^{\mu_1\cdots\mu_P}_{i_3\cdots i_P} T^N_{00 i_3\cdots i_P}\\
&+ \sum_{i_5,\cdots,i_p=1}^{N-1}\{ggp\cdots p\}^{\mu_1\cdots\mu_P}_{i_5\cdots i_P} T^N_{0000 i_5\cdots i_P} + \cdots\\
&+ \begin{cases}\sum_{i_P=1}^{N-1}\{g\cdots gp\}^{\mu_1\cdots\mu_P}_{i_P} T^N_{\underbrace{0\cdots 0}\,i_P}, & \text{for } P \text{ odd},\\ \{g\cdots g\}^{\mu_1\cdots\mu_P} T^N_{\underbrace{0\cdots 0}_{P}}, & \text{for } P \text{ even}\end{cases}
\end{aligned}$$

其中 $P/2$ 是小于等于 $P/2$ 的最大整数。对于洛伦兹协变结构的每一个度规张量，对应的系数携带一个 00 指标，对于每一个动量 p_{i_r}，系数携带一个指标 i_r。通过定义，张量系数 $T^N_{i_1\cdots i_P}$ 对于指标 $i_1,\cdots,i_P$ 是完全对称的。

UV-- or IR--singular 积分利用维数正规化来表示，其中 $D = 4-2\epsilon$，as,

$$T^N = \tilde{T}^N_{\rm fin} + a^{\rm UV}(\Delta_{\rm UV} + \ln\frac{\mu^2_{\rm UV}}{Q^2}) + a^{\rm IR}_2(\Delta^{(2)}_{\rm IR} \tag{1.6}$$

$$+ \Delta^{(1)}_{\rm IR}\ln\frac{\mu^2_{\rm IR}}{Q^2} + \frac{1}{2}\ln^2\frac{\mu^2_{\rm IR}}{Q^2}) + \tilde{a}^{\rm IR}_1(\Delta^{(1)}_{\rm IR} + \ln\frac{\mu^2_{\rm IR}}{Q^2}) \tag{1.7}$$

$$= T^N_{\rm fin}(\mu^2_{\rm UV},\mu^2_{\rm IR}) + a^{\rm UV}\Delta_{\rm UV} + a^{\rm IR}_2(\Delta^{(2)}_{\rm IR} + \Delta^{(1)}_{\rm IR}\ln\mu^2_{\rm IR}) + a^{\rm IR}_1\Delta^{(1)}_{\rm IR}] \tag{1.8}$$

其中

$$\Delta_{UV} = \frac{c(\epsilon_{UV})}{\epsilon_{UV}}, \qquad c(\epsilon) = \Gamma(1+\epsilon)(4\pi)^{\epsilon}$$
$$\Delta_{IR}^{(2)} = \frac{c(\epsilon_{IR})}{\epsilon_{IR}^2}, \qquad \Delta_{IR}^{(1)} = \frac{c(\epsilon_{IR})}{\epsilon_{IR}}$$

我们让所有的 UV 和 IR 极点清晰的展示出来，包括对应的质量能标 μ_{UV} 和 μ_{IR}。我们进一步提取出因子 $c(\epsilon) = \Gamma(1+\epsilon)(4\pi)^{\epsilon} = 1 + \mathcal{O}(\epsilon)$，

并把它吸收到 Δ_{UV},$\Delta_{IR}^{(2)}, \Delta_{IR}^{(1)}$ 的定义里。为了避免在 (1.8) 中第一个方程的对数里出现有量纲的量，我们分离出一个辅助的能标 Q，它隐式地由进入各个圈图的质量和动量决定。

COLLIER 的输出对应 (1.8) 的最后一行，包括正比于 $a^{UV}, a_2^{IR}, a_1^{IR}$ 的项。参量 $\mu_{UV}^2, \mu_{IR}^2, \Delta_{UV}, \Delta_{IR}^{(2)}, \text{and}, \Delta_{IR}^{(1)}$ 可以由用户自由选取，但不会影响 UV-和 IR-极限下有限的量。

注意我们区分 IR 和 UV 起源的奇点，默认 $a^{UV}, a_2^{IR}, a_1^{IR}$ 被设置为 0，输出就是 $T_{fin}^N\big(\mu_{UV}^2, \mu_{IR}^2\big)$。将 Δ's 设置成不同于 0 的数，可以数值的模拟极点 ϵ 的影响。

默认 IR-和 UV-奇点在维数正规化中计算。共线奇点也可以通过质量正规化。为了达到这个目的，相应的质量，下文称为 $\overline{m_i}$，必须在初始化中被声明为 *small*，此外，在后续子程序调用的时候，各质量参数必须和在初始化文件中是精确相同的（但不必要很小）。小质量在标量和张量函数中被当作无穷小量对待，其有限值只在质量-奇点的对数项中保留。

阿贝尔类型的软奇点，i.e. 当 $a_2^{IR} = 0$，和共线奇点，通过质量 $\overline{m_i}$ 被正规化。当参数 $\Delta_{IR}^{(1)}$ 被设置为 0 之后，参数 μ_{IR} 可以看作是无穷小的光子或胶子质量。

变动参数 $\mu_{UV}^2, \mu_{IR}^2, \Delta_{UV}, \Delta_{IR}^{(2)}, \text{and}, \Delta_{IR}^{(1)}$ 的值可以检查奇点的相消情况。此外，给 $\Delta_{UV}, \Delta_{IR}^{(2)}, \text{and}, \Delta_{IR}^{(1)}$ 选择适合的值，可以允许用户在不同的约定中转换，考虑到提取前置因子 $c(\epsilon)$ 的不同方式。例如, 在 [1] 中，相关的 ϵ 因子是 π^{ϵ}，其中 $r_\Gamma = \Gamma^2(1-\epsilon)\Gamma(1+\epsilon)/\Gamma(1-2\epsilon)$，而本文 (1.1) 中的约定为 $(2\pi)^{2\epsilon}$。

因此我们必须如下替换我们的 $c(\epsilon)$:

$$\frac{c(\epsilon)}{r_\Gamma(4\pi)^{\epsilon}} = \frac{\Gamma(1+\epsilon)}{r_\Gamma} = \frac{\Gamma(1-2\epsilon)}{\Gamma^2(1-\epsilon)} = 1 + \epsilon^2\frac{\pi^2}{6} + \mathcal{O}\left(\epsilon^3\right)$$

为了得到 [1] 中约定下的奇点积分。这等价于作替换 $\Delta_{IR}^{(2)} \to \Delta_{IR}^{(2)} + \pi^2/6$,

同时保持我们计算中的 $\Delta_{UV}\text{and}\Delta_{IR}^{(1)}$ 不变。在此变换之后，我们的参数 $\Delta_{UV}, \Delta_{IR}^{(2)}, \text{and}, \Delta_{IR}^{(1)}$ 就分别对应于文献 [1] 中的极点 $1/\epsilon, 1/\epsilon^2, \text{and} 1/\epsilon$。

1.1.1 总结

对于单圈 N 点张量积分：

$$T^{N,\mu_1,\cdots,\mu_P}(p_1,\cdots,p_{N-1},m_0,\cdots,m_{N-1}) = \frac{(2\pi\mu)^{4-D}}{i\pi^2}\int \mathrm{d}^D q \frac{q^{\mu_1}\cdots q^{\mu_P}}{N_0 N_1 \cdots N_{N-1}}$$

点的数目是 N，那么引入的外动量数目是 $N-1$，最后张量指标的数目也是 $N-1$。依照惯例，把分子上不含积分动量的积分称为标量积分 $A, B, C\cdots, G$,

一个张量积分可以约化成 coefficients and 张量结构的形式。coefficients 就是一些含有不同外动量参数的标量积分 $A, B, C\cdots, G$，张量结构由外动量的对称化和度规张量组成。张量结构中外动量的对称性，可以从积分表达式中看出来。

1.2 introduction

multi-leg one-loop amplitudes 振幅求值的巨大进步，来自于两方面：

1. 传统费曼图方法的系统改进
2. 基于推广的幺正性关系的新理论技术

在第二种方法中，单圈振幅被直接表示成标量积分的组合。这种向固定标量积分基的直接约化，会引发相空间特定区域的数值问题。一般可以通过采用四次精度的数值计算克服。

相反，费曼图方法，包括最近的递归方法依赖于张量积分。此方法允许分解方法自适应于相空间的不同区域，在相当大的程度上，通过最优选择避免数值不稳定性。COLLIER 库提供了计算标量和张量积分的全面工具。在两种互补的方法中都能应用。

将张量积分约化到一小族基本积分的方法，可以追溯到 Brown and Feynman, Melrose, and Passarino and Veltman。又经过了数十年的发展，文献 [43, 50] 展示的完整方法，是 COLLIER 代码的基础。作为张量分解基础的标量积分首次由 t' Hooft and Veltman 进行了系统研究。已经存在数个计算单圈标量和张量积分的库，比如：

- FF，LoopTools，
- QCDLoop，OneLoop，
- GoLem95C，PJFRY，Package-X

这里介绍的 COLLIER 库，提供完全的张量积分集合，处理带有复数质量的过程，并且没有先验的粒子数限制。

COLLIER 已在用于多个前沿课题的计算。

文章结构：

- Section 2：COLLIER 相关约定
- Section 3：计算张量积分的方法轮廓
- Section 4：COLLIER 库的内部结构
- Section 5：用法
- Section 6：总结
- Appendix A：定义单圈积分的运动学输入细节

1.3 实现方法

1.3.1 张量系数的计算

计算张量积分的方法依赖于它的传播子数目 N。对于 $N = 1, 2$，我们使用显式的数值稳定表达式 [2, 3]。

对于 $N = 3, 4$，所有张量积分被数值约化到基本的标量积分，通过使用文献 [4–6] 给出的解析表达式。默认情况下，约化使用的是标准的 Paassarino-Veltman [2] 约化。在相空间不稳定殿，其 Gram 行列式变得很小，Paassarino-Veltman 约化变得不稳定。这在些点上，我们使用专用的递归展开方法 [3]。所有这些方法都在 COLLIER 中得到实现，并且对于展开参数可以到任意阶。为了决定一个特定相空间点的约化方法，以下步骤被采用：

1. 默认是 Passarino-Veltman ，它的可靠性通过一个给标量积分预先分配的精度来估计，然后估计约化时的误差传递。如果积分最高

rank$\hat{P}$ 的最终的误差结果 $\Delta T^N\left(\hat{P}\right)$，小于一个预定义的精度标签 $\eta_{\rm req}$ (required precision)，结果被保留并传递给用户。

2. 若步骤 1 没有提供足够的精度，一般意味着张量积分包括外动量的小 Gram 行列式。COLLIER 切换到专用展开。为决定到哪一阶是足够的，一个先验的误差估计式 $\Delta T^N_{\rm prelim}(P)$，对于系数 $T^N_{i1,\cdots iP}$ 被构建，对于不同方法的最高 rank$\hat{P}$。误差估计基于展开的期望精度评定，和所需标量积分的简化的误差传递。$\Delta T^N_{\rm prelim}(P)$ 最小的展开方法被采用。在展开式的实际计算中，评估的是更加现实的精度 $\Delta T^N(P)$，通过分析最后一次迭代的修正。如果预定义的精度标签 $\eta_{\rm req}$ 达到，结果被保存并返回给用户。否则在达到一个预定义的展开深度时，展开停止，或者从一次迭代到下一次，精度没有增加。

3. 如果步骤 2 没有提供所需精度。将对其他方法进行重复，这些方法对于足够小的 $\Delta T^N_{\rm prelim}(P)$，将能够保证收敛。如果经过这些重复任一个，达到预定义精度，结果被保留并返回给用户。

4. 如果 Paassarino-Veltman 和其他展开方法都不能达到目标精度，具有最小误差估计的 $\Delta T^N(P)$ 的方法，其结果将被返回给用户。

通过这种方法，对于几乎所有相空间的点，都能得到稳定的结果。保证了可靠的 Monte Carlo 积分。

对于 $N=5,6$，张量积分被约化到具有更低 rank 和 N 的积分，遵循 [3,7]，i.e. 不涉及到 Gram 行列式的逆。对于 $N\geq 7$，文献 [3]Section 7 中 6–点张量积分约化的修改版被应用（见 (7.10) 之后的文字）。

1.3.2 全张量的计算

目前为止，文中描述的方法是用洛伦兹不变的项进行约化，得到的系数为 $T^N_{i1,\cdots,iP}$。新一代的 NLO 生成器比如 OPENLOOPS, RECOLA，需要全张量 $T^{N,\mu 1,\cdots,\mu P}$ 的分量。为了达到这个目的，COLLIER 中实现了一套高效的算法，来从 $T^N_{i1,\cdots,iP}$ 构建 $T^{N,\mu 1,\cdots,\mu P}$。它递归地计算 4 中那些单举动量构造的张量结构。张量结构的非零分量包括度规矩阵递归地得到，通过添加成对相等的洛伦兹指标，到带有更少度规的张量上，考虑到指标组合因子和度规张量引起的符号。相关的组合系数在 COLLIER 初始化的时候计算并列表。

coefficients for	$\widehat{P}=0$	$\widehat{P}=1$	$\widehat{P}=2$	$\widehat{P}=3$	$\widehat{P}=4$	$\widehat{P}=5$	$\widehat{P}=6$
$N=3$	1	3	7	13	22	34	50
$N=4$	1	4	11	24	46	80	130
$N=5$	1	5	16	40	86	166	296
$N=6$	1	6	22	62	148	314	610
$N=7$	1	7	29	91	239	553	1163
components	1	5	15	35	70	126	210

Table 1: Number $n_c(N,\widehat{P})$ of invariant coefficients $T^N_{i_1\dots i_P}$ for $N=3,\dots,7$ and rank $P\le\widehat{P}=0,\dots,6$ (rows 2–6) and number $n_t(\widehat{P})$ of independent tensor components $T^{N,\mu_1\cdots\mu_P}$ for rank $P\le\widehat{P}=0,\dots,6$ (last row).

$$B_{i_1\dots i_P} \qquad C_{i_1\dots i_P} \qquad D_{i_1\dots i_P} \qquad E_{i_1\dots i_P} \qquad F_{i_1\dots i_P} \qquad G_{i_1\dots i_P} \qquad \dots$$

$$B^{\mu_1\cdots\mu_P} \qquad C^{\mu_1\cdots\mu_P} \qquad D^{\mu_1\cdots\mu_P} \qquad E^{\mu_1\cdots\mu_P} \qquad F^{\mu_1\cdots\mu_P} \qquad G^{\mu_1\cdots\mu_P} \qquad \dots$$

Figure 1: Reduction chains in COLLIER: For $N\ge 6$ reduction can be performed at the tensor level.

图 1.1: COLLIER 约化链：对于 $N\ge 6$，约化可以在张量层次进行

洛伦兹不变的系数 $T^N_{i1,\cdots,iP}$ 和 $T^{N,\mu1,\cdots,\mu P}$ 被列在 fig.1.1中.

对于 $N\le 4$，不变系数的数目小于张量分量的数目，这也是 Passarino-Veltman 约化方法的前提。另一方面，对于 $N\ge 5$，情况发生反转。实际上，[3] 中 (7.7) 中对于 $N\ge 6$ 所展现的方法，约化是用全张量的项推导的。若要得到张量系数，需要进行对陈化操作，得到的系数也不是唯一的，由于张量结构具有的冗余性。所以，对于 $N\ge6$ 的张量 $T^{N,\mu1,\cdots,\mu P}$ 的约化，COLLIER 直接在张量层次实现，而不用依赖于协变分解。

而当 $N\le 5$，迭代在 coefficient 的层次 exclusively 进行，$T^{N,\mu1,\cdots,\mu P}$ 随后才构建，由各自的 coefficients $T^N_{i1,\cdots,iP}$ ，对于 $N\ge6$，约化也可以在 tensors 层次完成。意思是，若要计算一个 $N\ge6$ 的张量积分，可以选取一个 $5<N_{\text{tenred}}\le N$ 的 N_{tenred}，对于 $N<N_{tenred}$，进行 coefficients 层次的递归运算，对于 $N\ge N_{tenred}$，在 tensor 水平进行计算。从 coefficients 到 tensor 的转变可以发生在 $N_{\text{tenred}}-1$。可能的约化链示于图1.1。

1.4 库的结构

COLLIER 的结构在图 2 中图形化的展示出来。库的核心由模块 COLI 和 DD 组成。它们是标量积分 T_0^N 和洛伦兹不变系数 $T^N_{i_1\cdots i_P}$ 的两种独立的实现，利用前文描述的方案。模块 *tensors* 提供了从 $T^N_{i_1\cdots i_P}$ 构建 $T^{N,\mu_1\cdots\mu_P}$ 的路径，同时也包括 $N \geq 6$ 的时候 N 点积分，张量层次的直接约化。用户通过 COLLIER 的全局界面和 COLI，DD,and tensors 的常规流程交互。它提供了路径，去设置或提取 COLI 和 DD 的参数，还有计算张量系数 $T^N_{i_1\cdots i_P}$ 和张量元 $T^{N,\mu_1\cdots\mu_P}$ 的路径。用户可以选择使用那个分支–COLI 或 DD。也可以用两个分支计算每个积分，来做结果的交叉检验。

在计算一个典型的单圈矩阵元时，一个张量积分会被调用好几次，并输入相同的运动学参数：另一方面，在计算 $P \geq 2$ 的 N 点积分时，会引起对更低阶 $N\prime$ 积分的递归调用。在约化树中，对于 $N\prime \leq N-2$ 阶的积分，有不同的抵达路径。为了避免对同一个积分进行重复运算，COLLIER 的子库连接到了一个 Global 的 cache 系统，其工作原理如下：

参数 N_{ext} 计数外部程序的积分调用次数，在约化过程中，内部调用被一个二进制标识符 id 记录。对于每一个索引对 $\left(N_{\text{ext},}id\right)$ 分配一个指针。对于第一个相空间点的计算，相应函数的参数被比较，具有相同参数的索引对 $\left(N_{\text{ext},}id\right)$ 被指向 cache 中的相同地址。第一次计算的结果被写入 cache，后续相空间点的计算可以读取这些结果，如果指向相同的地址。使用 external cache 系统是可选择的，在一个 Monte Carlo 积分中，对张量积分的调用，需要相空间所有点的次序是 exactly 相同的，在初始化之后（初始化标志着矩阵元计算的开始，对于各个相空间点）。此外，对于每个事件，第一次和最后一次调用积分的内部参数必须保持不变。

1.5 库的使用

1.5.1 安装

下载包 COLLIER-v.tar.gz collier homepage。其中 v 是库的版本。还应该安装 CMAKE 创建系统。由于 COLLIER 是一个单机的 Fortran95 代码，无需额外的库。

gunzip and untar COLLIER-v.tar.gz 将会解压到./COLLIER-v 文件夹，包含以下文件和文件夹

1. Cmakelists.txt ： cmake makefile 用来产生 COLLIER 库。
2. src：COLLIER 源代码库，包含 COLLIER 的主要代码和子库的主要代码。
   - COLI：包含 COLI 分支的文件
   - DDlib：包含 DD 分支的文件
   - tensors：包含构建张量和直接张量约化的文件
   - Aux：包含辅助文件。
3. build：build 文件夹，CMAKE 存放所有创建用必须文件的地方，比如对象文件。
4. modules：空文件夹，for fortran 模块文件。
5. demos：展示 COLLIER 用法的示例文件夹。
6. COPYING：版权信息文件。

使用以下命令创建 COLLIER 库：

```
cd build
cmake
make
```

默认 cmake 会建立一个动态库。如果需要静态库，在 COLLIER-v 文件夹中，使用以下选项

```
cmake -Dstatic=ON ..
```

如果不指定编译器，cmake 会自动寻找安装的 fortran 编译器，选择合适的。使用特定的编译器比如 ifort，可以用以下选项

```
cmake -DCMAKE_Fortran_COMPILER=ifort ..
```

可以给出编译器的全路径。

makefile 创建以后，make 命令就会产生动态库 libCOLLIER.so 或者静态库 libCOLLIER.a 在 COLLIER-v 文件夹中，可以用来链接到用户程序。

若要创建 demos 文件夹内示例程序的可执行文件，在文件夹 COLLIER-v/build 内使用

```
make demo
make democache
```

所有使用 make 命令创建的文件可以用 “make clean’’ 来丢弃,在 COLLIER-v/build 中来运行。

也可以删除 COLLIER-v/build 中的所有文件。

Sample programs

在 COLLIER-v/build 文件夹中使用以下命令创建两个示例程序

```
make demo
make democache
```

可以在 COLLIER-v/demo 文件夹使用以下命令运行

```
./demo
./democache
```

程序 demo 专门用来计算 single 张量积分。

在运行过程中，会询问用户在 N 点-积分中选择一个例子来计算。计算结果被写入 demo_Npoint_exampleX.dat 中，它指向 demo.f90 中的段落，其中给出了计算各个积分的源代码，接着一小段程序，用来给 COLLIER 输出化，对于所有示例程序都差不多。其中包含了很多被注释的行，去掉感叹号就会起作用，其中展示了很多 COLLIER 的 global 参数，可以修改使用。

程序 decmocache 展示了 cache 的用法。对于 1000 个相空间点，8 个张量积分被计算了数次。这个玩具 monte carlo 在四种子集中相继执行: 先用 COLI，用或不用缓存，再用 DD，用或不用缓存。源代码存储在文件 democache.f90 中。

1.5.2 概括使用说明

为了在 FORTRAN 程序中使用 COLLIER，COLLIER-ν/modules 中的对应模块必须被载入:

```
use COLLIER
```

COLLIER-ν 中的 *library libCOLLIER.so* or *libCOLLIER.a* 必须提供给 LINKER。这样程序才有权访问 COLLIER 的公共函数和子程序。所有的子程序都带有后缀”_cll’’。为了避免冲突，也为了增加可读性。

在使用 COLLIER 之前，必须进行初始化,Calling

```
subroutine Init_cll(Nmax,rin,folder_name,noreset)
integer Nmax :maximal # of loop propagators
integer, optional rin :maximal rank of loop integrals
character, optional folder_name : name of folder for output
logical, optional noreset : no new output folder and files
```

`Nmax` 是强制性的，另外两个参数 `rin`,`folder_name`, and `noreset` 是可选的。利用 Nmax 指定所需计算张量圈积分 $T^{N,P}$ 的最大传播子数目。$(N < N_{max})$，可选参数 rin 制定了最高阶 $P_{max}(p \leq P_{max})$。如果参数 rin 被忽略，那么默认将 rin 设置为 Nmax，对于可重整化理论足够。Nmax and rin 决定了 COLLIER 内部产生的表格的大小。

folder_name 参数指定特定名称的输出文件夹。缺省值是'output_cll'。也可以传递给 init_cll 一个空字符串 foldername=", 这样会阻止创建输出文件夹，除了初始化信息和重要错误会被写入到标准输出通道 stdout_cll=6.

在后续的调用和计算中，如果 noreset 被设置为.true.，那么输出文件夹不会被重新创建，但是文件会被覆盖。在第一次 call init_cll 的时候，flag noreset 会被忽略。

call init_cll 会将所有内部参数设置为 Table.1.2 中的值。在后续的调用中，如果 noreset 设置为.true.，那么自行设定的参数值不会被重置为这里的初始化值。

在初始化之后，很多参数可以被设置得不同，满足用户需要。为实现这个功能，COLLIER 提供了 subroutine SetX_cll 对每个参数 X，subroutines SwitchOnY_cll, SwithchY_cll，对于每个 flag Y。To read out the current value of parameter X，a subroutine getX_cll is available. 这些参数可以被用户修改，subroutine 在 section5.4 进了详细描述。

在初始化之后和可能潜在的重新定义之后，COLLIER 就可以计算张量积分了。

通用 subroutine TN_cll 计算洛伦兹协变分解中的张量系数 $T^N_{i1,\cdots,iP}$

TNten_cll 返回张量元 $T^{N,\mu 1\cdots\mu p}$。

此外也提供可选的特定 subroutine A_cll, B_cll,...,\G_cll, and Aten_cll, Bten_cll,..., Gten_cll for the 1-,2-,...,7-point 积分，同样也有 A0_cll, B0_cll, ..., D0_cll 对于标量积分。

两点函数的动量导数，通常需要用来计算重整化常数，可以使用通用

parameter	type	set with	default
mode	integer $\in \{1,2,3\}$	SetMode_cll	1
$\eta_{\rm req}$	double precision	SetReqAcc_cll	1d-8
$\eta_{\rm crit}$	double precision	SetCritAcc_cll	1d-1
$\eta_{\rm check}$	double precision	SetCheckAcc_cll	1d-2
$\mu^2_{\rm UV}$	double precision	SetMuUV2_cll	1d0
$\mu^2_{\rm IR}$	double precision	SetMuIR2_cll	1d0
$\Delta_{\rm UV}$	double precision	SetDeltaUV_cll	0d0
$\Delta^{(1)}_{\rm IR}, \Delta^{(2)}_{\rm IR}$	double precision	SetDeltaIR_cll	0d0,0d0
$\{\overline{m}^2_1, \ldots, \overline{m}^2_{n_{\rm reg}}\}$	double complex ($n_{\rm reg}$)	SetMinf2_cll	{}
$\sigma_{\rm stop}$	integer < 0	SetErrStop_cll	-8
$N_{\rm tenred}$	integer ≥ 6	SetTenRed_cll	6
$n_{\rm cache}$	integer ≥ 0	InitCacheSystem_cll	0
$N^{\rm max}_{\rm cache}$	integer ($n_{\rm cache}$) ≥ 1	SetCacheLevel_cll	–
$n_{\rm err}$	integer ≥ 0	SetMaxErrOut_cll	100
$n_{\rm err,COLI}$	integer ≥ 0	SetMaxErrOutCOLI_cll	100
$n_{\rm err,DD}$	integer ≥ 0	SetMaxErrOutDD_cll	100
$n_{\rm inf}$	integer ≥ 0	SetMaxInfOut_cll	1000
$n^{N,\rm max}_{\rm check}$	integer ($N_{\rm max}$) ≥ 0	SetMaxCheck_cll	$\{50, ..., 50\}$
$n^{B',\rm max}_{\rm check}$	integer ≥ 0	SetMaxCheckDB_cll	50
$n^{N,\rm max}_{\rm crit}$	integer ($N_{\rm max}$) ≥ 0	SetMaxCrit_cll	$\{50, ..., 50\}$
$n^{B',\rm max}_{\rm crit}$	integer ≥ 0	SetMaxCritDB_cll	50
$\widehat{P}^{\rm max}$	integer ≥ 6	SetRitmax_cll	14
outlev	integer $\in \{0,1,2\}$	SetInfOutLev_cll	2

图 1.2: Table 2: lists of COLLIER parameters

subroutine DB_cll 计算到任意阶。对于最低阶，可以使用特定 subroutine, DB0_cll, DB1_cll, DB00_cll, and DB11_cll。更多介绍张量积分计算 subroutine 的信息在 section5.3 中给出。

COLLIER 的一个典型应用是在一个 NLO Monte Carlo 生成器中，提供单圈张量积分。在这种情况下，主程序对 MC 事件进行一个循环，对于每个事件，主程序调用 COLLIER 计算一组张量积分来得出矩阵元。在这种情形下中，the subroutine

```
subroutine InitEvent_cll(cacheNr)
integer, optional cacheNr: number of cacher
```

应该在每次计算张量积分之前被调用。这个调用将初始化 error flag and accuracy flag of COLLIER，这些 flag 可以在积分计算完成后，读取用来得到计算 status 的整体信息。如果使用了 cache system，call of InitEvent_cll 将是必须的，以用来初始化每次 MC 事件的 cache。如果使用了 multiple caches，那么各自的 cache number `cacheNr` 需要传递给 `InitEvent_cll`，作为一个可选参数。更多关于使用缓存的信息，可以在 section5.5 中找到。

为了帮助用户熟悉 COLLIER 的使用，两个示例程序 demo 和 democache 一并包含在发行版中。在 section 5.7 中可以找到描述。

1.5.3 张量积分的计算

对于张量积分，COLLIER 提供了 subroutine，传递洛伦兹协变分解系数 $T^N_{i1,\cdots,ip}$，和张量元 $T^{N,\mu_1,\cdots,\mu_P}$。

$$T^{N,\mu_1,\cdots,\mu_P}(p_1,\cdots,p_{N-1},m_0,\cdots,m_{N-1}) = \frac{(2\pi\mu)^{4-D}}{i\pi^2}\int \mathrm{d}^D q \frac{q^{\mu_1}\cdots q^{\mu_P}}{N_0 N_1 \cdots N_{N-1}} \tag{1.9}$$

积分中的外部变量共 $N-1$ 个，分子上的带指标被积动量共有 p 个。所以出来的洛伦兹结构中，也是有 p 个指标，

构成这 p 个指标的材料为，度规张量和 $N-1$ 个外动量，从中挑选出 p 个，所以外动量的循环指标是 $1 \sim N-1$.

其中分母因子 $N_k = (q+p_k)^2 - m_k^2 + i\epsilon$, $k = 0,\cdots,N-1, p_0 = 0$, 其中 $i\varepsilon$ 是无穷小的虚部。

$$T^{N,\mu 1,\cdots,\mu P} = \sum_{i_1,\cdots,i_p=1}^{N-1} p_{i_1}^{\mu_1}\cdots p_{i_P}^{\mu_P} T^N_{i_1\cdots i_P} + \sum_{i_3,\cdots,i_p=1}^{N-1} \{gp\cdots p\}^{\mu_1\cdots\mu_P}_{i_3\cdots i_P} T^N_{00i_3\cdots i_P} + \sum_{i_5,\cdots,i_p=1}^{N-1} \{ggp\cdots p\}^{\mu_1\cdots\mu_P}_{i_5\cdots i_P} T^N_{0000i_5\cdots i_P} + \cdots$$

张量系数 $T^N_{i1,\cdots,ip}$ 表示成一个 N–维数组，type double complex，并按照如下的约定：

$$TN\,(n_0, n_1, \cdots, n_{N-1}) = T^N_{\underbrace{0\cdots0}_{2n0}\underbrace{1\cdots1}_{n1}\underbrace{2\cdots2}_{n2}\underbrace{N-1\cdots N-1}_{nN-1}}$$

利用这种方法，所有张量系数，$T^N_{i1\cdots iP}$，其中 $P = 0, \cdots \hat{P}$, up to a given rank $\hat{P}$, 可以储存进同一个数组中

double complex TN$(0 : [\hat{P}/2], \underbrace{0 : \hat{P}, \cdots, 0 : \hat{P}}_{N-1})$

注意到相同的系数 $T^N_{i1\cdots ip}$，通过一个指标的置换 $\{i1, \cdots, iP\}$ 相互关联，在 `TN` 中也被表示成同一个 entry。作为例子，张量系数 $D_{i1..iP}$ 和数组 `D` 的对应关系，展现在 Table.1.3 中。这是 4 点函数到阶数 4 的情形。

圈积分 D 有 4 个传播子，3 个外动量，所以允许的最高阶指标数目为 4，

左边一列的下标中的数字，指的是传播子中外动量的序号，在结果中出现。重复的表示重复出现。

右边一列的四个位置，相当于四个传播子，其中的数字，是每个外动量，在最终结果中出现的次数。

它们是一一对应的。

可选的，N-点积分的张量系数 up to rank $\hat{P}$，可以通过一维数组得到

$$\text{double complex TN1}\left(\eta_c\left(N, \hat{P}\right)\right)$$

其中 $\eta_c\left(N, \hat{P}\right)$ 是张量系数 $T^N_{i1\cdots iP}$ 的总数目，其中 $i1 \le i2 \le \cdots \le iP$ and $P \le \hat{P}$。对于 $N = 1, \cdots, 7$ and $\hat{P} = 0, \cdots 6$,$\eta_c\left(N, \hat{P}\right)$ 的具体数值在 table.1.4 中给出。张量系数在数组 `TN1` 中按照升序排列，从 $P = 0$ 到 $P = \hat{P}$。相同 rank 的系数 $T^N_{i1\cdot iP}$ 和 $T^N_{j1\cdots jP}$ 按照他们的第一个相异的指标

D_0	D(0,0,0,0)	D_{113}	D(0,2,0,1)	D_{1112}	D(0,3,1,0)
D_1	D(0,1,0,0)	D_{122}	D(0,1,2,0)	D_{1113}	D(0,3,0,1)
D_2	D(0,0,1,0)	D_{123}	D(0,1,1,1)	D_{1122}	D(0,2,2,0)
D_3	D(0,0,0,1)	D_{133}	D(0,1,0,2)	D_{1123}	D(0,2,1,1)
D_{00}	D(1,0,0,0)	D_{222}	D(0,0,3,0)	D_{1133}	D(0,2,0,2)
D_{11}	D(0,2,0,0)	D_{223}	D(0,0,2,1)	D_{1222}	D(0,1,3,0)
D_{12}	D(0,1,1,0)	D_{233}	D(0,0,1,2)	D_{1223}	D(0,1,2,1)
D_{13}	D(0,1,0,1)	D_{333}	D(0,0,0,3)	D_{1233}	D(0,1,1,2)
D_{22}	D(0,0,2,0)	D_{0000}	D(2,0,0,0)	D_{1333}	D(0,1,0,3)
D_{23}	D(0,0,1,1)	D_{0011}	D(1,2,0,0)	D_{2222}	D(0,0,4,0)
D_{33}	D(0,0,0,2)	D_{0012}	D(1,1,1,0)	D_{2223}	D(0,0,3,1)
D_{001}	D(1,1,0,0)	D_{0013}	D(1,1,0,1)	D_{2233}	D(0,0,2,2)
D_{002}	D(1,0,1,0)	D_{0022}	D(1,0,2,0)	D_{2333}	D(0,0,1,3)
D_{003}	D(1,0,0,1)	D_{0023}	D(1,0,1,1)	D_{3333}	D(0,0,0,4)
D_{111}	D(0,3,0,0)	D_{0033}	D(1,0,0,2)		
D_{112}	D(0,2,1,0)	D_{1111}	D(0,4,0,0)		

Table 3: Mapping between tensor coefficients $D_{i_1 \ldots i_P}$ ($P \leq 4$) and elements $\mathtt{D}(n_0, n_1, n_2, n_3)$ of the array $\mathtt{D}(0\!:\!2, 0\!:\!4, 0\!:\!4, 0\!:\!4)$. The mapping onto the elements of the one-dimensional array representation $\mathtt{D1}(46)$ is obtained by numerating the coefficients in the table starting from the top left entry downwards.

图 1.3: table 3

coefficients for	$\widehat{P}=0$	$\widehat{P}=1$	$\widehat{P}=2$	$\widehat{P}=3$	$\widehat{P}=4$	$\widehat{P}=5$	$\widehat{P}=6$
$N=3$	1	3	7	13	22	34	50
$N=4$	1	4	11	24	46	80	130
$N=5$	1	5	16	40	86	166	296
$N=6$	1	6	22	62	148	314	610
$N=7$	1	7	29	91	239	553	1163
components	1	5	15	35	70	126	210

Table 1: Number $n_c(N,\widehat{P})$ of invariant coefficients $T^N_{i_1\dots i_P}$ for $N=3,\dots,7$ and rank $P\le\widehat{P}=0,\dots,6$ (rows 2–6) and number $n_t(\widehat{P})$ of independent tensor components $T^{N,\mu_1\cdots\mu_P}$ for rank $P\le\widehat{P}=0,\dots,6$ (last row).

图 1.4: table 1

i_k, j_k 进行排列。对于 4 点函数至 rank4，排序表见于1.3。由于 FORTRAN 数组只支持到 7 维，所以，对于 $N\ge 8$ 的 N 点积分，只能表示成一维数组的格式。

全张量积分 $T^{N,\mu_1,\cdots,\mu_P}$ 通过 type double complex 的 4-维数组表示，并按照以下约定

$$\text{TNten}\,(n_0,n_1,n_2,n_3)=T^{N,\overbrace{0\cdots0}^{n_0}\overbrace{1\cdots1}^{n_1}\overbrace{2\cdots2}^{n_2}\overbrace{3\cdots3}^{n_3}}$$

按照这种方法，所有张量元 $T^{N,\mu_1,\cdots,\mu_P}$，其中 $P=0,\dots,\hat{P}$ 至一给定 rank $\hat{P}$，被存储在相同的数组中

$$\text{double complex TNten}\left(0:\hat{P},0:\hat{P},0:\hat{P},0:\hat{P}\right)$$

注意，全同的张量元 $T^{N,\mu_1,\cdots,\mu_P}$，通过一个指标的置换 被 $\{\mu_1,\dots,\mu_p\}$ 彼此联系的，在数组 TNten 中用同一个 entry 表示。张量元 $T^{N,\mu_1,\cdots,\mu_P}$ 和数组 TNten 间的对应，在 table1.5中展示，至 rank $\hat{P}=3$。

洛伦兹协变分解1.9中的系数 $T^N_{i1}\cdot_{iP}$，来自于张量积分 $T^{N,P}$，其中 $N=1,\cdots,7$ 可以通过下列 subroutine 分别计算：A_cll,...,G_cll. subroutine N_cll(N_cll=A_cll,...,G_cll) 的参数结构如下给出：

subroutine N_cll(TN,TNuv ,MomInv,mass2,R,TNerr)

T_0	TNten(0,0,0,0)	T^{22}	TNten(0,0,2,0)	T^{033}	TNten(1,0,0,2)
T^0	TNten(1,0,0,0)	T^{23}	TNten(0,0,1,1)	T^{111}	TNten(0,3,0,0)
T^1	TNten(0,1,0,0)	T^{33}	TNten(0,0,0,2)	T^{112}	TNten(0,2,1,0)
T^2	TNten(0,0,1,0)	T^{000}	TNten(3,0,0,0)	T^{113}	TNten(0,2,0,1)
T^3	TNten(0,0,0,1)	T^{001}	TNten(2,1,0,0)	T^{122}	TNten(0,1,2,0)
T^{00}	TNten(2,0,0,0)	T^{002}	TNten(2,0,1,0)	T^{123}	TNten(0,1,1,1)
T^{01}	TNten(1,1,0,0)	T^{003}	TNten(2,0,0,1)	T^{133}	TNten(0,1,0,2)
T^{02}	TNten(1,0,1,0)	T^{011}	TNten(1,2,0,0)	T^{222}	TNten(0,0,3,0)
T^{03}	TNten(1,0,0,1)	T^{012}	TNten(1,1,1,0)	T^{223}	TNten(0,0,2,1)
T^{11}	TNten(0,2,0,0)	T^{013}	TNten(1,1,0,1)	T^{233}	TNten(0,0,1,2)
T^{12}	TNten(0,1,1,0)	T^{022}	TNten(1,0,2,0)	T_{333}	TNten(0,0,0,3)
T^{13}	TNten(0,1,0,1)	T^{023}	TNten(1,0,1,1)		

Table 4: Mapping between tensor components $T^{\mu_1 \cdots \mu_P}$ $(P \leq 3)$ and elements $\texttt{Tten}(n_0, n_1, n_2, n_3)$ of the array $\texttt{TNten}(0:3, 0:3, 0:3, 0:3)$. The mapping onto the elements of the one-dimensional array representation $\texttt{TNten1}(35)$ is obtained by numerating the coefficients in the table starting from the top left entry downwards.

图 1.5: table 4

double complex(0:R/2,$\underbrace{0:R,\ldots,0:R}_{N-1}$) TN : $T^{N,P}_{i_1,\ldots,i_P}$with $P \leq R$

double complex(0:R/2,$\underbrace{0:R,\ldots,0:R}_{N-1}$) TNuv : $T^{N,P\,\mathrm{UV}}_{i_1,\ldots,i_P}$with $P \leq R$

double complex(1:$n_{\mathcal{P}}$) MomInv : momentum invariants

double complex(0:$N-1$) mass2 : squared masses

integer R : maximal rank

double precision(0:R) optional TNerr : error estimates

一共有 $n_{\mathcal{P}} = \begin{pmatrix} N \\ 2 \end{pmatrix} = \frac{N(N-1)}{2} = \frac{(N-1)(N-2)}{2} + \frac{2(N-1)}{2}$ 个动量组成的不变量 $\mathcal{P}_N$，用符号 MomInv 表示，按照如下顺序排列：头 N 个对应 N 个 incoming 动量 k_i，后面 N 个是毗连动量的和的平方 $(k_i + k_{i+1})^2$，如此等等。如果用 off-set（偏移）动量 $p_i = k_1 + \ldots + k_i$（在1.9中的传播子中出现）来写的话，将会是

$$\begin{aligned}\mathcal{P}_{2k} = &\left(p_1-p_0\right)^2, \left(p_2-p_1\right)^2, \ldots, \left(p_{2k-1}-p_{2k-2}\right)^2, \left(p_0-p_{2k-1}\right)^2 \\ &\left(p_2-p_0\right)^2, \left(p_3-p_1\right)^2, \ldots, \left(p_0-p_{2k-2}\right)^2, \left(p_1-p_{2k-1}\right)^2, \\ &\ldots \\ &\left(p_{k-1}-p_0\right)^2, \left(p_k-p_1\right)^2, \ldots, \left(p_{k-3}-p_{2k-2}\right)^2, \left(p_{k-2}-p_{2k-1}\right)^2, \\ &\left(p_k-p_0\right)^2, \left(p_{k+1}-p_1\right)^2, \ldots, \left(p_{2k-2}-p_{k-2}\right)^2, \left(p_{2k-1}-p_{k-1}\right)^2, \quad (1.10)\end{aligned}$$

$$\begin{aligned}\mathcal{P}_{2k+1} = &\left\{\left(p_1-p_0\right)^2, \left(p_2-p_1\right)^2, \ldots, \left(p_{2k}-p_{2k-1}\right)^2, \left(p_0-p_{2k}\right)^2\right\} \\ &\left(p_2-p_0\right)^2, \left(p_3-p_1\right)^2, \ldots, \left(p_0-p_{2k-1}\right)^2, \left(p_1-p_{2k}\right)^2, \\ &\ldots \\ &\left(p_k-p_0\right)^2, \left(p_{k+1}-p_1\right)^2, \ldots, \left(p_{k-2}-p_{2k-1}\right)^2, \left(p_{k-1}-p_{2k}\right)^2, \quad (1.11)\end{aligned}$$

注意1.10中的前 $k-1$ 行每行有 $N=2$ 个元素，第 k 行只有 $k = N/2$ 个元素。而1.11中 k 行中每行都有 $N = 2k+1$ 个元素。

由于存在一个 overall 的动量守恒，N 个 incoming ki, 但是独立的偏移量 p_i 只有 $N-1$ 个。

可以想象一个圆周，圆周上等距离的画着 N 个点，相当于两点之间连线，然后开始转动，要求两端的点不能重复。如果连线是一条直径，那么只

有一半是不重复的。但是只有当圆周上的点是 $2k$ 的时候，才可能实现这种情况。

对于 $N = 2, \ldots, 7$，不变动量 $\mathcal{P}_N$ 的集合列在 Appendix A 中。它们必须以 type double complex 提供给 subroutine N_cll。或者是长度为 $n_{\mathcal{P}}$ 的数组，或者是 $n_{\mathcal{P}}$ 个 single 参数。值得注意的是，尽管变量类型是 double complex, 现版本的 COLLIER 还不支持动量不变量有虚部的情况。从长远看更重要的是，保证动量不变量的组合（对应单个外线粒子的质量平方）采用它们的精确数值，以避免任何偏差，比如对这些动量平方进行数值计算时。此外，在 IR-divergent 积分中，程序内部将会比较动量和质量参数来决定采用的解析表达式。显然，在单点积分 A_cll 中，参数 MomInv 将会省略。

squared masses 集合

$$\mathcal{M}_N = \left\{m_0^2, m_1^2, \ldots, m_{N-1}^2\right\} \tag{1.12}$$

进入到圈传播子中，在1.9给出，由 N 个 type double complex 的参数表示，记作 mass2。这些参数可以有不为零（负的）虚部，应当传递给 N_cll，格式为单个数组或者独立的参数，取决于动量不变量 MomInv 采用的格式。

integer 参数 R 表示张量积分的最高 rank $\hat{P}$。因此它定义了输出数组 TN and TNuv 的 size（type complex）。像之前描述的，它们可以是 N-维数组，$(0 : [\hat{P}/2], (0 : P), \ldots (0 : P)$。也可以是一维数组，长度为 $n_c\left(N, \hat{P}\right)$。由 COLLIER 在初始化过程中表格化，可以由以下函数获取

```
function GetNc_cll(N,R) result(nc)
integer N,R,nc:
```

integer N,R,nc:$N, \hat{P}, n_c(N, \hat{P})$

最后，可以给出额外的输出数组 TNerr ，添加 $\left(0 : \hat{P}\right)$ 个 type double precision 的 entries 到参数列表。如果存在的话，这个数组的成员传递了张量系数 $T^N_{i1\cdot iP}$ 的决定误差，其中所有的 $i_k \neq 0$，对于相应的 rank P。误差估计 $\Delta T^N(P)$ 大概由以下方法决定：迭代计算中的误差传递，和展开式中忽略的高阶项（见 section 3.1）。返回的误差值不应该被理解为精确且可靠的，而应该被当成低层不确定性的数量级估计。

代替单独的 subroutine A_cll,..., G_cll, 也可以用通用 subroutine 计算任意 N 的张量系数

```
subroutine TN_cll(TN,TNuv,MomInv,mass2,Nn,R,TNerr)
```

generic TN_cll 的参数根特定的 A_cll,..., G_cll 的不同仅在于 additional integer Nn,

```
integer Nn: # of loop propagators (=N)
```

定义了圈图中传播子的数目。在 TN_cll 的情况下，momentum invariants **MomInv**, squared masses **mass2**，coefficients **TN**, **TNuv** 只能按照一维数组的方式处理，长度分别为 $n_{\mathcal{P}}, N,$ and $n_c\left(N,\hat{P}\right)$。

张量元 $T^{N,\mu_1,\cdots,\mu_P}$ with $N=1,\cdots,7$ 可以通过各自的 subroutine Aten_cll, ..., Gten_cll 进行计算。这些 subroutine Nten_cll 的参数结构如下

```
subroutine
        Nten_cll(TNten, TNtenuv, MomVec, MomInv, mass2, R, Tntenerr)
```

除了 MomInv 和 mass2，跟 N_cll 中的用法一致。这里还必须提供 $N-1$ 个四矢量 p_i，就是出现在的传播子中的那些。用符号 MomVec 表示。

```
double complex MomVec(...)
```

momentum components

可以用 $N-1$ 个数组表示，每个数组 $(0:3)$，也可以用单个数组，维数为 $(0:3,N-1)$。注意 MomVec, MomInv, mass2 的参数形式应该一样，或者是单个数组，或者是一堆参数。和 MomInv 一样，type double complex (尽管在现版本的 COLLIER 中不支持虚部)。对于单点积分 A_cll, MomVec 参数应该省略。

整数 R 代表张量积分的最高阶 $\hat{P}$,决定了输出数组 TNten and TNtenuv 的大小 (type complex)。如同之前描述的,它们可以是 4-维数组 $\left(0:\hat{P},0:\hat{P},0:\hat{P},0:\hat{P}\right)$，或者一维数组，长度为 $n_t\left(P\right)$。$n_t\left(P\right)$ 在 COLLIER 初始化的时候决定，可以用以下函数获得

```
function GetNt_cll(R) result(nt)
```

integer R,nt :$\hat{P}, n_t(\hat{P})$

同样可以获得误差估计，通过在参数列表中提供可选的 output array TNtenerr 。它的 entries $\left(0:\hat{P}\right)$of type precision 提供了，张量元对应 rank 绝对误差的幅值，如同在 subroutine N_cll 中。

除了使用独立的 Aten_cll,..., Gten_cll ，还可以用 generic subroutine 计算张量元到任意 N(原则上)

```
subroutine
        TNten_cll(TNten,TNtenuv,MomVec,MomInv,mass2,Nn,R,TNtenerr).
```

TNten_cll 跟 Aten_cll,...,Gten_cll 不同之处在于，多了一个 integer Nn 参数，用来指定圈图传播子的数目。

如果使用 TNten_cll, 那么 MomVec,MomInv,mass2 只能是单个数组的形式，长度为 $(0:3,1:N-1,),(1:n_{\mathcal{P}})$ and, $(0:N-1)$，而不能是参数集合。但是输出中的 Tnten TNtenuv 用户仍然可以选择使用 $\left(0:\hat{P},0:\hat{P},0:\hat{P},0:\hat{P}\right)$, 或者 $(1:n_t(P))$ 的形式。

显然，不管是系数 subroutine A_cll,..,G_cll，TN_cll，还是张量元 subroutine Aten_cll,...,Gten_cll,or TNten_cll 的调用，都在各自的输出中，给出了相应标量积分的结果。如果用户只对标量 1–,...,4–点主积分感兴趣，可以选择限制 rank:$R=0$(i.e.$\hat{P}=0$)，或者使用可选的 routines N0_cll=A0_cll,..,D0_cll：

```
subroutine N0_cll(TN0,MomInv,mass2)
```

double complex TN0: T_0^N.

这些 routines 提供了标量积分的结果，输出为单个变量 TN0 of type complex，而输入 MomInv and mass2 可以在通常的用法中作选择。注意 routines A0_cll,...,D0_cll 没有连接到 cache system，并且如果 3–点函数的 Gram 行列式和 4–点函数的 Cayley 行列式为零，可能会 fail。

最后，COLLIER 也提供 routines, 来计算 2–点系数的动量导数，在对外线粒子做波函数重整化的时候会用到。需要用到的 subroutine 是 DB_cll, 参数结构是

```
subroutine DB_cll(DB,DBuv,MomInv,mass2,R,DBerr)
double complex DB(...)
double comple DBuv(...)
double complex DBerr(...)
```

导数 $B'_{i_1\cdots i_{\hat{P}}}\left(p_1^2\right)\equiv\partial B_{i_1\cdots i_{\hat{P}}}\left(p_1^2\right)/\partial p_1^2$，的结果通过数组 DB and DBuv 返回。输入和输出参数的约定和 B_cll 完全类似。函数 $B'_0,B'_1,B'_{00},\text{and}B'_{11}$ 可以用以下 subroutines 得到，作为单个 double complex variables.

```
subroutine DB0_cll(DB0,MomInv,mass2)
douoble complex DB0,
```

```
subroutine DB1_cll(DB0,MomInv,mass2)
douoble complex DB1,
subroutine DB00_cll(DB0,MomInv,mass2)
douoble complex DB00,
subroutine DB11_cll(DB0,MomInv,mass2)
douoble complex DB011,
```

由于导数 $B'_{i_1\cdots i_{\tilde{P}}}\left(p_1^2\right)$ 没有被 cached, 所以 calls of the subroutine DB_cll, DB0_cll, DB1_cll, DB00_cl, and DB11_cll 与 COLLIER 的缓存系统不相干。

1.5.4 设置和获取参数

张量积分的结果不仅依赖于质量和动量参数的确切值，还依赖于 regularization parameters，as well as on technical parameters 决定约化方案的选择，和展开方法的迭代次数。最后的两组参数，对于一组确定的积分调用，通常是固定的。在 COLLIER 初始化期间，它们被初始化成默认值，在1.2中给出，并可以在稍后修改。稍后我们会给出这些参数的细节，以及使用 subroutine 改变或读取它们的值。

首先，我们注意到，COLLIER 的版本可以通过下面的 calling 获取：

```
subroutine GetVersionNumber_cll(version)
character(len=5) version
```

1.5.4.1 正规化参数

COLLIER 使用维数正规化来处理 UV 发散。因此 UV 发散积分的结果依赖于1.8中的正规子 Δ_{UV}, 和维数正规化能标 μ_{UV}，更精确地说，依赖于组合 $\Delta_{UV}+\ln\left(\mu_{UV}^2/Q^2\right)$，其中 Q^2 是张量积分中出现的一些能标。在微扰理论的固定阶，物理的 S-矩阵不依赖于 Δ_{UV}andμ_{UV}。在 COLLIER 中，Δ_{UV}andμ_{UV}^2 被视为 type double precision 的数值参数，默认值为 $\Delta_{UV}=0$and$\mu_{UV}^2=1$，其数值可以通过以下 subroutines 修改

```
subroutine SetDeltaUV_cll(delta)
double precision delta,
subroutine SetMuUV2_cll(mu2)
double precision mu2
```

一方面，通过 varying 这些参数，用户数值地可以验证 S–矩阵元的 UV 有限性。另一方面，在重整化方案如 MS or $\overline{MS}$，维数重整化标度 μ_{UV} 等同于跑动耦合 $g(\mu_{ren})$ 的重整化标度 $\mu_{\rm ren}$。在这种情况下，它具有了物理诠释，并且对 S–矩阵元有影响。Δ_{UV}andμ^2_{UV} 值可以通过 subroutines 得到

```
subroutine GetDeltaUV_cll(delta)
double precision delta,
subroutine GetMuUV2_cll(mu2)
double precision mu2
```

默认行为是。IR 发散也通过维数正规化。因此 IR-发散的积分，其结果依赖于 $\Delta^{(1)}_{IR}$and $\Delta^{(2)}_{IR}$defined in 1.8，and scale of dimensional regularization, μ_{IR}。在微扰理论的固定阶，IR-finite 的物理量不依赖于 $\Delta^{(1)}_{IR}$, $\Delta^{(2)}_{IR}$and μ_{IR}，一旦 virtual and real 修正被组合。在 COLLIER 中，$\Delta^{(1)}_{IR}$, $\Delta^{(2)}_{IR}$and μ^2_{IR} 用 type double precision 的数值参数来表示，默认值为 $\Delta^{(1)}_{IR} = \Delta^{(2)}_{IR} = 0$ and $\mu^2_{IR} = 1$，可以通过以下 subroutines 修改

```
subroutine SetDeltaIR_cll(delta1,delta2)
double precision delta1,delta2
subroutine SetMuIR2_cll(mu2)
double precision mu2
```

注意，特别地，$\Delta^{(1)}_{IR}$and $\Delta^{(2)}_{IR}$ 可以被独立地改变。对 $\Delta^{(1)}_{IR}$, $\Delta^{(2)}_{IR}$and μ^2_{IR} 的 variation 可以对可观测量的 IR 有限性进行数值检验。现有的值可以通过以下 calling 获取

```
subroutine GetDeltaIR_cll(delta1,delta2)
double precision delta1,delta2,
subroutine GetMuIR2_cll(mu2)
double precision mu2
```

collinear 发散也可以通过引入一列质量正规子来正规化，

$$\mathcal{R}_{n_{reg}} = \left\{\overline{m}^2_1, \overline{m}^2_2, \ldots, \overline{m}^2_{n_{\rm reg}}\right\}$$

为了使用此功能，用户可以这样设置

```
subroutine SetMinf2_cll(nminf,minf2)
double complex minf2(nminf)
```

```
integer nminf
```

其中 integer 变量 nminf 代表不同正规子质量的数目 $n_{\rm reg}$，数组 minf2 包含了 $\overline{m}_i^2$ 的平方值，of type double complex。可选择地，正规子质量可以相继被添加，通过 calling the subroutine

```
subroutine AddMinf2_cll(m2)
double complex m2
```

它让 $n_{\rm reg}$ 增加 1，然后添加 double complex value m2 到列表 $\mathcal{R}_{n_{\rm reg}}$ 中。当一个张量积分被调用，它的参数（质量平方和动量不变量）被数值地与 $\mathcal{R}_{n_{\rm reg}}$ 的元素进行比较。相同的 entries 被当成无穷小 throughout the calculation，它们的数值（并不需要很小）are only kept in otherwise singular logarithms。在 calls of all subroutines，small masses 具有 exactly 相同的值是很重要的。mass regulators 的数目 $n_{\rm reg}$and list of squared values can be read out with

```
subroutine GetNminf_cll(nminf)
subroutine GetMinf2_cll(minf2)
integer nminf
```

finally，the subroutine

```
subroutine ClearMinf2_cll
```

允许清除列表 $\mathcal{R}_{n_{\rm reg}}$and 重置 $n_{\rm reg}$to zero.

1.5.4.2 技术参数

COLLIER 可以在三种不同模式下运行，通过

```
subroutine SetMode_cll(mode)
integer mode
```

其中 integer argument mode=1,2,3。For mode=1(默认值)，使用 COLI branch ，for mode=2, the DD branch is used, for mode=3, the integrals 在两种模式下都计算。在最后一种情形下，COLLIER 会返回两种结果中更好的那个。误差估计（如果在调用中指定可选参数的话）则根据两个分支的具体情况，并返回大的那一个。COLI and DD 之间大于一定阈值的差别，将会被存储的文件 CheckOut.cll 中。mode 的值可以被获取

```
subroutine GetMode_cll(mode)
integer mode
```

计算中设定的精度目标 $\eta_{\rm req}$ 可以通过下列 calling 设定

```
subroutine SetReqAcc_cll(acc)
double precision acc
```

参数 acc of type double precision。为了达到精度目标，COLLIER 会选择合适的 scheme，如果必要的话会作多种选择，展开到足够的阶。因此，$\eta_{\rm req}$ 的选择决定了结果的 precision 和运行时间，最好权衡一下。默认值是 $\eta_{\rm req=10^{-8}}$，并且 library 对此设定进行了优化。$\eta_{\rm req}$ 的当前值可以如下获得：

```
subroutine GetReqAcc_cll(acc)
double precision acc
```

至于实际结果满足精度 $\eta_{\rm req}$ 的程度取决于问题的复杂度。作为第二道精度门槛，一个关键精度 $\eta_{\rm crit}$，应该比 $\eta_{\rm req}$ 要大，可以如下设定

```
subroutine SetCritAcc_cll(acc)
double precision acc
```

参数 acc of type double precision。关键精度并不影响实际计算，它只是一个记账策略：如果计算的积分序列中有一个在某相空间点的不确定度达到 $\eta_{\rm crit}$，就升起一个 accuracy flag 来指示一个警告。用户可以查询这个 flag，然后决定如何继续（比如放弃这个相空间点，或者改用不同的方法等等）。而且，关键的积分可以被监视。如果这个选项启用，它们的参数和结果会被自动写入到输出文件里。更多关于 accuracy flag and 监视关键积分的信息在 section 5.6 给出。关键精度被初始化为 $\eta_{\rm crit} = 10^{-1}$；它的值可以如下设定

```
subroutine GetCritAcc_cll(acc)
double precision acc
```

最后，第三个精度参数 $\eta_{\rm check}$，应该比 $\eta_{\rm req}$ 大，管理 COLI 和 DD 结果的比较。$\eta_{\rm check}$ 的默认值应该是 $\eta_{\rm check} = 10^{-4}$；它可以如下修改：

```
subroutine SetCheckAcc_cll(acc)
subroutine GetCheckAcc_cll(acc)
double precision acc
```

For mode=3，如果 COLI 和 DD 的结果差距超过 $\eta_{\rm check}$，将会被记录到 file CheckOut.cll 中。mode=1 and mode=2 的时候，$\eta_{\rm check}$ 参数无关。

除了分别使用 subroutines 设置，还可以用一个 subroutine 同时设置 $\eta_{\rm req}$，$\eta_{\rm crit}$，$\eta_{\rm check}$，

```
subroutine SetAccurary_cll(acc0,acc1,acc2)
double precision acc0,acc1,acc2
```

double precision arguments acc0,acc1,acc2 分别代表 $\eta_{\rm req}$，$\eta_{\rm crit}$，$\eta_{\rm check}$。

一个更重要的技术参数是，在迭代计算中，张量积分的最高 rank $\hat{P}^{\rm max}$，它也定义了展开方法的 cut-off order。可以如下进行设定

```
subroutine SetRitmax_cll(ritmax)
integer ritmax
```

其中 ritmax 可以大于等于 7。ritmax 的值可以通过以下的方式获取

```
subroutine GetRitmax_cll(ritmx)
```

如果对于 4–点函数，$\hat{P}^{\rm max} \geq 7$ 作为最高 rank，那么在内部，对于 3–点和 2–函数，自动设置为 $\hat{P}^{\rm max}+2$and $\hat{P}^{\rm max}+4$。所以如果 $N \leq 4$，$\hat{P}^{\rm max}$ 的值将会影响计算精度和时间（from external and internal calls），library 为默认值 $\hat{P}^{\rm max}=14$ 特地进行了优化。注意，为了能够计算 $N=N_{\rm max}$and $P=P_{\rm max}$ 的所有张量积分 $T^{N,P}$（$N_{\rm max}, P_{\rm max}$ 的值在初始化 call of Init_cll 中被指定），$\hat{P}^{\rm max}$ 的值不能小于 $P_{\rm max}+4-N_{\rm max}$。

如同在 section 3 中解释的，对于 $N \geq 6$，约化方法既用 $T^{N}_{i_1,\ldots i_P}$ 实现，也用 $T^{N,\mu_1\cdots\mu_P}$ 实现了。所以当 $N \geq 6$ 时，对于一个通常的 $T^{N,\mu_1\cdots\mu_P}$ 的计算分为 3 步：首先，对于 $5 \leq \bar{N} = N_{\rm tenred}-1 \leq 6$，系数 $T^{\bar{N}}_{i_1,\ldots i_{P_{\bar{N}}}}$ 递归的从 2–点系数开始计算。然后用 $T^{\bar{N}}_{i_1,\ldots i_{P_{\bar{N}}}}$ 构建 $T^{\bar{N},\mu_1\cdots\mu_{P_{\bar{N}}}}$。最后，张量 $T^{N,\mu_1\cdots\mu_P}$ 递归地计算自 $T^{\bar{N},\mu_1\cdots\mu_{P_{\bar{N}}}}$，见图1.6。阈值 $N_{\rm tenred}$-从何处开始张量约化，可以如下设定和获取

```
subroutine SetTenRed_cll(Ntenred)
subroutine GetTenRed_cll(Ntenred)
integer Ntenred
```

其中参数 Ntenred 代表参数 $N_{\rm tenred}$，subroutine

```
subroutine SwitchOnTenRed_cll
```

$B_{i_1\ldots i_P}$ $C_{i_1\ldots i_P}$ $D_{i_1\ldots i_P}$ $E_{i_1\ldots i_P}$ $F_{i_1\ldots i_P}$ $G_{i_1\ldots i_P}$ ...

$B^{\mu_1\cdots\mu_P}$ $C^{\mu_1\cdots\mu_P}$ $D^{\mu_1\cdots\mu_P}$ $E^{\mu_1\cdots\mu_P}$ $F^{\mu_1\cdots\mu_P}$ $G^{\mu_1\cdots\mu_P}$ ...

Figure 1: Reduction chains in COLLIER: For $N \geq 6$ reduction can be performed at the tensor level.

图 1.6: figure1 reduction chains

等价于 SetTenRed_cll(Ntenred) 并令 Ntenred=6，opts for the maximal level of tensor reduction，而 subroutine

```
subroutine SwitchOffTenRed_cll
```

则将它完全关闭。默认设定是 maximal tensor reduction $N_{\text{tenred}} = 6$，考虑到 run time 因素。

1.5.5 使用缓存系统

COLLIER 安排有 Cache 系统，避免重复计算相同的积分，以此来加快运算速度。它可以运行在 local 或者 global mode：在 local mode，仅仅在单个 subroutine call 的过程中（Section 5.3），积分被存储。cache 系统探测在约化过程中，通过不同路径到达的相同积分，然后避免它们的重复计算。在 Global 中，不同的 subroutine call 也被连接起来。local mode 总是在工作的，而 global mode 需要被显式指定。为了这个目的，也许需要创建 n_{cache} 个单独的 cache，来储存系数或者张量的结果。可以实现如下

```
subroutine InitCacheSystem_cll(ncache,Nmax)
integer ncache, Nmax
```

其中参数 ncache and Nmax 分别代表，缓存的总数目 n_{cache} 以及计算 N–point 积分被 cached 的数目 up to$N = N_{\text{cache}}^{\max}$。为了启用 global mode，必须在每个计算每个相空间点之前，用 call of InitEvent(cacheNr) 把这个队列的 integral calls 分配到第 cacheNr 缓存。我们强调，对于每一个相空间点，integral calls 必须按照相同的顺序，并且在同一个相空间点，global parameters(like μ^2_{UV}，mode of COLLIER 等等) 不能被 reset，因为积分是依靠在用户调用序列中的次序来 identified。注意对 cacheNr 的数目没有限

制，需要的内存被动态分配，during the first phase-space points。取决于所解决的问题，cache in global mode 可能会引起 hig use of memory resources。

除了在一开始固定缓存总数 n_{cache}，还能在后续添加缓存，通过 calling of subroutine

```
subroutine AddNewCache_cll(cache_no,Nmax)
integer cache_no,Nmax
```

新的 cache，被初始化为最多储存 Nmax–点积分的结果作为输入，分配的缓存数目通过 output 参数 cache_no 返回。如果之前没有初始化 cache system，call of AddNewCache_cll 相当于 call of InitCacheSystem(ncahce=1,Nmax)。

The threshold $N_{\text{cache}} \leq N_{\text{cache}}^{\max}$up to which 积分被 cached，可以被单独调整，for each cache。使用如下 subroutine

```
subroutine SetCacheLevel_cll(cache_no,Nmax)
integer cache_no,Nmax
```

注意第 cache_no 个缓存的 level $N_{\text{cache}}^{\max}$，只能在这个缓存的第一个相空间点被计算之前改变。(i.e. 在 InitEvent_cll 被首次计算之前，with the argument cache_no)

可以使用 subroutine

```
subroutine SwitchOffCacheSystem_cll
```

暂时关闭 global cache 系统。如果需要临时在 integral calls 序列中加入额外的 call，这个选项将会很有用

```
subroutine SwitchOnCacheSystem_cll
```

将 global 缓存再次打开，将在它被中断的地方重新开始工作。

也可以之关闭一个特定的 cache calling

```
subroutine SwitchOffCache_cll(cache_no)
integer cache_no
```

在这种情况下，使用

```
subroutine SwitchOnCache_cll(cache_no)
integer cache_no
```

再次打开。cache 在被暂停的地方重新开始，或者，如果在暂停期间 subroutine InitEvent 被调用，从缓存 cache_no 积分列表中的第一个开始。

1.5.6 错误处理和输出文件

内部错误或者精度不够，在 COLLIER 中有两种处理方法：一方面，可以在运行过程中，设置和读取 flags for errors and accurary；另一方面，对应的错误信息，和出错的积分 calls，将会被记录在输出文件中。

error flag $\sigma_{\rm err}$ 获取 calling

```
subroutine GetErrFlag_cll(errflag)
integer errflag
```

integer errflag 的值从 $\sigma_{\rm err} = 0$（无错误）到 $\sigma_{\rm err} = -10$（fatal errors)。$\sigma_{\rm err}$ 保存它的值，直到被重写为更小的负数，表明遇到更加严重的错误。如此，errflag 指出了它遇到的最严重的错误。当 call of InitEvent_cll 之后，对于新相空间点的计算，它自动被重置为 $\sigma_{\rm err} = 0$，也可以在任意时间重置

```
subroutine InitErrFlag_cll
```

如果 $\sigma_{\rm err}$ 的值小于一个门槛 $\sigma_{\rm stop}$，程序的执行将会自动停止。默认值是 $\sigma_{\rm stop} = -8$，以使得对于相空间点的特定错误计算不会停止，而在所有相空间点出现共性错误时，停止计算。可以给 $\sigma_{\rm stop}$ 设置不同的值，或者获取它的值，通过给 subroutine 提供 integer argument stopflag

```
subroutine SetErrStop_cll(stopflag),
subroutine GetErrStop_cll(stopflag)
integer stopflag
```

可以通过调用

```
subroutine SwitchOffErrStop_cll()
```

避免程序停止。

精度 flag $\sigma_{\rm acc}$ 的工作方式类似。它反映了结果的精度，可以作为 integer argument accflag 被获取

```
subroutine GetAccFlag_cll(accflag)
integer accflag
```

初始化为 $\sigma_{\rm acc} = 0$，如果没有达到 $\eta_{\rm req}$，就降为 $\sigma_{\rm acc} = -1$，如果没有达到 $\eta_{\rm crit}$，就降为 $\sigma_{\rm acc} = -2$（这些参数的细节见1.5.4.2)。想在 error flag 的情形一样，$\sigma_{\rm acc}$ 会被更加小的负值覆盖，来表明计算中最差的精度（从 $\sigma_{\rm acc}$ 被

初始化之后)。call of InitEvent_cll for a new phase-space point 将会自动初始化 $\sigma_{\text{acc}} = 0$，也可以用

```
subroutine InitAccFlag_cll
```

在初始化 COLLIER 的时候，用户可以选择 errors and accurary 的 messages 返回的方式。默认地，COLLIER 把它们存储在

```
./output_cll/
```

下的独立文件中。像在 section1.5.2中描述的，用户可以自定义输出路径，通过添加相应的字符串，作为 subroutine Init_cll 的第二个可选参数。如果传入一个空字符串，那么相当于不创建输出文件夹。这个预定义的设置可以在随后修改，通过

```
subroutine SwitchOffFileOutput_cll,
subroutine SwitchOnFileOutput_cll,
```

或者创建新的输出文件夹

```
subroutine SetOutputFolder_cll(fname)
character(len=*) fname.
```

输出文件夹的名字是 fname，可以被获取

```
subroutine GetOutputFolder_cll(fname)
character(len=*) fname
```

Error messages 被导出至 files ErrOut.coli, ErrOut.dd, and ErrOut.cll 取决于 err 来自于 COLI,DD,or global 接口 (or module tensors)。在初始化 COLLIER 的时候,这些文件被创建,一个 free output channel(number>100) 被分配给它们。Output channel 可以被用户手动分配，通过

```
subroutine SeterroutCOLI_cll(outchan)
subroutine SeterroutDD_cll(outchan)
subroutine Seterrout_cll(outchan)
integer outchan
```

channel number outchan 是 integer 参数。尤其是,可以通过选择 outchan=6,将 error 重定向到标准 channel。注意,如果文件输出被关闭,通过 subroutine SwitchOffFileOutput_cll，standard channel 并不会被关闭，而且，重定向

到标准 channel（terminal 或者一个专用文件）的 COLLIER 输出会继续传递。当前选择的输出 channel 可以如下获取

```
subroutine GeterroutCOLL_cll(outchan)
subroutine GeterroutDD_cll(outchan)
subroutine Geterrout_cll(outchan)
integer outchan.
```

为了避免输出文件体积过大，默认的 error messages 展示数目为 100。可以通过如下更改

```
subroutine SetMaxErrOutCOLI_cll(nout)
subroutine SetMaxErrOutDD_cll(nout)
subroutine SetMaxErrOut_cll(nout)
integer nout
```

指定对应的 integer 数目 nout 即可。默认 error 等的 counters 在每次 COLLIER 重新初始化之后被重置，如果不希望重置的话，可以将 noreset=.true. 传递给 Init_cll，这些 counters 也可以手动重置

```
subroutine InitErrCntCOLI_cll
subroutine InitErrCntDD_cll
subroutine InitErrCnt_cll
```

error 输出可以被 dis- and enabled by calling

```
subroutine SetErrOutLev_cll(outlev)
integer outlev
```

其中 outlve=0 或者 outlev=1。如果 COLLIER 初始化的时候被传递了空数组作为输出文件夹的名字，那么 error 输出默认被关闭，其他情况都是被打开的。

额外信息和与 errors 无关的状态信息被记录在 log-file InfOut.cll，在初始化 COLLIER 时指定的目录中。同时也会有一个 free output channel(number $>$ 100) 自动产生，并且可以被用户修改，通过传递 integer outchan to

```
subroutine Setninfout_cll(outchan)
integer outchan
```

同样允许设置为标准 channel outchan=6。获得现在的 channel 可以用

```
subroutine Getinfout_cll(outchan)
integer outchan
```

默认 output 上限是 $n_{\rm inf}^{max} = 1000$，但是用户也可以修改

```
subroutine SetMaxInfOut_cll(nout)
integer nout
```

默认情形下，$n_{\rm inf}^{max}$ 以及其他的 counter 均会在 COLLIER 重新初始化期间被重置，除非设置了 noreset=.true.，作为 Init_cll 的额外参数。informative 输出的程度可以被控制

```
subroutine SetInfOutLev_cll(outlev)
integer outlev
```

其中的 integer argument outlev=0,1,2。用空数组作为输出文件夹时，隐含了 outlev=0(不输出)，其他情况输出被设置为 outle=2（最大输出）。在后一种情况，任何内部参数的改变，都会被记录在输出文件中，可能会导致文件体积很大。比如有些参数（UV 标度 $\mu_{\rm UV}^2$）被重复修改的时候（例如对于每个相空间点）。因此，也提供了折中的输出层次 outlev=1，它只追踪那些特别的活动，发生的频率较低。

当 COLLIER 的模式被首次切换到 mode=3 的时候，CheckOut.cll 在通常的输出文件夹中创建。同样有相应的 channel

```
subroutine Setncheckout_cll(outchan)
subroutine Getncheckout_cll(outchan)
integer outchan
```

在 mode=3，积分同时用 library 中的 COLI 和 DD branch 计算，文件 CheckOut.cll 收集这些积分的 input and results ，如果相对偏差大于 $\eta_{\rm check}$（见 Section1.5.4.2）。2–点函数的导数也会被比较和保存，如果偏差大于 $\eta_{\rm check}$。输出数目被限制到 $n_{\rm check}^{\rm N,max}$ 个问题积分和 $n_{\rm check}^{\rm B\prime,max}$ 个问题导数。对于每个 N，可以单独修改 $n_{\rm check}^{\rm N,max}$

```
subroutine SetMaxCheck_cll(npoints,N)
integer npoints,N
```

npoints and N 代表 $n_{\rm check}^{\rm N,max}$ 和 N。对于 $n_{\rm check}^{\rm B\prime,max}$ 也是类似的

```
subroutine SetMaxCheckDB_cll(npoints)
integer npoints
```

也可以一次性设定 $n_{\text{check}}^{1,\max},\cdots,n_{\text{check}}^{\text{N},\max}$，通过传递 integer array$\left\{n_{\text{check}}^{1,\max},\cdots,n_{\text{check}}^{\text{N},\max}\right\}$ 作为单个参数 npointarray 给

```
subroutine SetMaxCheck_cll(npointarray)
integer npointarray(Nmax)
```

注意 $N_{\max}$ 的值从上次 COLLIER 重新初始化时，call of Init_cll 之后就是固定的（见1.5.2）。初始值是 $n_{\text{check}}^{1,\max}=\cdots=n_{\text{check}}^{\text{N},\max}=n_{\text{check}}^{\text{B}\prime,\max}=50$。call of Init_cll 导致 limits 和 counters 的重新初始化，除非 noreset=.true.。更进一步的，注意 counters for output messages in Checkout.cll 可以被手动重置为 0，通过

```
subroutine InitCheckCnt_cll
```

对于 N—点积分，and

```
subroutine InitCheckCntDB_cll
```

对于 2–导数。

此外，用户也可以要求关于积分更详细的信息，for which 估计精度没有达到门槛 η_{crit}（见1.5.4.2）这个 feature 必须显式开启

```
call InitMonitoring_cll
```

激活之后，input and outputs for integrals 没有达到指定精度的，将会被记录到 CritPointsOut.cll。同样分配有 channel number

```
subroutine Setncritpointsout_cll(outchan)
subroutine Getncritpointsout_cll(outchan)
integer outchan
```

输出被限制为头 $n_{\text{crit}}^{\text{N},\max}=50$ 个有问题的 N–点积分和头 $n_{\text{crit}}^{\text{B}\prime,\max}=50$ 个导数. 这些 limits 都可以更改

```
subroutien SetMaxCritPoints_cll(npoints,N),
subroutien SetMaxCritPoints_cll(npointarray),
subroutien SetMaxCritPointsDB_cll(npoints),
integer npoints,N
integer npointarray(Nmax)
```

工作方式和 subroutine SetMaxCheck_cll and SetMaxCheckDB_cll 是完全相似的。初始化或者后续初始化 of COLLIER 不会改变 $n_{\rm crit}^{\rm N,max}, n_{\rm crit}^{\rm B\prime,max}$ 和对应的 counters。但是，call of InitMonitoring_cll 将重置 $n_{\rm crit}^{\rm 1,max} = \cdots = n_{\rm crit}^{\rm N,max} = n_{\rm crit}^{\rm B\prime,max} = 50$，相应的 counters 也将重置为 0。若想只重置 conunters, 可以 calling

```
subroutine InitPointsCnt_cll.
```

1.5.7 示例程序

在文件夹 diretory COLLIER-v/build 中执行命令

```
make demo
make democache
```

会产生两个示例程序，可以通过下面的命令执行

```
./demo
./democache
```

in the folder COLLIER-v/demos

程序 demo 专门用来计算单个张量积分。在运行过程中，用户会被要求指定 mode, 并在众多 N–点积分的例子中选择一个偏爱的进行计算。计算结果写入到文件 demo_Npoint_exampleX.dat 中，引导用户查找到 demo.90 中各个积分的计算程序。在许多例子中，展示了同一个积分的各种调用，展示了如何与 subroutine 传递参数的不同方式。例子中使用的变量定义在文件 demo.f90 的一开始。其中包含了很多注释，通过移除前面的感叹号，可以激活它们，以此来改变 COLLIER 的各种全局参数。

程序 democache 展示了缓存的使用。对 1000 个相空间的点集，计算了一系列共 8 个张量积分，并重复几次。这个玩具 Monte Carlo 在四种配置下分别计算：使用 COLI 分支，with or without cache, 然后使用 DD 分支，with and without cache。源代码放在 democache.f90 中。

1.6 总结

fortran-based library COLLIER 数值计算单圈标量或张量积分，并且对粒子的 multiplicities 没有先天的限制。COLLIER 的特别在于：对相空间的

delicate 区域, 用专用技术自动优化数值稳定性, 支持不稳定粒子的 complex 质量, 对于红外发散, 可以选择使用维数或质量正规化。此外, COLLIER 支持检查结果的正确性和数值稳定性, 由于它使用了两种独立的积分 libraries, COLI and DD。

COLLIER 可以用在传统的费曼图方法和现代的幺正性方法中。The library 已经是 essential building block 的代表,在自动化单圈振幅 generator, 如 OPENLOOPS and RECOLA, 也将被更多其他生成器所使用。

1.7 附录 A. 动量不变量的集合，对于 $N = 1, ..., 7$

N–点张量积分依赖于动量不变量 $\mathcal{P}_N$ 的完整集合, $\mathcal{P}_N$ 由1.9式中传播子中的动量 p_i 组成。我们对集合 $\mathcal{P}_N$ 中元素次序的约定在1.10和1.11中给出。为了方便, 我们 $N = 2, \cdots, 7$ 的 $\mathcal{P}_N$ 清楚地列在这里:

$$\mathcal{P}_2 = \left\{p_1^2\right\},$$

$$\mathcal{P}_3 = \left\{p_1^2, (p_2-p_1)^2, p_2^2\right\},$$

$$\mathcal{P}_4 = \left\{p_1^2, (p_2-p_1)^2, (p_3-p_2)^2, p_3^2, p_2^2, (p_3-p_1)^2\right\},$$

$$\begin{aligned}\mathcal{P}_5 =\{&p_1^2, (p_2-p_1)^2, (p_3-p_2)^2, (p_4-p_3)^2, p_4^2,\\ &p_2^2, (p_3-p_1)^2, (p_4-p_2)^2, p_3^2, (p_1-p_4)^2\},\end{aligned}$$

$$\begin{aligned}\mathcal{P}_6 =\{&p_1^2, (p_2-p_1)^2, (p_3-p_2)^2, (p_4-p_3)^2, (p_5-p_4)^2, p_5^2,\\ &p_2^2, (p_3-p_1)^2, (p_4-p_2)^2, (p_5-p_3)^2, p_4^2, (p_1-p_5)^2,\\ &p_3^2, (p_4-p_1)^2, (p_5-p_2)^2\},\end{aligned}$$

$$\begin{aligned}\mathcal{P}_7 =\{&p_1^2, (p_2-p_1)^2, (p_3-p_2)^2, (p_4-p_3)^2, (p_5-p_4)^2, (p_6-p_5)^2, p_6^2,\\ &p_2^2, (p_3-p_1)^2, (p_4-p_2)^2, (p_5-p_3)^2, (p_6-p_4)^2, p_5^2, (p_1-p_6)^2,\\ &p_3^2, (p_4-p_1)^2, (p_5-p_2)^2, (p_6-p_3)^2, p_4^2, (p_1-p_5)^2, (p_2-p_6)^2\}.\end{aligned}$$

end of file

end of file

参考文献

[1] ref61

[2] ref37

[3] ref50

[4] ref45

[5] ref52

[6] ref59

[7] ref43