QMUL-......
SAGEX-.....

SpinorHelicity4D: a Mathematica toolbox for the four-dimensional spinor-helicity formalism

Manuel Accettulli Huber

Centre for Research in String Theory
School of Physics and Astronomy
Queen Mary University of London
Mile End Road, London E1 4NS, United Kingdom

Abstract

We present the Mathematica package SpinorHelicity4D, a dedicated suite for analytic and numeric calculations involving four-dimensional massless and massive spinor-helicity formalism. Analytic features of the package include for example: manipulation of contracted and uncontracted spinor quantities, automated application of Schouten identities for expression simplification, contractions of spinor products into chains, re-expression of chains in terms of Dirac traces and evaluation of such traces, derivatives of arbitrary functions of spinor quantities. Numeric features of the package include among others: generation of arbitrary n-point numerical complex kinematics, allowing for both massless and massive external states, fully numeric or parametric kinematics, and numeric generation on either $\mathbb{R}$ or $\mathbb{Q}$, the latter providing output immediately suitable for finite field applications. Furthermore, the package features userfriendly, intuitive but also highly customizable input options, thus providing an approachable tool for the casual user while still supporting more advanced applications for more adept and frequent users. All of the output is returned in the standard bracket notation, making it easily interpretable, but at the same it retains all of the analytic properties of the objects, allowing for copy-pasted and manipulated output to be provided as new input. This makes it ideal for front-end applications on a Mathematica notebook, while still allowing for deployment on a cloud server for more heavy calculations.

m.accettullihuber@qmul.ac.uk

Contents

1 Introduction

The spinor-helicity formalism provides a parametrization of scattering amplitudes which, in many instances, is arguably the most convenient. A classic example is the Parke-Taylor [1] formula for MHV gluon-amplitudes, which in spinor notation reduces to a single term at any multiplicity n:

$$\mathcal{A}_n^{\text{tree}}(1^+,\dots,i^-,\dots,j^-,\dots,n^+) = ig^{n-2}\frac{\langle i\,j\rangle^4}{\langle 1\,2\rangle\langle 2\,3\rangle\cdots\langle n\,1\rangle}\,. \tag{1}$$

This formalism is very flexible, and while originally it was mainly used for four-dimensional massless particles like in the Parke-Taylor case, it has since been adapted to a variety of different settings, including for example four-dimensional massive [2, 3] and six-dimensional massless particles [4]. In this paper, we present a `Mathematica` package designed to deal with four-dimensional massless and massive particles, where in particular massive particles are represented through a pair of massless states.

In the design of the package, great effort has been put into keeping it accessible to the largest possible audience but without sacrificing the possibility of performing more advanced manipulations, which inevitably require larger amounts of technical knowledge. For the sake of userfriendliness four different ways of expression input are available, ranging from a very intuitive palette to the possibility of defining custom names for every object in the package. The latter functionality is aimed, among others, at providing a smooth transition from the use of other codes, be they private or other public packages like S@M [5], to the here presented routines. Output of the functions is always presented in the standard bracket notation. This was achieved through the use of `Mathematica`'s "box" functions, which provide the backbone of the front-end display of `Mathematica`'s own built-in objects. Consequently, the output representation does not affect the objects' analytic properties which are preserved at any time, allowing for output to be copied, pasted, edited and then independently re-run[1].

Input objects featured in the package include both the Lorentz-invariant spinor brackets and chains, as well as uncontracted covariant spinors and Levi-Civita tensors, and momentum and polarization vectors. Whenever uncontracted indices are present, standard contraction properties are automatically applied to form Lorentz-invariants, *i.e.* $\lambda_1^a\lambda_{2\,a} \to \langle 1\,2\rangle$ and $p^\mu q_\mu \to p\cdot q$. Intrinsic object properties are always accounted for, like for example antisimmetry of the spinor brackets or linearity of the scalar product. Some of these properties, including linearity among others, requires explicit declaration of momentum labels or of masslessness of states, so that no assumption is made internally by the code and the user retains full control over expressions. Whenever such a declaration is required, it is highlighted both in the attached example notebook as well as in this user guide by a light-gray cell background, and it is clearly stated in the function documentation. As one might expect, the main purpose of most of the analytic features in the package is to convert invariants among each other and apply properties so to simplify expressions. Some prime examples include `ChainSimplify`, which makes use of the Clifford algebra relation $\{\sigma^\mu,\bar{\sigma}^\nu\} = 2\eta^{\mu\nu}$ to simplify products of spinor brackets closed into continuos chains, and `SchoutenSimplify` which applies the Schouten identity $\langle i\,j\rangle\langle k| + \langle j\,k\rangle\langle i| + \langle k\,i\rangle\langle j| = 0$. These routines are entirely automated and do not require the user to specify which particular replacement to perform in order to simplify a given expression.

Every object present in the package can also be numerically evaluated. This allows for quick comparisons even of very intricate analytic expressions, but it is especially useful when combined with functional reconstruction techniques (see [6]) or ansatz generation with consequent system solution (see [7]), allowing to recover analytic results from numerical evaluations. With this

[1]This holds true both in `StandardForm` as well as `TraditionalForm`.

in mind, numeric kinematics is by default generated on $\mathbb{Q}$, ready to be mapped to finite fields. Numeric kinematics can be generated for a wide range of processes, including for example mixed massive-massless external states, or amplitudes on a unitarity cut where the momenta of some legs of one amplitude are related to momenta of another one.

The present user guide is structured into three parts, reflecting the three modules which compose the package. First, upon recalling some of the properties of the formalism and explaining how to get access to the package, we introduce the individual building blocks and their properties, all of which are defined in the module `SpinorBuildingBlocks` and their list can be accessed through the standard `?SpinorBuildingBlocks*`. Next, we go through the functions responsible for the manipulation of the analytic objects, which are defined in the module `SpinorHelicity4D`, and lastly we describe the numeric features provided by the module `SpinorNumerics`. As a final remark, it is important to stress that, despite most of the functions we present have been extensively tested and used by the author, bugs happen. We would be grateful for any bug-related flag which is raised, either through email or directly on github, but also gladly take any suggestion for improvement and welcome any further third-party development based on the functions here presented[2].

2 Building blocks and expression input

2.1 A lightning review of the spinor-helicity formalism

In this first section we briefly review the four-dimensional spinor-helicity formalism, with the main purpose of setting our notations and conventions. We limit our discussion to Weyl spinors only, which are our building blocks of choice. For a more detailed review of the topic see for example [8–11].

Definition of spinors

The proper orthochronous Lorentz group $\mathrm{SO}^+(1,3)$ admits a universal covering given by $\mathrm{SL}(2,\mathbb{C})$, consequently there is a one-to-one correspondence between projective representations of $\mathrm{SO}^+(1,3)$ on the Hilbert space and the infinite-dimensional unitary representations of $\mathrm{SL}(2,\mathbb{C})$. The states of the theory transform under such unitary representations and induce the fields to transform under finite-dimensional (non-unitary) representation. All the irreducible finite-dimensional representations of $\mathrm{SL}(2,\mathbb{C})$ are labelled by a pair of semi-integers (m_L, m_R)[3], and can be obtained from completely symmetrized tensor products of $2m_L$ copies of the fundamental and $2m_R$ copies of the anti-fundamental representations, usually denoted as $\left(\frac{1}{2},0\right)$ and $\left(0,\frac{1}{2}\right)$.

The fundamental two-dimensional objects transforming in the $\left(\frac{1}{2},0\right)$ are denoted by λ_α and the associated Lorentz indices are greek undotted letters, whereas $\tilde{\lambda}^{\dot{\alpha}}$ transform in the $\left(0,\frac{1}{2}\right)$ and the associated indices are dotted greek letters. These objects will collectively be called (Weyl) *spinors*. The spinor indices can be raised and lowered by contraction with a two-dimensional Levi-Civita tensor as

$$\lambda^\alpha = \epsilon^{\alpha\beta}\lambda_\beta \ , \qquad \tilde{\lambda}_{\dot{\alpha}} = \epsilon_{\dot{\alpha}\dot{\beta}}\tilde{\lambda}^{\dot{\beta}} \ , \tag{2}$$

where in our convention

$$\epsilon^{12} = -\epsilon_{12} = \epsilon^{\dot{1}\dot{2}} = -\epsilon_{\dot{1}\dot{2}} = 1 \ , \tag{3}$$

[2]We will happily expand the package through external code by the standard pull-request mechanics on git.

[3]Recall that the algebra $\mathfrak{sl}(2,\mathbb{C})$ is isomorphic to $\mathfrak{su}(2)_L \times \mathfrak{su}(2)_R$, and (m_L, m_R) are related to the eigenvalues of the Casimir operators $\mathbf{J}^2_{L/R} = (J^1_{L/R})^2 + (J^2_{L/R})^2 + (J^3_{L/R})^2$, with $J^i_{L/R}$ generators of the $\mathfrak{su}(2)_{L/R}$. Elements of the two-dimensional vector space $\mathfrak{su}(2)_L$ will be denoted with λ and elements of $\mathfrak{su}(2)_R$ with $\tilde{\lambda}$.

and

$$\epsilon_{\alpha\beta}\epsilon^{\beta\gamma} = \delta_\alpha{}^\gamma \, . \tag{4}$$

In order to avoid an often unnecessary cluttering of indices, one can introduce the shorthand bracket notation

$$\lambda_{i\,\alpha} = |\lambda_i\rangle \, , \quad \tilde{\lambda}_i^{\dot{\alpha}} = |\tilde{\lambda}_i] \, , \qquad \lambda_i^\alpha = \langle\lambda_i| \, , \quad \tilde{\lambda}_{i\,\dot{\alpha}} = [\tilde{\lambda}_i| \, . \tag{5}$$

It is then possible to build Lorentz invariant contractions as follows

$$\begin{aligned} \langle i\, j\rangle &:= \langle\lambda_i\, \lambda_j\rangle := \lambda_i^\alpha \lambda_{j\,\alpha} = -\langle j\, i\rangle \, , \\ [i\, j] &:= [\tilde{\lambda}_i\, \tilde{\lambda}_j] := \tilde{\lambda}_{i\,\dot{\alpha}}\tilde{\lambda}_j^{\dot{\alpha}} = -[j\, i] \, , \end{aligned} \tag{6}$$

which will be called *angle* and *square brackets* respectively. Notice that given a triplet of spinors λ_i^α, λ_j^β, λ_k^γ any completely antisymmetrized combination of them gives zero, thus upon contraction with a single Levi-Civita tensor one gets the so called Schouten identity

$$\langle i\, j\rangle\langle k| + \langle j\, k\rangle\langle i| + \langle k\, i\rangle\langle j| = 0 \, , \tag{7}$$

where the analogue for the $\tilde{\lambda}$ spinors clearly also holds.

Massless momenta

Given a Lorentz four-vector p^μ, it can be shown that the corresponding finite-dimensional representation of $\mathrm{SL}(2,\mathbb{C})$ is $\left(\frac{1}{2},\frac{1}{2}\right)$. The map between the two representations can be explicitly realized through

$$\begin{aligned} p_{\alpha\dot{\alpha}} &:= p_\mu \sigma^\mu_{\alpha\dot{\alpha}} \, , \\ p^{\dot{\alpha}\alpha} &:= p_\mu \bar{\sigma}^{\mu\,\dot{\alpha}\alpha} \, , \end{aligned} \tag{8}$$

where given the Pauli matrices σ^i, we define $\sigma^\mu = \{\mathbb{1}_2, \vec{\sigma}\}$ and $\bar{\sigma}^\mu = \{\mathbb{1}_2, -\vec{\sigma}\}$, which satisfy the Clifford algebra

$$\{\sigma^\mu, \bar{\sigma}^\nu\} = 2\eta^{\mu\nu} \, . \tag{9}$$

If p^μ is the momentum associated to a particle of mass m, it is easy to see that

$$m^2 = p^2 = det(p_{\alpha\dot{\alpha}}) = \frac{1}{2} p^{\dot{\alpha}\alpha} p_{\alpha\dot{\alpha}} \, . \tag{10}$$

If we consider now a *massless particle*, the masslessness condition given by the vanishing of the determinant of $p_{\alpha\dot{\alpha}}$ is trivialized by setting

$$p_{\alpha\dot{\alpha}} = \lambda_\alpha \tilde{\lambda}_{\dot{\alpha}} \, , \tag{11}$$

due the antisymmetry of the angle and square brackets. Furthermore, notice that in momentum space the massless Dirac equation translates into

$$\begin{cases} p_{\alpha\dot{\alpha}}\lambda_p^\alpha = 0 \\ p_{\alpha\dot{\alpha}}\tilde{\lambda}_p^{\dot{\alpha}} = 0 \end{cases} \tag{12}$$

which is again automatically satisfied by writing the momentum as in eq. (11).

We can associate a vector p^μ to a momentum given in the spinor representation $p_{\alpha\dot{\alpha}}$ through the inverse map of eq. (8), given by

$$p^\mu = \frac{1}{2}\langle p\, \sigma^\mu\, p] = \frac{1}{2}[p\, \bar{\sigma}^\mu\, p\rangle \, . \tag{13}$$

When considering a momentum $k = -p$, it is easy to see from eq. (11) that we can write the spinors λ_k and $\tilde{\lambda}_k$ in terms of λ_p and $\tilde{\lambda}_p$, simply by defining

$$\lambda^{\alpha}_{-p} \equiv i\lambda^{\alpha}_{p} \,, \qquad \tilde{\lambda}^{\dot{\alpha}}_{-p} \equiv i\tilde{\lambda}^{\dot{\alpha}}_{p} \,. \tag{14}$$

The spinors λ and $\tilde{\lambda}$ are in general complex-valued, and the reality condition on the momentum p^{μ} eq. (13) translates into

$$\lambda = \tilde{\lambda}^{*} \tag{15}$$

up to an arbitrary phase which we set to 1. Usually one is rather lenient towards this condition, since it turns out that, both in some analytic as well as numeric settings (see Section 4), it is very convenient to allow for complex momenta[4]. Notice that there is no unique way of associating spinors to a given momentum p^{μ}, since the rescaling

$$\lambda \mapsto t\,\lambda \,, \qquad \tilde{\lambda} \mapsto \frac{1}{t}\tilde{\lambda} \tag{16}$$

clearly leaves the momentum invariant. Here $t \in \mathbb{C}$ in general whereas it is just a complex phase if eq. (15) applies. Equation (16) is called little group scaling, and it implements at the level of spinors those Lorentz transformations which preserve the given momentum.

When discussing processes involving particles of spin one, polarization vectors are usually required. In our conventions these can be written as

$$\varepsilon_{+}^{\dot{\alpha}\alpha}(p,r) = \sqrt{2}\,\frac{\tilde{\lambda}^{\dot{\alpha}}_{p}\lambda^{\alpha}_{r}}{\langle r\,p\rangle} = \sqrt{2}\,\frac{|p]\langle r|}{\langle r\,p\rangle} \,, \qquad \varepsilon_{-}^{\dot{\alpha}\alpha}(p,r) = \sqrt{2}\,\frac{\tilde{\lambda}^{\dot{\alpha}}_{r}\lambda^{\alpha}_{p}}{[p\,r]} = \sqrt{2}\,\frac{|r]\langle p|}{[p\,r]} \,, \tag{17}$$

or equivalently as vectors

$$\varepsilon_{+}^{\mu}(p,r) = \frac{1}{\sqrt{2}}\frac{\langle r\,\sigma^{\mu}p]}{\langle r\,p\rangle} \,, \qquad \varepsilon_{-}^{\mu}(p,r) = \frac{1}{\sqrt{2}}\frac{\langle p\,\sigma^{\mu}r]}{[p\,r]} \,. \tag{18}$$

It can be checked that these satisfy all the properties required for polarization vectors (in the Lorentz gauge):

$$\begin{aligned} \varepsilon_{+}(p,r)^{*} &= \varepsilon_{-}(p,r) \,, & p_{\mu}\,\varepsilon^{\mu}_{\pm}(p,r) &= 0 \,, \\ |\varepsilon_{\pm}(p,r)|^{2} &= -1 \,, & \varepsilon_{+}(p,r)\cdot\varepsilon_{-}(p,r)^{*} &= 0 \,. \end{aligned} \tag{19}$$

Given two massless momenta p_i and p_j, the associated Mandelstam invariant is defined as

$$s_{ij} := (p_i + p_j)^2 = 2p_i\cdot p_j = p_i^{\dot{\alpha}\alpha}p_{j\,\alpha\dot{\alpha}} = \langle i\,j\rangle[j\,i] \,, \tag{20}$$

where, using $|p\rangle[p| = p$ for massless momenta, we can rewrite

$$\langle i\,j\rangle[j\,i] = \langle i\,j\,i] \,. \tag{21}$$

We will refer to structures such as this as chains. Other examples of chains include

$$\begin{aligned} \langle q\,p_1\ldots p_{2n}\,k\rangle &= -\langle k\,p_{2n}\ldots p_1\,q\rangle \\ [q\,p_1\ldots p_{2n}\,k] &= -[k\,p_{2n}\ldots p_1\,q] \\ [q\,p_1\ldots p_{2n+1}\,k\rangle &= \langle k\,p_{2n+1}\ldots p_1\,q] \end{aligned} \tag{22}$$

where the p_i are not necessarily massless.

[4]Or alternatively to consider a different space-time signature [12], which similarly invalidates eq. (15).

Massive momenta

Considering massive particles, while eq. (8) still applies, eq. (11) does not hold anymore. It is nonetheless still possible to write the momentum directly in terms of the spinors λ and $\tilde{\lambda}$, by simply considering it as a linear combination of two massless momenta [2]

$$P^{\mu} := q^{\mu} + \frac{m^2}{2q \cdot k} k^{\mu} \quad \rightarrow \quad P_{\alpha\dot{\alpha}} = \lambda_{\alpha}\tilde{\lambda}_{\dot{\alpha}} + \frac{m^2}{\langle k\, q\rangle [q\, k]} \mu_{\alpha}\tilde{\mu}_{\dot{\alpha}} \,, \tag{23}$$

with $q^2, k^2 = 0$ and $q_{\alpha\dot{\alpha}} = \lambda_{\alpha}\tilde{\lambda}_{\dot{\alpha}}$ and $k_{\alpha\dot{\alpha}} = \mu_{\alpha}\tilde{\mu}_{\dot{\alpha}}$. Notice that counting the number of (real) degrees of freedom of the spinors in eq. (23) one finds three too many. These are spurious degrees of freedom corresponding to the little-group transformations of the massive momentum, which in four dimensions is implemented by an SO(3) subgroup of the Lorentz group. Just as the massless spinors defined up to (16) are equivalent, one can use these spurious degrees of freedom to set the spinors associated to k to arbitrary values. We will refer to μ and $\tilde{\mu}$ as reference spinors.

From eq. (23) one can define a pair of spinors associated to the massive momentum P

$$|P\rangle = |q\rangle + \frac{m}{[q\, k]} |k] \,, \qquad |P] = |q] + \frac{m}{\langle q\, k\rangle} |k\rangle \,, \tag{24}$$

which have very different properties compared to the massless spinors, for example if P and K are massive then in general $\langle PK] \neq 0$. The advantage of the decomposition of eq. (23) is that it allows to recycle much of the technology introduced for massless particles. The price to pay however is that certain symmetries of the amplitude are obscured, in particular the covariance of the amplitude under little group transformations of the massive momenta. To make such property manifest it is more convenient to introduce a new set of spinors λ^I_{α}, $\tilde{\lambda}^I_{\dot{\alpha}}$, carrying an additional SU(2) index[5], accounting explicitly for little group transformations [3]. We then have

$$P_{\alpha\dot{\alpha}} = \lambda^I_{\alpha}\tilde{\lambda}_{\dot{\alpha}I} = \epsilon_{IJ}\lambda^I_{\alpha}\tilde{\lambda}^J_{\dot{\alpha}} \,, \tag{25}$$

and the massive equivalent relation of 16 reads $\lambda^I_{\alpha} \rightarrow U^I_J \lambda^J_{\alpha}$ for $U^I_J \in \mathrm{SU}(2)$. It can be shown that the previously introduced spinors $|P\rangle$ and $|P]$ simply correspond to a specific choice of the λ^I_{α} and $\tilde{\lambda}^I_{\dot{\alpha}}$, where the little group index has been fixed. We live the implementation of this covariant formalism for future work.

2.2 Installation of the package

The package, which is available on the author's github[6], consists of one main script and two dependencies:

- `SpinorHelicity4D` is the main script and is the only one the user needs to call. This script contains all the functions dealing with analytic manipulations of the spinor expressions.
- `SpinorBuildingBlocks` is the dependency where the spinor and vector building blocks are defined, along with some related functions. No user interaction with this module is required.
- `SpinorNumerics` is the dependency where everything related to generation of numeric kinematics is defined, again no user interaction with this module is required.

[5]While for massless spinors the little group is just an U(1) phase, for massive particles in four dimensions is given by SU(2).

[6]Link to package repository: https://github.com/accettullihuber/SpinorHelicity4D.

In order to use the package, all three modules need to be installed, and this can be achieved in different ways, we provide here three examples:

1. (recommended) clone the git repository to your local machine and add the folder path to `Mathematica`'s `init.m` file.

   (a) Instructions on how to clone a repository can be found on the official github documentation. In order to manage your repositories easily we suggest to use a git GUI like for example "Github Desktop". Cloning the `SpinorHelicity4D` repository instead of simply downloading its content, will allow you to stay up to date with future releases or possible bug fixes. Once the repository has been cloned it will be available as a folder on your local machine at a path of your choosing, let's call it `M:\your\path\SpinorHelicity4D`.

   (b) In order to access the package within any `Mathematica` session without the need of specifying the package location every time, we recommend to add the line of code `AppendTo[$Path,"M:\your\path\SpinorHelicity4D"]` to the `init.m` file in the `Kernel` folder, whose location can be obtained by typing `$UserBaseDirectory` or `$BaseDirectory` in a `Mathematica` notebook.

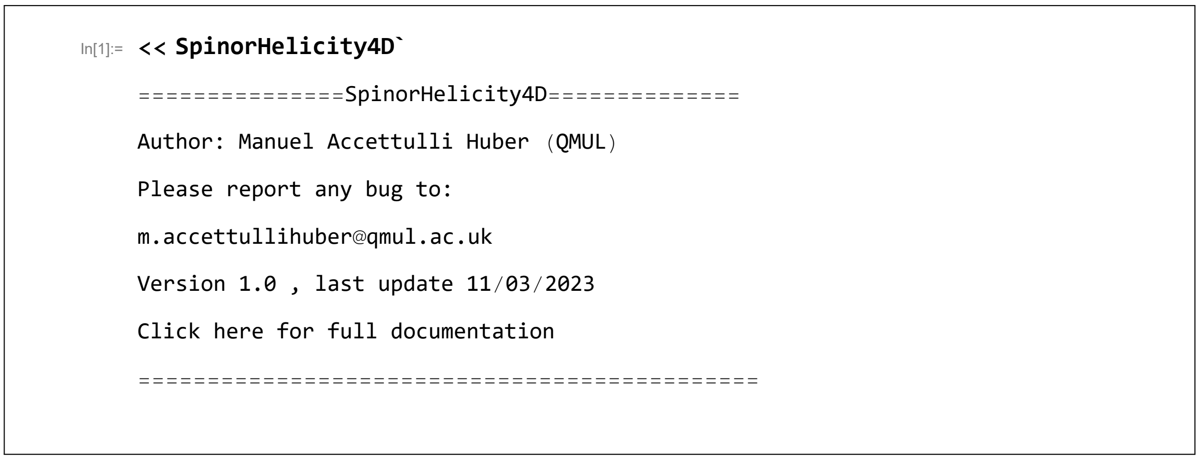
2. Download the repository's content and add the folder path to `Mathematica`'s `init.m` file:

   (a) Download the repository and save the three `.wl` file components to a folder at a location of your choosing, let's call it `M:\your\path\SpinorHelicity4D`. Opting for this method will require you to download the pacakge from scratch in order to get access to updates and new releases.

   (b) Edit the `init.m` file as described in 1.b above.

3. Download the repository's content and copy the three `.wl` files into the `Applications` folder of your `Mathematica` installation, whose location is again found at `$UserBaseDirectory` and `$BaseDirectory`. The advantage of this approach is that no editing of the `$Path` variable is required, since by default the `Applications` folder is where `Mathematica` looks for packages. The disadvantage is that in order to get access to updates or future releases will require you to download the package from scratch and repeat the procedure.

Once any of the above procedures was carried out successfully, the package can be accessed by simply typing `<<SpinorHelicity4D`, as shown below.



```
In[1]:= << SpinorHelicity4D`

        ================SpinorHelicity4D==============

        Author: Manuel Accettulli Huber (QMUL)

        Please report any bug to:

        m.accettullihuber@qmul.ac.uk

        Version 1.0 , last update 11/03/2023

        Click here for full documentation

        ================================================
```

In order to access a list of the functions in the package, type `?PackageName*` replacing PackageName with the name of the appropriate module, depending on whether you look for analytic, numeric or building block functions.

2.3 Expression input

There are four different but completely equivalent ways for inputting expressions.

- Through direct input of the function name.

```
In[2]:= SpinorAngleBracket[1, 2]
Out[2]= ⟨1 2⟩

In[3]:= SpinorSquareBracket[k1, k2]
Out[3]= [k1 k2]
```

- Through an input palette, consisting of a small external notebook with a collection of control buttons. This palette will appear automatically when the package is loaded, but it can also be accessed by typing `SpinorPalette[]`. Clicking on a given button will produce the corresponding object in the notebook, one can then navigate through the placeholders with the TAB keyboard command replacing them with the desired momenta/indices.
- Through keyboard shortcuts associated to every building block, consisting of a short string which, when escaped, produces the desired input. A list of all the available shortcuts can be accessed by typing `$Shortcuts`.
- Through customized definitions of the functions' names with `AssignNames`. For example, an accustomed S@M [5] user could define the angle and square brackets as `Spaa` and `Spbb` respectively. This is achieved as shown below.

```
In[5]:= (*Relabel angle and square brackets*)
        AssignNames[Spaa[x_, y_] := SpinorAngleBracket[x, y],
         Spbb[x_, y_] := SpinorSquareBracket[x, y]]

In[6]:= Spaa[1, 2]
        Spbb[k1, k2]
Out[6]= ⟨1 2⟩

Out[7]= [k1 k2]

In[8]:= (*Display all the given definitions*)
        AssignNames[] // MatrixForm
Out[8]//MatrixForm=
        ( Hold[Spaa[x_, y_] :→ ⟨x y⟩] )
        ( Hold[Spbb[x_, y_] :→ [x y]] )
```

The difference between using `AssignNames` and directly writing the definitions in the notebook is that the former creates a file [7], where the definitions are stored, and then loaded along with the package every time this is called. The definitions become thus permanent.

```
In[∘]:=  (*Upon quitting the kernel and reloading
          the package the definitions are still available*)
         Quit[]

In[1]:=  << SpinorHelicity4D`;

In[2]:=  Spaa[k1, k2]

Out[2]=  ⟨k1 k2⟩
```

The definitions can be cleared using `ClearNames`, which can be called either with one or more arguments to clear the desired functions' definitions, or without arguments to clear them all.

```
In[3]:=  (*Clear only Spaa*)
         ClearNames[Spaa]

In[4]:=  Spaa[1, 2]

Out[4]=  Spaa[1, 2]

In[5]:=  (*Clear all the definitions*)
         ClearNames[]

In[6]:=  AssignNames[]

Out[6]=  No available definitions.
```

2.4 Declaration of momentum labels

Many of the functions defined in the package posses definite properties with respect to their arguments. For example the scalar product, which we call `mp`, is clearly linear with respect to four-vectors, thus given three momenta p, q, k and a constant c one expects

$$p \cdot (q + c\,k) = p \cdot q + c\,p \cdot k\ . \tag{26}$$

In order for this and other similar properties to be applied correctly in Mathematica, it is necessary to first declare which symbols have to be treated as particle/momentum labels. This is achieved through `DeclareMom`:

[7]This file is located in the same folder as the package.

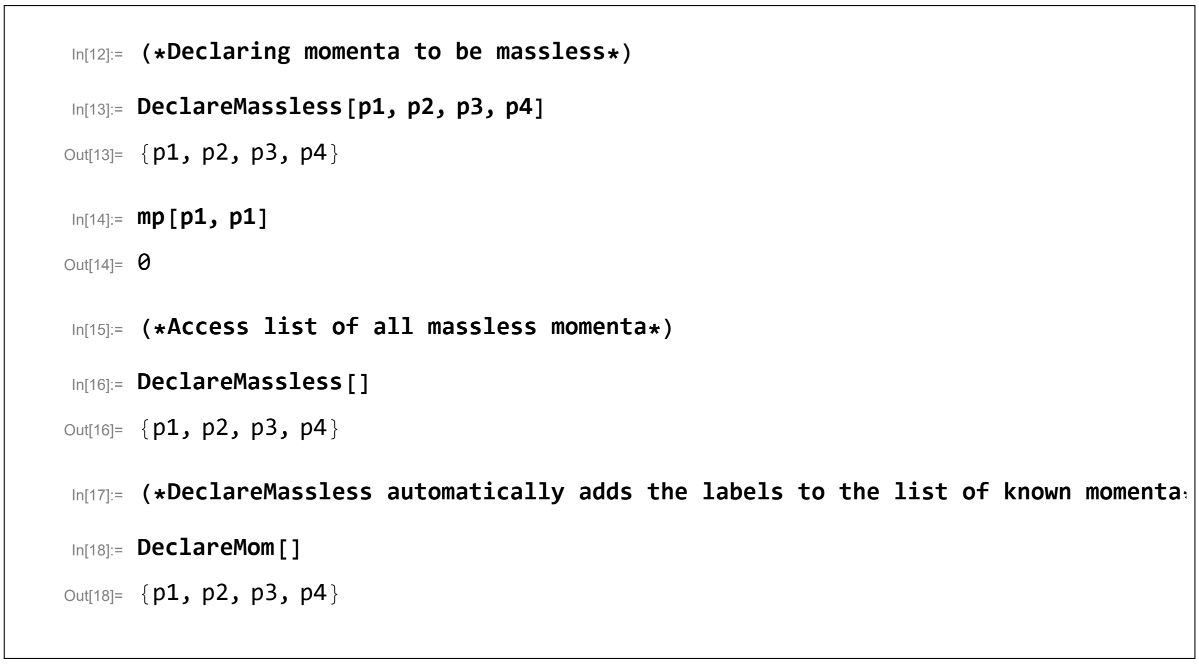
```
In[2]:=  mp[p1, p2 + c * p3]
Out[2]=  (p1 · p2 + c p3)

In[3]:=  (*Declare a set of symbols to be momentum labels*)

In[4]:=  DeclareMom[p1, p2, p3, p4]
Out[4]=  {p1, p2, p3, p4}

In[5]:=  mp[p1, p2 + c * p3]
Out[5]=  (p1 · p2) + c (p1 · p3)
```

From here on, whenever momentum declaration is required to obtain the displayed output, we will use the light gray background for the associated input. It is worth pointing out that declaration of momentum labels is only necessary if access to the full properties of functions and objects is needed, thus the basic usage of the package retains an intuitive nature. Notice also that it is not possible to declare numbers as momentum labels. The list of so far declared labels can be accessed by calling `DeclareMom` without any arguments, and labels can be removed from this list with `UndeclareMom`.

```
In[6]:=  (*Access the list of declared labels*)

In[7]:=  DeclareMom[]
Out[7]=  {p1, p2, p3, p4}

In[8]:=  (*Remove specific labels from the list*)

In[9]:=  UndeclareMom[p3, p4]
Out[9]=  {p1, p2}

In[10]:= (*Clear the list of labels*)

In[11]:= UndeclareMom[]
Out[11]= { }
```

2.5 Declaration of massless momenta

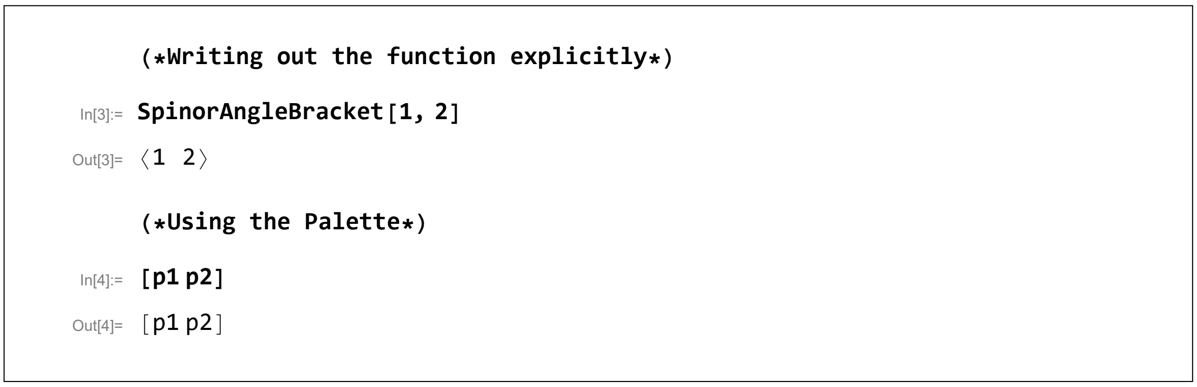
In general, since the package is designed in order to be able to handle both massless and massive momenta, no assumption is made on momentum labels. Thus, in order to access the full range of simplifications occurring due to a momentum being massless, one has to declare it as such through DeclareMassless. To remove labels from the list of massless momenta use `UndeclareMassless`, and to clear them all `UndeclareMassless` without any arguments, just as in the case of `UndeclareMom`.



```
In[12]:= (*Declaring momenta to be massless*)

In[13]:= DeclareMassless[p1, p2, p3, p4]
Out[13]= {p1, p2, p3, p4}

In[14]:= mp[p1, p1]
Out[14]= 0

In[15]:= (*Access list of all massless momenta*)

In[16]:= DeclareMassless[]
Out[16]= {p1, p2, p3, p4}

In[17]:= (*DeclareMassless automatically adds the labels to the list of known momenta

In[18]:= DeclareMom[]
Out[18]= {p1, p2, p3, p4}
```

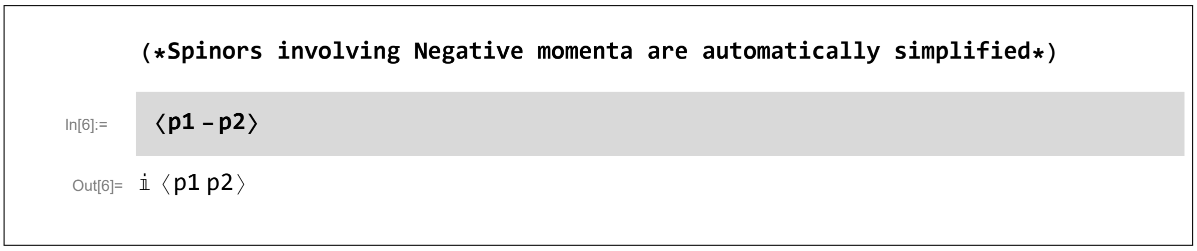
2.6 Angle and square brackets

Angle and square brackets are entered through the associated functions `SpinorAngleBracket` and `SpinorSquareBracket` respectively, or equivalently using the palette or the shortcuts.



```
         (*Writing out the function explicitly*)

In[3]:=  SpinorAngleBracket[1, 2]
Out[3]=  ⟨1  2⟩

         (*Using the Palette*)

In[4]:=  [p1 p2]
Out[4]=  [p1 p2]
```

Here it is important to stress that even when displayed as $\langle 1\,2\rangle$, this object is still recognised as the function `SpinorAngleBracket[1,2]`, and thus it can for example be safely copy-pasted or otherwise manipulated. Once momentum labels are defined, the convention on negative momenta eq. (14) is automatically applied.



```
         (*Spinors involving Negative momenta are automatically simplified*)

In[6]:=  ⟨p1 -p2⟩

Out[6]=  𝕚 ⟨p1 p2⟩
```

The arguments of the spinor brackets are ordered according to Mathematica's canonical ordering, taking into account antisymmetry.

```
        (*Ordering of the arguments and antisymmetry*)

In[7]:= [p2 p1]

Out[7]= - [p1 p2]

In[8]:= [p1 p1]

Out[8]= 0
```

Spinors are generally not linear with respect to their momentum argument, in other words

$$\lambda^{\alpha}_{p+q} \neq \lambda^{\alpha}_{p} + \lambda^{\alpha}_{q} \, , \tag{27}$$

however when typing into the computer it is usually very convenient to apply the following definition:

$$\langle p_1 \, (p_2 + c \, p_3) \rangle \equiv \langle p_1 \, p_2 \rangle + c \langle p_1 \, p_3 \rangle \, . \tag{28}$$

We remark that this is purely a convention that we decided to adopt for faster typing and has nothing to do with algebraic properties of the spinors. Thus, upon declaring momentum labels, one can write

```
        (*Having declared momentum labels*)

In[5]:= ⟨p1 (p2 + c * p3)⟩

Out[5]= ⟨p1 p2⟩ + c ⟨p1 p3⟩
```

One should be careful about the delicate interplay between the above defined "linearity convention" and negative momenta.

2.7 Chains

All four of the possible invariants obtained by contracting two spinors with an arbitrary number of momenta, to which we will refer as chains, are inputted through the single function `Chain`. This requires as input the type and labels of the spinors which start and end the chain, and a list of the internal momenta. For example:

$$\langle q_1 \, p_1 \cdots p_{2n} \, q_2 \rangle \quad \mapsto \quad \texttt{Chain[\$angle,q1,\{p1,...,p2n\},q2,\$angle]} \, , \tag{29}$$

and similarly for $[\ldots]$, $\langle \ldots]$ and $[\ldots\rangle$.

```
In[9]:= Chain[$square, p1, {p2}, p3, $angle]

Out[9]= [p1 {p2} p3⟩
```

Clearly chains are linear with respect to internal momenta, and the external spinors follow the same convention for negative momenta (eq. (14)) as the angle and square brackets do.

```
In[10]:=  [p1 {p2 + p4} p3⟩

Out[10]=  [p1 {p2} p3⟩ + [p1 {p4} p3⟩

In[11]:=  [p1 {p2} -p3⟩

Out[11]=  ⅈ [p1 {p2} p3⟩
```

Once again, for faster typing of expressions we extend the definition eq. (28) to the external spinors of the chains as well. This must not be confused with the genuine linearity of the internal momenta.

```
In[12]:=  [p1 {p2} p3 + c * p4⟩

Out[12]=  [p1 {p2} p3⟩ + c [p1 {p2} p4⟩
```

2.8 Uncontracted spinors

Despite the fact that is generally more convenient to work with Lorentz-invariant quantities such as the spinor brackets and the Mandelstam invariants, in some instances the use of uncontracted spinors might be required. The direct input of such objects is accomplished by the functions `SpinorUndot` and `SpinorDot`, which require to specify a spinor label, the spinor type (λ or reference spinor μ) and a spinor index along with its position. The spinor type is specified through the use of the protected symbols `$lam` and `$mu`, and the position specification of the Lorentz index by setting to `Null` the absent upper or lower index.

```
In[25]:= SpinorUndot[p1][$lam][Null][a]

Out[25]= λa[p1]

In[26]:= SpinorDot[p1][$mu][a][Null]

Out[26]= μ̃a[p1]
```

We follow the convention that $\lambda^{\alpha}_{-p} \equiv i\lambda^{\alpha}_{p}$ and $\tilde{\lambda}^{\dot{\alpha}}_{-p} \equiv i\tilde{\lambda}^{\dot{\alpha}}_{p}$ which is automatically implemented

```
         (*Minus signs are automathically taken care of*)

In[27]:=  λ^a[-p1]

Out[27]=  i λ^a[p1]

In[28]:=  λ̃^a[-p1]

Out[28]=  i λ̃^a[p1]
```

Furthermore, just as in the case of the angle and square brackets a pseudo-linearity convention on the momentum labels is in place, allowing for faster expression input (see eq. (27) and related discussion).

```
In[33]:=  λ^a[p1 + c * p2]

Out[33]=  λ^a[p1] + c λ^a[p2]
```

Spinor indices can be raised and lowered through the Levi-Civita tensor:

```
In[9]:=   ϵ^{a b} λ_b[p1]
Out[9]=   λ^a[p1]

In[10]:=  ϵ_{a b} λ̃^b[p1]
Out[10]=  λ̃_a[p1]
```

In a product of spinors, if possible spinor contractions into invariants are performed:

```
         (*Index contraction is automatically performed*)

In[29]:=  λ^a[p1] λ_a[p2]
Out[29]=  ⟨p1 p2⟩

In[30]:=  λ̃^a[p1] λ̃_a[p2]
Out[30]=  - [p1 p2]

In[31]:=  λ^a[p1] (λ_a[p2] + λ_a[p3]) // Expand
Out[31]=  ⟨p1 p2⟩ + ⟨p1 p3⟩
```

Notice also that dotted and undotted indices, belonging to different representations, are not mixed even if the same label is used:

```
        (*Dotted and undotted indices are not mixed*)

In[32]:= λ^a[p1] λ̃_a[p2] λ_a[p3] λ̃^a[p4]
Out[32]= ⟨p1 p3⟩ [p2 p4]
```

Finally, there are a set of objects called `SpinorUndotBare` and `SpinorDotBare` which are not physically meaningful objects but are merely required by the function `SpinorReplace`, see Section 3.8.

2.9 Reference spinors

As has already been mentioned in Section 2.1, it is possible to describe massive particles through the massless spinor helicity formalism, at the price of introducing redundant degrees of freedom in form of reference spinors, see eq. (23) which we display here again for convenience:

$$P_{\alpha\dot{\alpha}} = \lambda_\alpha \tilde{\lambda}_{\dot{\alpha}} + \frac{m^2}{\langle k\, q\rangle [q\, k]} \mu_\alpha \tilde{\mu}_{\dot{\alpha}} \,, \tag{30}$$

where $P^2 = m^2$ and q, k are massless with $q_{\alpha\dot{\alpha}} = \lambda_\alpha \tilde{\lambda}_{\dot{\alpha}}$ and $k_{\alpha\dot{\alpha}} = \mu_\alpha \tilde{\mu}_{\dot{\alpha}}$. We will refer to k as the reference spinor for the massive particle P. Since they represent partially redundant degrees of freedom, reference momenta/spinors get special treatment, both in the uncontracted form of Section 2.8 as well as when they appear inside spinor invariants. Doing so allows to simplify expressions by setting the references to convenient values as well as retrace invariants involving massive particles from their spinor components. The way reference spinors are distinguished is by using the μ label instead of λ in the uncontracted form, and by an overbar in the contracted form.

```
In[5]:= SpinorUndot[p2][$mu][Null][a]
Out[5]= μ_a[p2]

In[6]:= % * λ^a[p1]
Out[6]= ⟨p1 p2̅⟩
```

Spinor invariants containing reference spinors can be defined using the function `obar`:

```
In[7]:= ⟨p1 obar[p2]⟩
Out[7]= ⟨p1 p2̅⟩
```

`obar` is linear in declared momenta:

```
In[8]:=  ⟨p1 obar[p2 + c * p3]⟩
Out[8]=  ⟨p1 p2̄⟩ + c ⟨p1 p3̄⟩
```

Reference spinors are mainly used when dealing with massive states, notice however that also for massless particles reference spinors might come up, for example in the polarizations, and these can as well be described using the μ spinors. Since the intrinsic purpose of these built-in objects is the description of massive particles, some care is needed when doing so. In particular, when numerical kinematics is generated the reference spinors of massless particles are automatically initialised to `Null` and associated masses to zero so that eq. (23) reduces to $P^{\dot{\alpha}\alpha} = \tilde{\lambda}^{\dot{\alpha}}\lambda^{\alpha}$ as it should.

2.10 Scalar product and Mandelstam invariants

Scalar products are represented by the protected symbol `mp`, which has attribute orderless meaning that of course $p_2 \cdot p_1 = p_1 \cdot p_2$, and is also linear in declared momenta:

```
In[11]:= mp[p2, p1]
Out[11]= (p1 · p2)

In[12]:= mp[p1, p2 + c * p3]
Out[12]= (p1 · p2) + c (p1 · p3)
```

Similarly also the Mandelstam invariants, labelled by `S`, are orderless but do not present additional properties.

```
In[13]:= S[p2, p4, p1, p3]
Out[13]= S[p1, p2, p3, p4]
```

The following shorthand notation is available:

```
         (*mp[p] is equivalent to mp[p,p]*)
In[32]:= mp[p]
Out[32]= (p)²
```

When a momentum is declared as massless the associated scalar product is set to zero.

```
In[33]:= DeclareMassless[p];

In[34]:= mp[p]

Out[34]= 0
```

In many instances it might be useful to set the expressions of some of the invariants to given values, this can be achieved with `SetInvariants`, which takes as input a list of replacements for scalar products and Mandelstam invariants, and whose output is a list of the invariants whose values have been fixed. For example consider setting $s_{12} \equiv s$ and $p_1 \cdot p_3 \equiv t/2$:

```
         (*Set Lorentz invariants to given values*)

In[35]:= SetInvariants[S[p1, p2] → s, mp[p1, p3] → t/2]

Out[35]= {mp[p, p] :→ 0, mp[p1, p3] :→ t/2, S[p, p] :→ 0, S[p1, p2] :→ s}

In[36]:= mp[p1, p3]

Out[36]= t/2
```

Clearly, if p_i and p_j are massless $s_{ij} = 2p_i \cdot p_j$, which is account for by `SetInvariants`:

```
         (*If the momenta are set to massless mp and S are related*)

In[37]:= DeclareMassless[p1, p2, p3];

In[38]:= SetInvariants[S[p1, p2] → s, mp[p1, p3] → t/2];

In[39]:= S[p1, p3]

Out[39]= t

In[43]:= mp[p1, p2]

Out[43]= s/2
```

If `SetInvariants` is called without arguments it simply displays the list of already fixed values, also invariants are cleared with `ClearInvariants`, which takes as input the invariants to be cleared. If no argument is given to `ClearInvariants` all the invariants are cleared.

```
        (*Invariants are cleared with ClearInvariants*)

In[41]:= ClearInvariants[S[p1, p3]];

In[42]:= mp[p1, p3]
Out[42]= (p1 · p3)
```

Notice that `SetInvariants` does not require the invariants to be set to constants:

```
In[48]:= SetInvariants[S[p3, p4] → S[p1, p2]];

In[49]:= S[p3, p4]
Out[49]= S[p1, p2]
```

2.11 Vector quantities

The package also features a limited number of vector quantities which often appear in amplitudes calculations. These include uncontracted momentum vectors, polarization vectors, the flat space metric tensor η, the Kronecker delta and the vector of Pauli matrices[8]. The polarizations also present an input form stripped of the Lorentz index called `PolarBare`, which is required when the polarizations appear inside scalar products for example.

```
        (*Momentum vectors*)

In[2]:= {Mom[a][μ][Null], Mom[b][Null][ν]}
Out[2]= {a^μ, b_ν}

        (*Polarization vectors and bare polarization*)

In[3]:= {Polar[p, q][μ][Null], Polar[k, l][Null][ν], PolarBare[m, n]}
Out[3]= {ε^μ[p, q], ε_ν[k, l], ε[m, n]}

        (*Metric tensor and Kronecker delta*)

In[4]:= {Eta[μ, ν][$up], Eta[μ, ν][$down], DeltaVec[μ, ν]}
Out[4]= {η^μν, η_μν, δ^μ_ν}

        (*Pauli matrix place holder, needed for VecToSpinors*)

In[5]:= {PauliSH[μ][$up], PauliSH[μ][$down]}
Out[5]= {σ^μ, σ_μ}
```

All the vector quantities enjoy automatic contraction properties among each other.

[8]These are only required as a placeholders inside Chain expressions.

(*Contraction properties*)

In[6]:= $\eta^{\mu\,\nu}\, p_{\mu}\, \varepsilon_{\nu}[q, k]$

Out[6]= $(p \cdot \varepsilon[q, k])$

In[7]:= $\eta^{\mu\,\nu}\, \varepsilon_{\mu}[p, l]\, \varepsilon_{\nu}[q, k]$

Out[7]= $(\varepsilon[p, l] \cdot \varepsilon[q, k])$

In[8]:= $\delta^{\mu}_{\nu}\, p^{\nu}\, q_{\mu}$

Out[8]= $(p \cdot q)$

In[9]:= $p^{\nu}\, \varepsilon_{\nu}[p, k]$

Out[9]= 0

These contraction properties include the trace of the Kronecker delta, whose value can be modified through the option `$DeltaVecDim` of `DeltaVec`.

(*Trace of DeltaVec is set to 4 by default*)

In[10]:= $\{\delta^{\mu}_{\mu},\ \eta^{\mu\,\nu}\, \eta_{\mu\,\nu}\}$

Out[10]= $\{4, 4\}$

(*Space-time dimension can be changed through $DeltaVecDim*)

```
In[11]:= SetOptions[DeltaVec, $DeltaVecDim → dim]

Out[11]= {$DeltaVecDim → dim}
```

In[12]:= $\{\delta^{\mu}_{\mu},\ \eta^{\mu\,\nu}\, \eta_{\mu\,\nu}\}$

Out[12]= $\{dim, dim\}$

2.12 Clearing everything, NewProcess

It is possible to reset all the given definitions relating to the building blocks presented in this whole section, without the need of quitting the `Mathematica` kernel. This is achieved through `NewProcess` which clears the values of the invariants, the list of momenta declared as massless and the list of labels declared to be momenta.

```
In[50]:= DeclareMassless[]

Out[50]= {p, p1, p2, p3}

In[51]:= DeclareMom[]

Out[51]= {p, p1, p2, p3, p4}

In[53]:= NewProcess[]

In[54]:= DeclareMassless[]

Out[54]= { }

In[55]:= DeclareMom[]

Out[55]= { }
```

3 Manipulation of analytic expressions

In this section we describe the manipulation of given analytic expressions defined in terms of the building blocks introduced so far.

3.1 ToChain, contraction of spinor products into chains

When dealing with the product of multiple angle and square brackets it is possible to contract them into continuos chains by means of the replacement

$$\cdots|p\rangle[p|\cdots \mapsto \cdots|p|\cdots\ , \tag{31}$$

this operation is achieved through `ToChain`. The contraction of spinor products into chains is not unique, consider for example the following

$$\langle 1\,2\rangle[2\,5][2\,3]\langle 3\,4\rangle = \begin{cases}\langle 1\,2\,5][2\,3\,4\rangle\\ \langle 1\,2\,3\,4\rangle[2\,5]\end{cases} \tag{32}$$

both are perfectly valid contractions but depending on the situation one or the other may be preferred. `ToChain` allows for the option `ChainSelection` which provides some degree of control over how chain contractions are performed:

- "MostTraces" selects the contraction which maximises the number of Dirac traces in the expression, in other words the number of chains of the type $\langle p\cdots p]$
- "LongestChain" tries to find the longest continuos contraction into a chain
- "ShortestChain" tries to find the shortest contractions where still as many spinor brackets as possible are contracted into a chain[9]
- "RandomChain" is the default argument and provides a contraction based on no specific criterion.

[9]Of course without the latter requirement the shortest contraction would simply be no contraction at all.

In order for any of the first three criteria to be applied it is first necessary to find all possible contractions and then select the appropriate one. The price to pay comes in terms of computational efficiency: since of course the more spinor invariants there are the more contractions are possible, the more computationally expensive it becomes to find all the chain contractions. There are however situations in which one might only be interested in getting one possible contraction but not be concerned about which one. This is the case for example when one considers a helicity-neutral structure, such as those often appearing in generalised unitarity calculations upon factoring out an appropriate overall helicity factor. In these situations we recommend the use of the default option "RandomChain" which is based on an algorithm designed to simply return one possible contraction without classifying them all, and is thus much faster than the other criteria.

```
In[23]:= test = ⟨1 2⟩ [2 3] ⟨3 4⟩ [4 1] [4 5] ⟨5 6⟩ [6 7] ⟨7 4⟩ ⟨7 8⟩ [8 9] [4 3];

         (*Deafult option RandomChain, a random contraction is returned*)

In[24]:= test // ToChain

Out[24]= -⟨7 {8} 9] [3 {2, 1, 4, 7, 6, 5, 4} 3⟩ ⟨3 4⟩

         (*Contraction with two explicit Dirac Traces*)

In[25]:= ToChain[test, ChainSelection -> "MostTraces"]

Out[25]= -⟨1 {2, 3, 4} 1] ⟨5 {6, 7, 4} 5] ⟨7 {8} 9] [3 4]

         (*Shortest chains where as many invaraints as possible are contracted*)

In[26]:= ToChain[test, ChainSelection -> "ShortestChain"]

Out[26]= -⟨1 {2, 3, 4} 1] ⟨7 {8} 9] [3 {4, 7, 6, 5} 4]

         (*Longest possible chain contraction*)

In[27]:= ToChain[test, ChainSelection -> "LongestChain"]

Out[27]= -⟨7 {4, 1, 2, 3, 4, 5, 6, 7, 8} 9] [3 4]

         (*Timing comparison: RandomChain vs MostTraces*)

In[30]:= ToChain[test, ChainSelection -> "RandomChain"] // AbsoluteTiming

Out[30]= {0.0039978, -⟨7 {8} 9] [3 {2, 1, 4, 7, 6, 5, 4} 3⟩ ⟨3 4⟩}

In[31]:= ToChain[test, ChainSelection -> "MostTraces"] // AbsoluteTiming

Out[31]= {0.141822, -⟨1 {2, 3, 4} 1] ⟨5 {6, 7, 4} 5] ⟨7 {8} 9] [3 4]}
```

3.2 ChainToSpinor

The function `ChainToSpinor` performs the inverse operation of `ToChain` by splitting chains into spinor brackets. Notice that, while the contraction into chains is always possible, splitting a chain into spinor brackets is only allowed if the involved particles are massless, which needs to be declared through `DeclareMassless`.

```
      (*If momenta are not declared massless nothing happens*)

In[∘]:= ⟨1 {2, 3, 4} 5] // ChainToSpinor

Out[∘]= ⟨1 {2, 3, 4} 5]

      (*Chains including massless spinors can be split into brackets*)

In[∘]:= DeclareMassless[3, 4]

Out[∘]= {3, 4}

In[∘]:= ⟨1 {2, 3, 4} 5] // ChainToSpinor

Out[∘]= ⟨1 {2} 3] ⟨3 4⟩ [4 5]
```

3.3 ChainMomentumCon

The function `ChainMomentumCon` allows to apply momentum conservation identities at the chain level without affecting uncontracted spinor brackets. If momentum labels are declared, linearity is as always applied.

```
In[∘]:= DeclareMom[p1, p2, p3, p4, p5, p6, p7]

Out[∘]= {p1, p2, p3, p4, p5, p6, p7}

In[∘]:= ChainMomentumCon[⟨p1 {p2, p3, p5} p3], p3 → p1 + 3 p2]

Out[∘]= ⟨p1 {p2, p1, p5} p3] + 3 ⟨p1 {p2, p2, p5} p3]

      (*Multiple simultaneous replacement are allowed*)

In[∘]:= ChainMomentumCon[⟨p1 {p2, p3, p5} p3], {p3 → p1 + 3 p2, p5 → p7}]

Out[∘]= ⟨p1 {p2, p1, p7} p3] + 3 ⟨p1 {p2, p2, p7} p3]
```

In the particular case in which the momentum which we want to replace appears as the extrema of a Dirac trace chain, the chain is automatically rearranged so to allow for the application of momentum conservation. The rearrangement of the chain into a different Dirac trace is only possible if at least one of the momenta inside the chain is massless.

```
        (*If the momentum is an extremum of a Dirac trace
         but no massless momentum is present inside the chain:*)

In[∘]:= ChainMomentumCon[⟨p1 {p2, p3, p5} p1], p1 → p3 + p5]

Out[∘]= ⟨p1 {p2, p3, p5} p1]

        (*With a massless momentum present the chain is automatically rearranged*)

In[∘]:= DeclareMassless[p2];

In[∘]:= ChainMomentumCon[⟨p1 {p2, p3, p5} p1], p1 → p3 + p6]

Out[∘]= ⟨p2 {p3, p5, p3} p2] + ⟨p2 {p3, p5, p6} p2]
```

3.4 ChainSort

The function `ChainSort` allows to sort the momenta present in contracted chains through the use of the Clifford algebra identity of the σ matrices $\{\sigma^\mu, \bar{\sigma}^\nu\} = 2\eta^{\mu\nu}$. This procedure can prove quite useful in simplification of expressions because it allows to reduce all chains to a common canonically ordered form. Notice that the sorting procedure is customizable, if an optional argument consisting of a list of labelles is passed to the function, this is used to sort the chains instead of the default ordering provided by `Mathematica`.

```
In[2]:= ChainSort[⟨p1 {p3, p2, p5} p1]]

Out[2]= -⟨p1 {p2, p3, p5} p1] + 2 ⟨p1 {p5} p1] (p2 · p3)

In[3]:= ChainSort[⟨p1 {p3, p2, p5, p6, p4} p1]]

Out[3]= -⟨p1 {p2, p3, p4, p5, p6} p1] +
         2 ⟨p1 {p2, p3, p6} p1] (p4 · p5) - 2 ⟨p1 {p2, p3, p5} p1] (p4 · p6) +
         2 (p2 · p3) (⟨p1 {p4, p5, p6} p1] - 2 ⟨p1 {p6} p1] (p4 · p5) + 2 ⟨p1 {p5} p1] (p4 · p6)

        (*Customise the ordering by specifying an ordered list of labels*)

In[4]:= ChainSort[⟨p1 {p2, p3, p2, p1} p4⟩, {p1, p3, p2}]

Out[4]= ⟨p1 {p1, p3, p2, p2} p4⟩ - 2 ⟨p1 {p2, p2} p4⟩ (p1 · p3) +
         2 (p2 · p3) (-⟨p1 {p1, p2} p4⟩ + 2 (p1 · p2) ⟨p1 p4⟩)

        (*It is not necessary for all the labels to have a specific order,
        for example we could specify only the order of two labels*)

In[5]:= ChainSort[⟨p1 {p2, p3, p2, p1} p4⟩, {p1, p3}]

Out[5]= ⟨p1 {p2, p1, p3, p2} p4⟩ + 2 ⟨p1 {p2, p3} p4⟩ (p1 · p2) - 2 ⟨p1 {p2, p2} p4⟩ (p1 · p3)
```

The extrema of the chain are also included in the sorting, and in the case of a Dirac trace (where the extrema can often be arbitrarily chosen) these can also mix with internal massless momentum labels.

```
         (*Extrema are also accounted for in the ordering*)
In[6]:=  ChainSort[⟨p4 {p3, p2} p1⟩]
Out[6]=  -⟨p1 {p2, p3} p4⟩

In[7]:=  ChainSort[⟨p4 {p2, p3, p2} p1], {p1, p3, p4, p2}]
Out[7]=  -[p1 {p3, p2, p2} p4⟩ + 2 [p1 {p2} p4⟩ (p2 · p3)

         (*If we have Dirac traces then also the extrama of the chain can
          be reordered. For this to be possible there must be an internal
          massless momentum to replace the external one*)
In[8]:=  ChainSort[⟨p4 {p2, p1, p3} p4]]
Out[8]=  -⟨p4 {p1, p2, p3} p4] + 2 ⟨p4 {p3} p4] (p1 · p2)

In[9]:=  DeclareMassless[p1, p4]
Out[9]=  {p1, p4}

In[10]:= ChainSort[Chain[$angle, p4, {p2, p1, p3}, p4, $square]]
Out[10]= -⟨p1 {p4, p2, p3} p1] + 2 ⟨p4 {p3} p4] (p1 · p2) +
          2 ⟨p1 {p3} p1] (p4 · p2) - 2 ⟨p1 {p2} p1] (p4 · p3)
```

3.5 ChainSimplify

While sorting the momenta inside chains may lead to simplifications, a more powerfull simplification algorithm is available to the user in form of `ChainSimplify`. This function, which takes as single input the expression to be simplified, makes use once again of $\{\sigma^\mu, \bar{\sigma}^\nu\} = 2\eta^{\mu\nu}$ to reshuffle momenta in such a way that chains in the input reduce to shorter chains and scalar products. Take as an example

$$\langle q\, p\, q\, k\rangle = 2p\cdot q\,\langle q\, k\rangle - \langle q\, q\, p\, k\rangle = 2p\cdot q\,\langle q\, k\rangle\ . \tag{33}$$

We stress that even simplifications which do not require reshuffling of the chains are performed by `ChainSimplify`, this includes for example the trivial $\langle q\, q\, p\, k\rangle = 0$.

```
In[7]:=  [q {q, p} k] // ChainSimplify

Out[7]=  0

In[6]:=  ⟨q {p, q} k⟩ // ChainSimplify

Out[6]=  -2 (p · q) ⟨k q⟩

In[9]:=  ChainSimplify[⟨p1 {p2, p3, p2, p1} p4⟩]

Out[9]=  -2 (p1 · p3) (p2)^2 ⟨p1 p4⟩ + 4 (p1 · p2) (p2 · p3) ⟨p1 p4⟩
```

Unless it is explicitly requested through the option `ReduceBySorting` momenta in the chains are not sorted, since a priori sorting does not guarantee a simpler final expression. By simply setting `ReduceBySorting -> True` standard ordering of the momenta is applied, while specifying a list allows to specify a custom ordering.

```
         (*Unless explicitly specified momenta are not sorted*)

In[15]:= ChainSimplify[[p1 {p2, p3} p4] + [p1 {p3, p2} p4]]

Out[15]= [p1 {p2, p3} p4] + [p1 {p3, p2} p4]

In[16]:= ChainSimplify[[p1 {p2, p3} p4] + [p1 {p3, p2} p4], ReduceBySorting → True]

Out[16]= 2 (p2 · p3) [p1 p4]

         (*Custom ordering is also available*)

In[19]:= ChainSimplify[[p1 {p2, p3} p4] - 2 (p2 · p3) [p1 p4], ReduceBySorting → True]

Out[19]= [p1 {p2, p3} p4] - 2 (p2 · p3) [p1 p4]

In[20]:= ChainSimplify[[p1 {p2, p3} p4] - 2 (p2 · p3) [p1 p4], ReduceBySorting → {p3, p2}]

Out[20]= - [p1 {p3, p2} p4]
```

The option `ReduceBySorting` under the hood calls the function `ChainSort`, thus the same rules apply, for example in the case of Dirac traces:

```
          (*The ChainSimplify option ReduceBySorting uses ChainSort,
          thus the same rules apply*)

 In[23]:= ChainSimplify[⟨p4 {p2, p3, p1} p4], ReduceBySorting → {p1, p4}]

Out[23]= ⟨p4 {p2, p3, p1} p4]

 In[24]:= DeclareMassless[p1];

 In[25]:= ChainSimplify[⟨p4 {p2, p3, p1} p4], ReduceBySorting → {p1, p4}]

Out[25]= ⟨p1 {p4, p2, p3} p1]
```

It is also possible to account for momentum conservation directly inside `ChainSimplify` through the option `MomCon`. This is equivalent to first applying the function `ChainMomentumCon` and then `ChainSimplify` right after. Notice that if a momentum conservation relation is given, it will be applied, independently of whether the final expression is simpler or not.

```
 In[27]:= DeclareMom[p1, p2, p3, p4];

 In[28]:= ChainSimplify[[p1 {p2, p4} p3], MomCon → {p4 → -p1 - p2 - p3}]

Out[28]= -2 (p1 · p2) [p1 p3] - (p2)^2 [p1 p3]
```

3.6 ToTrace

As a final application related to chain manipulation, we have the function `ToTrace`. This function allows to convert chains of the form $\langle q \cdots q]$ and $[q \cdots q\rangle$, which correspond to Dirac traces with a γ_5 insertion, into products of scalar products and Levi-Civita tensors dotted into momenta, through the identity

$$\begin{aligned} \langle q\, p_1 \dots p_{2n+1}\, q] &= \mathrm{Tr}\left[\left(\tfrac{1-\gamma_5}{2}\right)\gamma_\mu \dots \gamma_\nu\right]\; p_1^\mu \dots p_{2n+1}^\nu\,, \\ [q\, p_1 \dots p_{2n+1}\, q\rangle &= \mathrm{Tr}\left[\left(\tfrac{1+\gamma_5}{2}\right)\gamma_\mu \dots \gamma_\nu\right]\; p_1^\mu \dots p_{2n+1}^\nu\,. \end{aligned} \tag{34}$$

In the package Levi-Civita tensors are repesented by the object `epsSH`, where `epsSH[p1,p2,p3,p4]` $=\epsilon_{\mu_1,\mu_2,\mu_3,\mu_4} p_1^{\mu_1} p_2^{\mu_2} p_3^{\mu_3} p_4^{\mu_4}$.

```
30]:= ⟨p1 {p2, p3, p4} p1] // ToTrace
      1
[30]= - (4 ⅈ ∈[p1, p2, p3, p4] + 4 (p1 · p4) (p2 · p3) - 4 (p1 · p3) (p2 · p4) + 4 (p1 · p2) (p3 · p4)
      2
```

The function allows for the option `EpsilonSimplify` which by default is set to `None`. It can take the following string values:

- `ReduceEven`, this option takes into account the fact that the product of two ϵ tensors can be rewritten as a product of Kronecker deltas, and thus in our case where the ϵ tensors are always completely contracted into a set of momenta one gets

$$\epsilon(p_1, p_1, p_3, p_4)\epsilon(q_1, q_2, q_3, q_4) = \begin{vmatrix} (p_1 \cdot q_1) & \cdots & (p_1 \cdot q_4) \\ \vdots & & \vdots \\ (p_4 \cdot q_1) & \cdots & (p_4 \cdot q_4) \end{vmatrix}, \tag{35}$$

  which is the Gram determinant, and where $\epsilon(p_1, p_1, p_3, p_4)$ is the Levi-Civita tensor contracted into the four momenta p_i. Setting `EpsilonSimplify->ReduceEven` will thus convert all products of an even number of epsilon tensors in the output into products of scalar products.

- `KillOdd`, this option, beyond applying the relation (35), also sets to zero all the combinations of an odd number of epsilon tensors. This might be useful for example when dealing with a four-particle process, where thanks to momentum conservation $\epsilon[p_1, p_2, p_3, p_4] = -\epsilon[p_1, p_2, p_3, p_1] - \epsilon[p_1, p_2, p_3, p_2] - \epsilon[p_1, p_2, p_3, p_3] = 0$.

```
(*Simplify even combinations of Levi-Civita tensors*)

⟨p1 {p2, p3, p4} p1]^2 // ToTrace[#, EpsilonSimplify → "ReduceEven"] &

8 ⅈ ∈[p1, p2, p3, p4] (p1 · p4) (p2 · p3) +
 4 (p1 · p4)^2 (p2 · p3)^2 - 8 ⅈ ∈[p1, p2, p3, p4] (p1 · p3) (p2 · p4) -
 8 (p1 · p3) (p1 · p4) (p2 · p3) (p2 · p4) + 4 (p1 · p3)^2 (p2 · p4)^2 +
 8 ⅈ ∈[p1, p2, p3, p4] (p1 · p2) (p3 · p4) + 8 (p1 · p2) (p1 · p4) (p2 · p3) (p3 · p4) -
 8 (p1 · p2) (p1 · p3) (p2 · p4) (p3 · p4) + 4 (p1 · p2)^2 (p3 · p4)^2 -
 4 (- (p1 · p4)^2 (p2 · p3)^2 + 2 (p1 · p3) (p1 · p4) (p2 · p3) (p2 · p4) - (p1 · p3)^2 (p2 · p4)^2 +
    (p1 · p4)^2 (p2)^2 (p3)^2 - 2 (p1 · p2) (p1 · p4) (p2 · p4) (p3)^2 + (p1)^2 (p2 · p4)^2 (p3)^2 -
    2 (p1 · p3) (p1 · p4) (p2)^2 (p3 · p4) + 2 (p1 · p2) (p1 · p4) (p2 · p3) (p3 · p4) +
    2 (p1 · p2) (p1 · p3) (p2 · p4) (p3 · p4) - 2 (p1)^2 (p2 · p3) (p2 · p4) (p3 · p4) -
    (p1 · p2)^2 (p3 · p4)^2 + (p1)^2 (p2)^2 (p3 · p4)^2 + (p1 · p3)^2 (p2)^2 (p4)^2 - 2 (p1 · p2) (p1 · p3)
     (p2 · p3) (p4)^2 + (p1)^2 (p2 · p3)^2 (p4)^2 + (p1 · p2)^2 (p3)^2 (p4)^2 - (p1)^2 (p2)^2 (p3)^2 (p4)^2)

(*Setting to zero odd combinations while simplifying even ones*)

⟨p1 {p2, p3, p4} p1]^2 // ToTrace[#, EpsilonSimplify → "KillOdd"] &

4 (p1 · p4)^2 (p2 · p3)^2 - 8 (p1 · p3) (p1 · p4) (p2 · p3) (p2 · p4) +
 4 (p1 · p3)^2 (p2 · p4)^2 + 8 (p1 · p2) (p1 · p4) (p2 · p3) (p3 · p4) -
 8 (p1 · p2) (p1 · p3) (p2 · p4) (p3 · p4) + 4 (p1 · p2)^2 (p3 · p4)^2 -
 4 (- (p1 · p4)^2 (p2 · p3)^2 + 2 (p1 · p3) (p1 · p4) (p2 · p3) (p2 · p4) - (p1 · p3)^2 (p2 · p4)^2 +
    (p1 · p4)^2 (p2)^2 (p3)^2 - 2 (p1 · p2) (p1 · p4) (p2 · p4) (p3)^2 + (p1)^2 (p2 · p4)^2 (p3)^2 -
    2 (p1 · p3) (p1 · p4) (p2)^2 (p3 · p4) + 2 (p1 · p2) (p1 · p4) (p2 · p3) (p3 · p4) +
    2 (p1 · p2) (p1 · p3) (p2 · p4) (p3 · p4) - 2 (p1)^2 (p2 · p3) (p2 · p4) (p3 · p4) -
    (p1 · p2)^2 (p3 · p4)^2 + (p1)^2 (p2)^2 (p3 · p4)^2 + (p1 · p3)^2 (p2)^2 (p4)^2 - 2 (p1 · p2) (p1 · p3)
     (p2 · p3) (p4)^2 + (p1)^2 (p2 · p3)^2 (p4)^2 + (p1 · p2)^2 (p3)^2 (p4)^2 - (p1)^2 (p2)^2 (p3)^2 (p4)^2)
```

The function ToTrace uses the auxiliary functions TrG and TrG5 which separately perform the computations of $\mathrm{Tr}\left[\gamma_\mu \ldots \gamma_\nu\right]\ p_1^\mu \ldots p_{2n+1}^\nu$ and $\mathrm{Tr}\left[\gamma_5\gamma_\mu \ldots \gamma_\nu\right]\ p_1^\mu \ldots p_{2n+1}^\nu$ respectively.

```
In[34]:= TrG[{p1, p2, p3, p4}]

Out[34]= 4 (p1 · p4) (p2 · p3) - 4 (p1 · p3) (p2 · p4) + 4 (p1 · p2) (p3 · p4)

In[35]:= TrG5[{p1, p2, p3, p4}]

Out[35]= -4 ⅈ ϵ[p1, p2, p3, p4]
```

3.7 SchoutenSimplify

A key component in the simplification of spinor expressions is the application of Schouten identities, which we recall here for the reader's convenience:

$$\langle i\,j\rangle\langle k| + \langle j\,k\rangle\langle i| + \langle k\,i\rangle\langle j| = 0\,, \qquad [i\,j][k| + [j\,k][i| + [k\,i][j| = 0\,. \tag{36}$$

Application of these identities is performed through the function SchoutenSimplify, which takes as single input the expression to be simplified.

```
In[2]:= ⟨1 2⟩ ⟨3 4⟩ + ⟨1 3⟩ ⟨4 2⟩ // SchoutenSimplify

Out[2]= -⟨1 4⟩ ⟨2 3⟩

In[3]:= test = ⟨1 4⟩^2 ⟨2 3⟩^2 + 2 ⟨1 3⟩ ⟨1 4⟩ ⟨2 3⟩ ⟨2 4⟩ +
           ⟨1 3⟩^2 ⟨2 4⟩^2 + 2 ⟨1 2⟩ ⟨1 4⟩ ⟨2 3⟩ ⟨3 4⟩ + 2 ⟨1 2⟩ ⟨1 3⟩ ⟨2 4⟩ ⟨3 4⟩ +
           ⟨1 2⟩^2 ⟨3 4⟩^2 + ⟨4 7⟩^3 ⟨5 6⟩^3 - 3 ⟨4 6⟩ ⟨4 7⟩^2 ⟨5 6⟩^2 ⟨5 7⟩ +
           3 ⟨4 6⟩^2 ⟨4 7⟩ ⟨5 6⟩ ⟨5 7⟩^2 - ⟨4 6⟩^3 ⟨5 7⟩^3 + 3 ⟨4 5⟩ ⟨4 7⟩^2 ⟨5 6⟩^2 ⟨6 7⟩ -
           6 ⟨4 5⟩ ⟨4 6⟩ ⟨4 7⟩ ⟨5 6⟩ ⟨5 7⟩ ⟨6 7⟩ + 3 ⟨4 5⟩ ⟨4 6⟩^2 ⟨5 7⟩^2 ⟨6 7⟩ +
           3 ⟨4 5⟩^2 ⟨4 7⟩ ⟨5 6⟩ ⟨6 7⟩^2 - 3 ⟨4 5⟩^2 ⟨4 6⟩ ⟨5 7⟩ ⟨6 7⟩^2 + ⟨4 5⟩^3 ⟨6 7⟩^3;

In[6]:= SchoutenSimplify[test] // AbsoluteTiming

Out[6]= {0.0000821, 4 ⟨1 3⟩^2 ⟨2 4⟩^2}
```

SchoutenSimplify under the hood runs a modified version of Mathematica's function FullSimplify and thus allows for the same options, being also subject to the same limitations.

```
In[7]:= SchoutenSimplify[
          (4 Sin[x]^2 Cos[x]^2 + 4 Sin[x] Cos[x] + 1) (-⟨1 3⟩ ⟨2 4⟩ + ⟨1 2⟩ ⟨3 4⟩)]
Out[7]= - (Cos[x] + Sin[x])^4 ⟨1 4⟩ ⟨2 3⟩

        (*Same input but not allowing trigonometric identities*)

In[8]:= SchoutenSimplify[(4 Sin[x]^2 Cos[x]^2 + 4 Sin[x] Cos[x] + 1)
           (-⟨1 3⟩ ⟨2 4⟩ + ⟨1 2⟩ ⟨3 4⟩), Trig → False]
Out[8]= - (1 + 2 Cos[x] Sin[x])^2 ⟨1 4⟩ ⟨2 3⟩
```

3.8 SpinorReplace

Replacing spinor labels inside an expression might seem a simple enough task, however it can clearly not be accomplished by a mere $exp\,/.\,p_{old} \rightarrow p_{new}$. Spinor replacements are performed through the function `SpinorReplace[expression,rules]`, which acts on angle and square brackets as well as chain extrema. We stress that

- replacement rules are given in terms of bare spinors, which do not have explicit indices since the replacement will be applied independently of the index structure. Bare spinors have the same "linearity" properties with respect to declared momenta as all the other building blocks.

```
         (*The bare spinors*)

In[61]:= {SpinorUndotBare[p][$lam], SpinorUndotBare[p][$mu]}
Out[61]= {λ[p], μ[p]}

In[62]:= {SpinorDotBare[p][$lam], SpinorDotBare[p][$mu]}
Out[62]= {λ̃[p], μ̃[p]}
```

- replacements can involve any type of linear combinations of spinors. Notice that `SpinorDotPure[p1]->SpinorDotPure[p2+z*p3]` is equivalent to `SpinorDotPure[p1]->SpinorDotPure[p2]+z*SpinorDotPure[p3]` if `p1,p2,p3` have previously been declared as momentum labels through `DeclareMom`.

```
        (*Simple replacements*)

In[38]:= SpinorReplace[⟨p1 p2⟩, {λ[p2] -> λ[p3]}]

Out[38]= ⟨p1 p3⟩

In[39]:= SpinorReplace[[p2 {p1, p4} p2], {λ̃[p2] -> λ̃[p3]}]

Out[39]= [p3 {p1, p4} p3]

        (*Arbitrarily complex replacement rules are allowed*)

In[48]:= SpinorReplace[[p2 {p1, p4} p2], {λ̃[p2] -> λ̃[p2] + 3 * c * λ̃[p3]}]

Out[48]= [p2 {p1, p4} p2] + 3 c [p2 {p1, p4} p3] + 3 c [p3 {p1, p4} p2] + 9 c^2 [p3 {p1, p4} p3]

        (*Which is equivalent to*)

In[46]:= DeclareMom[p2, p3];

In[49]:= SpinorReplace[[p2 {p1, p4} p2], {λ̃[p2] -> λ̃[p2 + 3 * c * p3]}]

Out[49]= [p2 {p1, p4} p2] + 3 c [p2 {p1, p4} p3] + 3 c [p3 {p1, p4} p2] + 9 c^2 [p3 {p1, p4} p3]
```

- replacements can involve linear combinations of spinors dotted into momentum matrices simply through a standard dot, without the need to declare the momenta previously

```
        (*Replacements with momenta dotted into the spinors*)

In[41]:= SpinorReplace[⟨p1 p2⟩, {λ[p2] → pp.qq.λ[p3]}]

Out[41]= ⟨p1 {pp, qq} p3⟩

In[42]:= SpinorReplace[⟨p1 p2⟩, {λ[p2] → -3 c * pp.λ̃[p3]}]

Out[42]= -3 c ⟨p1 {pp} p3]

In[51]:= SpinorReplace[⟨p1 p2⟩, {λ[p2] → -3 c * pp.λ̃[p3] + qq.kk.λ[p7]}]

Out[51]= -⟨p7 {kk, qq} p1⟩ - 3 c [p3 {pp} p1⟩
```

- Replacements involving reference spinors work the same way, just by using bare μ spinors insteam of bare λ spinors.

```
        (*Replacements with reference momenta work the same way*)

In[56]:= SpinorReplace[⟨p̅2̅ {p1, p4} p2⟩, {μ[p2] -> λ[p3]}]

Out[56]= ⟨p3 {p1, p4} p2⟩
```

3.9 SpinorDerivative

Another interesting feature available in the package is the function `SpinorDerivative`, which allows to take the derivative of spinorial expressions with respect to the spinors themselves. This function takes two inputs, the expression to be differentiated and the spinor with respect to which to differentiate.

```
    (*Simple derivatives*)

63]:= test = {λ_a[p], μ_a[p], λ_a[q], μ_a[q], λ^a[p], μ^a[p], λ^a[q], μ^a[q], ⟨p q⟩, ⟨p p̅⟩,
        ⟨p q̅⟩, -⟨p q⟩, ⟨q p̅⟩, ⟨q q̅⟩, -⟨p p̅⟩, -⟨q p̅⟩, ⟨p̅ q̅⟩, -⟨p q̅⟩, -⟨q q̅⟩, -⟨p̅ q̅⟩};

64]:= SpinorDerivative[test, λ_b[p]]

[64]= {δ_a^b, 0, 0, 0, ϵ^{a b}, 0, 0, 0, -λ^b[q], -μ^b[p], -μ^b[q], λ^b[q], 0, 0, μ^b[p], 0, 0, μ^b[q], 0, 0}
```

Giving a lits or a products of spinor as second arguments wil perform sequential derivatives with respect to all the specified spinors:

```
        (*Multiple derivatives at once can be
         done giving a list or a product as second input*)

In[65]:= SpinorDerivative[⟨p q⟩ [p q], {λ^a[q], λ̃_b[p]}]

Out[65]= -λ̃^b[q] λ_a[p]

In[66]:= SpinorDerivative[⟨p q⟩ [p q], λ^a[q] * λ̃_b[p]]

Out[66]= -λ̃^b[q] λ_a[p]
```

`SpinorDerivative` can differentiate any arbitrary function of the spinors.

```
        (*Can differentiate any arbitrary function of the spinors*)

In[67]:= SpinorDerivative[1 / (⟨p q⟩ [p q]), {λ^a[q], λ̃_b[p]}]
```

$$\text{Out[67]=}\quad -\frac{\tilde{\lambda}^{b}[q]\,\lambda_{a}[p]}{\langle p\,q\rangle^{2}\,[p\,q]^{2}}$$

```
In[68]:= SpinorDerivative[Sqrt[⟨p q⟩ [p q]], {λ^a[q], λ̃_b[p]}]
```

$$\text{Out[68]=}\quad \frac{\langle p\,q\rangle\,[p\,q]\,\tilde{\lambda}^{b}[q]\,\lambda_{a}[p]}{4\,(\langle p\,q\rangle\,[p\,q])^{3/2}} - \frac{\tilde{\lambda}^{b}[q]\,\lambda_{a}[p]}{2\,\sqrt{\langle p\,q\rangle\,[p\,q]}}$$

```
        (*Completely generic function:*)

In[69]:= Clear[f]

In[70]:= SpinorDerivative[f[⟨p q⟩^2], λ^a[p]]
```

$$\text{Out[70]=}\quad 2\,\langle p\,q\rangle\,f'\left[\langle p\,q\rangle^{2}\right]\,\lambda_{a}[q]$$

3.10 ToMandelstam

The function ToMandelstam converts products of spinor brackets as well as scalar products into Mandelstam invariants, in other words it performs the replacement $\langle i\,j\rangle[j\,i], 2\,p_i \cdot p_j \mapsto s_{ij}$, as long as the momenta p_i and p_j have been declared as massless.

```
In[2]:= test = ⟨p1 p2⟩ [p1 p2] * mp[p3, p4];

        (*No massless momentum declared*)

In[3]:= test // ToMandelstam

Out[3]= (p3 · p4) ⟨p1 p2⟩ [p1 p2]

        (*p1 and p2 declared as massless*)

In[8]:= DeclareMassless[p1, p2];

In[5]:= test // ToMandelstam

Out[5]= - (p3 · p4) S[p1, p2]

        (*Also p3 and p4 declared massless*)

In[6]:= DeclareMassless[p3, p4];

In[7]:= test // ToMandelstam
```

$$\text{Out[7]=}\quad -\frac{1}{2}\,S[p1,\,p2]\times S[p3,\,p4]$$

3.11 VecToSpinors

The function `VecToSpinors` allows to convert uncontracted vector quantities in the input to to spinor expressions, in particular to spinor chains with an uncontracted Pauli matrix. The function requires as input argument the expression to be converted, and optionally two lists of momentum labels for the positive and negative helicity states. The latter are only required when polarization vectors are included in the input. Notice that currently this mapping is available only for massless momenta.

```
          (*Convert vector quantities to spinor quantities*)

In[2]:=   q_μ // VecToSpinors

Out[2]=   q_μ

In[3]:=   DeclareMassless[p, q];

In[4]:=   q_μ // VecToSpinors

          ⟨q {σ_μ} q]
Out[4]=   ───────────
               2

          (*The Pauli matrix place holder contracts with momenta*)

In[5]:=   % * p^μ

          ⟨q {p} q]
Out[5]=   ─────────
              2

          (*Fierz identity applies*)

In[6]:=   ⟨p1 {σ_μ} p2] ⟨q1 {σ^μ} q2]

Out[6]=   -2 ⟨p1 q1⟩ [p2 q2]
```

Below, specification of the helicity for polarization vectors.

```
         (*For polarization vectors we need to specify helicity*)

In[7]:=  ε_μ[q, k] // VecToSpinors

Out[7]=  ε_μ[q, k]

In[8]:=  VecToSpinors[ε_μ[q, k], {q}, {}]

            ⟨k {σ_μ} q]
Out[8]=  - ------------
            √2 ⟨q k⟩

In[9]:=  VecToSpinors[ε_μ[q, k], {}, {q}]

          ⟨q {σ_μ} k]
Out[9]=  ------------
          √2 [q k]

In[10]:= % * p^μ

          ⟨q {p} k]
Out[10]= ----------
          √2 [q k]
```

3.12 MpToSpinors

The function `MpToSpinors` converts scalar products into spinor products, it takes as argument the input expression to be converted as well as two optional arguments being lists of momentum labels identifying positive and negative helicity states. Once again, notice that the current implementation only works for massless states.

```
         (*Currently, if momenta are not declared massless nothing happens*)

In[2]:=  mp[p, q] // MpToSpinors

Out[2]=  (p · q)

In[3]:=  DeclareMassless[p, q];

In[4]:=  mp[p, q] // MpToSpinors

           1
Out[4]=  - - ⟨p q⟩ [p q]
           2
```

Below, the function when applied to scalar products including polarization vectors.

```
        (*With polarizations helicity information is required*)

In[5]:= mp[p, ε[q, k]]

Out[5]= (p · ε[q, k])

In[6]:= MpToSpinors[(p · ε[q, k]), {q}, {}]
```

$$\text{Out[6]}= \frac{\langle p\,k\rangle\,[p\,q]}{\sqrt{2}\,\langle q\,k\rangle}$$

```
In[7]:= MpToSpinors[(p · ε[q, k]), {}, {q}]
```

$$\text{Out[7]}= -\frac{\langle p\,q\rangle\,[p\,k]}{\sqrt{2}\,[q\,k]}$$

3.13 CompleteDenominators

The function `CompleteDenominators` completes individual squares brackets in denominators to full Mandelstam invariants. The current implementation of this functions only works when we are dealing with a fully massless case, in fact if $p_i = |i\rangle[i| + \rho_i|\bar{i}\rangle[\bar{i}|$ and $p_j = |j\rangle[j| + \rho_j|\bar{j}\rangle[\bar{j}|$ then $\langle i\,j\rangle[j\,i] \neq s_{ij}$, and reconstructing the invariat from the products of brackets becomes far more involved.

$$\text{In[2]}:= \frac{1}{\langle i\,j\rangle} \text{ // CompleteDenominators}$$

$$\text{Out[2]}= \frac{1}{\langle i\,j\rangle}$$

```
In[3]:= DeclareMassless[i, j];
```

$$\text{In[4]}:= \frac{1}{\langle i\,j\rangle} \text{ // CompleteDenominators}$$

$$\text{Out[4]}= -\frac{[i\,j]}{\text{S}[i,\,j]}$$

4 Numerics

In most applications it is extremely convenient to be able to perform numeric checks of various nature, this might include for example numerically comparing two very different analytic expression or make sure that a result involving some reference spinor is independent of the chosen reference. To take a step further, numerical evaluations can be used to extrapolate analytic results through functional reconstruction. Then, cumbersome and complicated intermediate expressions are reduced to mere numbers, while focus is put only on the final quantity of interested. This approach is particularly powerful when combined with numeric evaluations over finite fields [6], which provide exact numeric results (not subject to precision loss), while still maintaining good computational performances.

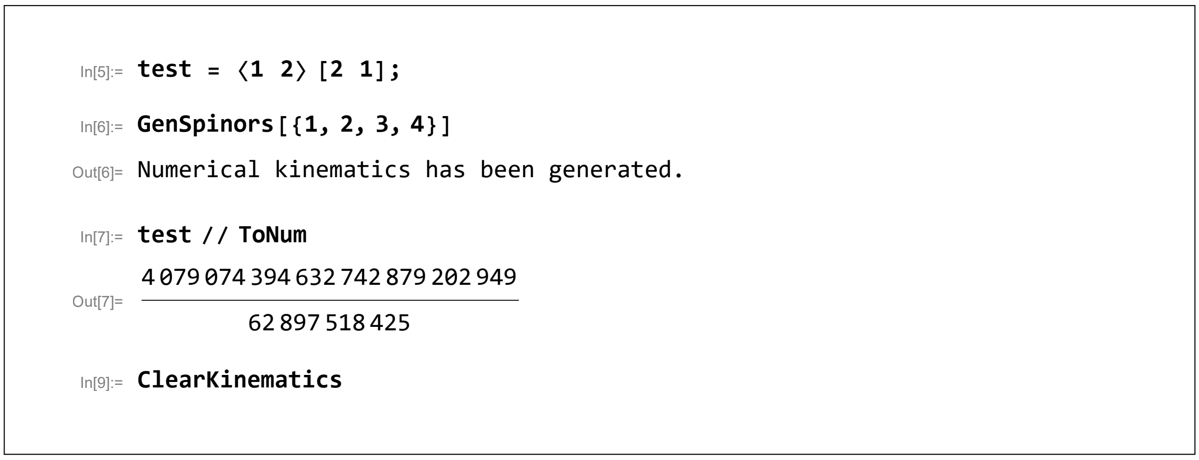
In this section we introduce the functions of the package which are related to numeric evaluations. The most important functions of the section are `GenSpinors`, which is used to generate numeric kinematics for a specified momentum configuration, and `ToNum` which converts a given analytic expression into the corresponding numerical value.

4.1 ToNum

The function `ToNum` translates a given analytic expression into the corresponding number, based on a numerical kinematic configuration generated through `GenSpinors`. The general logic is as follows:

- One uses `GenSpinors` in order to generate some numeric kinematics (for details on `GenSpinors` see Section 4.2), which is then stored in the background. At this stage only two-component Weyl spinors are numerically computed.
- To each analytic object we associate a corresponding alias in which numeric values are stored. For example `SpinorUndot` $\mapsto$ `SpinorUndotN`, where the first is an analytic object while the second is vector of two numbers.
- By applying `ToNum` to an analytic expression, all the functions appearing in it are replaced by their numeric aliases, producing a numeric output. At this stage, the fundamental objects generated by `GenSpinor` (*i.e.* the uncontracted spinors) are combined into derived quantities such as the spinor brackets. In order to optimise performances, these derived quantities are only computed if encountered in an expression, and then stored for later use.
- The stored numeric kinematics, both the fundamental as well as the derived objects, can be cleared through `ClearKinematics`

Below is a simple example.



```
In[5]:=  test = ⟨1 2⟩ [2 1];

In[6]:=  GenSpinors[{1, 2, 3, 4}]
Out[6]=  Numerical kinematics has been generated.

In[7]:=  test // ToNum
```

$$\text{Out[7]=}\quad \frac{4\,079\,074\,394\,632\,742\,879\,202\,949}{62\,897\,518\,425}$$

```
In[9]:=  ClearKinematics
```

4.2 GenSpinors

Generation of numerical kinematics to us means finding a set of numbers to associate to all the spinors describing a given process, such that momenta are on-shell and conserved[10]. Since we use spinors as building blocks, the on-shell condition is automatically satisfied in the massless case and can easily be accounted for in the massive case. In order to satisfy momentum conservation one could make use of momentum-twistor variables [13]: upon mapping the spinors to the twistor space both on-shellness and momentum conservation are trivialised thus any random set of numbers is suited to describe the kinematics (see for example [9] for more details). However, since we want to accommodate several different scenarios including the massless and massive case, parametric kinematics, degenerate (three-point) and constrained kinematics, we found it easier to opt for a different approach. We work in spinor space, and upon randomly generating most of the components, we account for momentum conservation and other constraints by solving a system in the final variables, which have been carefully chosen so that they are always rational combinations of the previously fixed variables. This procedure allows a large degree of flexibility in the type of kinematics one can generate, while also retaining the benefit of rational number output.

`GenSpinors` only has one mandatory argument, being a list of momentum labels for which kinematics needs to be generated, the function however allows for a great number of options. In the following subsections we give a detailed description of all these options, along with their intended use and possible limitations. Before going into the details we stress once again that the generated kinematics is always complex.

AllMassless: Massive vs massless states

When `GenSpinors` is provided only with the mandatory argument of a single list of labels, it is implicitly assumed that all the states should be massive and that no particular relation between the masses exists.

[10]We will follow a convention of all out-going momenta, thus conservation will always result in the momenta summing to zero.

```
         (*Generation of massive states*)

In[2]:=  GenSpinors[{1, 2, 3, 4}]

Out[2]=  Numerical kinematics has been generated.

         (*Momenta are conserved*)

In[3]:=  ({S[1, 2] == S[3, 4], S[1, 3] == S[2, 4], S[1, 4] == S[2, 3]}) // ToNum

Out[3]=  {True, True, True}

         (*Masses*)

In[4]:=  {S[1], S[2], S[3], S[4]} // ToNum

           2 116 004 486 300 755 356 408    320 465 584 797 824
Out[4]=  { ─────────────────────────────, ───────────────────,
                  3 298 414 553                  397
             2 152 061 700 226 812       1 832 693 083 145 907 345 612
          - ───────────────────────, - ─────────────────────────────── }
                   90 913                       471 202 079
```

Using the option `AllMassless` it is possible to generate fully massless kinematics, while by providing a list of two lists as inputs one can generate a mixed massive and massless process.

```
         (*Generation of only massless states*)

In[5]:=  GenSpinors[{1, 2, 3, 4}, AllMassless → True];

In[6]:=  {S[1], S[2], S[3], S[4]} // ToNum

Out[6]=  {0, 0, 0, 0}

         (*Generation of mixed massless and massive states*)

In[7]:=  GenSpinors[{{1, 2}, {3, 4}}];

In[8]:=  {S[1], S[2], S[3], S[4]} // ToNum

             486 428 415 680 278 048 512      143 648 686 877 584
Out[8]=  {- ───────────────────────────, - ───────────────────, 0, 0}
                  1 738 423 375                     299
```

Consecutive use of `GenSpinors` does not clear existing kinematics, it will only overwrite existing labels which reapper in a new call.

```
        (*Consecutive generation will not delete previous kinematics*)

In[17]:= GenSpinors[{1, 2, 3, 4}, AllMassless → True];

In[18]:= {S[1], S[2], S[3], S[4]} // ToNum

Out[18]= {0, 0, 0, 0}

In[19]:= GenSpinors[{{1, 2}, {q3, q4}}];

In[20]:= {S[1], S[2], S[3], S[4]} // ToNum
```

$$\text{Out[20]=}\ \left\{-\frac{150\,474\,066\,529\,273\,275\,055}{661\,520\,134}, \frac{14\,552\,913\,457\,680}{11}, 0, 0\right\}$$

SameMasses

Using the option `SameMasses` it is possible to declare some of the masses to be the same, this option takes either a single list as input (if only one set of masses is required to be the same) or a list of lists allowing for different subsets of momenta having the same masses.

```
        (*Generation of massive states with same masses*)

In[9]:=  GenSpinors[{1, 2, 3, 4}, SameMasses → {{1, 2}, {3, 4}}];

In[10]:= {S[1] == S[2], S[3] == S[4]} // ToNum

Out[10]= {True, True}

        (*Mixed states, three equal masses and two massless*)

In[11]:= GenSpinors[{{1, 2, 3}, {4, 5}}, SameMasses → {1, 2, 3}];

In[12]:= {S[1] == S[2], S[2] == S[3], S[4], S[5]} // ToNum

Out[12]= {True, True, 0, 0}
```

DisplaySpinors

The boolean option `DisplaySpinors` controls whether or not the numerical spinors are displayed by `GenSpinors` upon generation. The default value is False, and prompts the function to simply return a string stating the completion of the kinematics' generation.

```
(*Display the generated spinors*)

GenSpinors[{{1, 2}, {3, 4}}, DisplaySpinors → True]

Output reads {|λ⟩,|λ],|μ⟩,|μ]} and {|λ⟩,|λ]} for massive and massless spinors respectively

{{{{-908, -55 731 363 036 824/451 278 043}, {211, 162}, {-917, 584}, {705, 460}},
  {{1 374 458/241, 512}, {946, 241}, {-97, 922}, {945, 142}}},
 {{{-908, 724}, {510 414 549/109 414, 642}}, {{-416, 17 097 060 970 318/451 278 043}, {552, 502}}}}
```

Seed

The numerical kinematics is generated through `Mathematica`'s random number generator. In order for the same numeric result to be repeatable it is possible to seed the random generator through the option `Seed`. Since this option will simply pass its value to `Mathematica`'s built in `SeedRandom`, it accepts the same values as the latter. Below, upon use of `Seed`, repeatedly generated kinematics leads to the same numbers.

```
(*Seed the random generator for repeatable results*)

GenSpinors[{1, 2, 3, 4}, {AllMassless → True, DisplaySpinors → True, Seed → 7}]

Output reads {|λ⟩,|λ],|μ⟩,|μ]} and {|λ⟩,|λ]} for massive and massless spinors respectively

{{{{140 071 708/131 343, 63 182 447/394 029}, {957, 849}}, {{-29 014 859/43 781, -134 972 860/131 343}, {485, 842}},
  {{-279, 881}, {39, 758}}, {{-964, 435}, {714, 141}}}}

GenSpinors[{1, 2, 3, 4}, {AllMassless → True, DisplaySpinors → True, Seed → 7}]

Output reads {|λ⟩,|λ],|μ⟩,|μ]} and {|λ⟩,|λ]} for massive and massless spinors respectively

{{{{140 071 708/131 343, 63 182 447/394 029}, {957, 849}}, {{-29 014 859/43 781, -134 972 860/131 343}, {485, 842}},
  {{-279, 881}, {39, 758}}, {{-964, 435}, {714, 141}}}}
```

ParameterRange

It is possible to set the interval within which random numbers are generated when feeding into `GenSpinors`. This is done through the option `ParameterRange`, whose default value is 10^3, and which accepts either a single number n_1 (range will be 0 to n_1) or a list of two numbers $\{n_1, n_2\}$ (range will be between n_1 and n_2) as input. It is important to stress that this option only controls the range of the randomly generated parts of the kinematics, meaning that the spinor components which are fixed by solving momentum conservation conditions (or other constraints imposed on the kinematics through other options of `GenSpinors`) are not bound by this range.

```
(*Select range for the random parameters in the spinors*)

GenSpinors[{1, 2, 3, 4},
 {AllMassless → True, DisplaySpinors → True, ParameterRange → 100}]

Output reads {|λ⟩,|λ],|μ⟩,|μ]} and {|λ⟩,|λ]} for massive and massless spinors respectively

       86 255   24 821
{{{{- ───────, ───────}, {65, 79}},
        564      564

   266 749     124 891
 {{───────, - ───────}, {31, 29}}, {{-9, 68}, {41, 38}}, {{-68, 19}, {64, 19}}}}
     564         564
```

RationalKinematics

For a variety of applications, including for example the already mentioned functional reconstruction on finite fields [6], it is convenient to generate kinematics on the field of rational numbers. While this is the default for `GenSpinors`, it is possible to control whether kinematics is generated on $\mathbb{Q}$ or $\mathbb{R}$ through the option `RationalKinematics`.

```
(*Rational kinematics is default, but can be deactivated*)

GenSpinors[{1, 2, 3, 4},
 {AllMassless → True, DisplaySpinors → True, RationalKinematics → False}]

Output reads {|λ⟩,|λ],|μ⟩,|μ]} and {|λ⟩,|λ]} for massive and massless spinors respectively

{{{{821.121, -621.544}, {630.515, 770.692}},
  {{106.806, 5.5678}, {773.897, 439.291}}, {{-650.636, 362.736}, {889.287, 982.927}},
  {{-40.7878, 121.716}, {534.086, 986.138}}}}
```

Parametric and ParameterName

Considering for example an $n-$particle massless process, it is well known that it can be described by a minimal set of $3n - 10$ independent parameters. This can be accounted for in the numeric generation through the boolean option `Parametric`, whose default value is False. Setting it to True will generate a set of spinor components which are parametrized in terms of this minimal set of parameters called `$par`.

```
GenSpinors[{1, 2, 3, 4}, {AllMassless → True, DisplaySpinors → True, Parametric → True}]

Output reads {|λ⟩,|λ],|μ⟩,|μ]} and {|λ⟩,|λ]} for massive and massless spinors respectively

{{{{- 3 (-8615369 + 4062 $par[1] + 3372 $par[2])
      ───────────────────────────────────────────── ,
                        17909
   -48681943 + 28044 $par[1] + 11124 $par[2]
   ─────────────────────────────────────────── }, {27, 520}},
                     35818
  {{-2465883 + 44005 $par[1] + 36530 $par[2]   4644567 - 101270 $par[1] - 40170 $par[2]
    ────────────────────────────────────────, ──────────────────────────────────────────}
                     53727                                     107454
   {849, 432}}, {{-677, 779}, {$par[1], 704}}, {{-562, 309}, {$par[2], 452}}}}
```

The name of the parameters can be change through the option `ParameterName`.

```
(*Generate parametric kinematics with custom parameter name*)

GenSpinors[{1, 2, 3, 4},
 {AllMassless → True, DisplaySpinors → True, Parametric → True, ParameterName → zz}]

Output reads {|λ⟩,|λ],|μ⟩,|μ]} and {|λ⟩,|λ]} for massive and massless spinors respectively

{{{{9 (-2988958 + 14400 zz[1] + 11175 zz[2])
    ─────────────────────────────────────────── ,
                     278992
   - 15 (-3802274 + 15075 zz[1] + 22275 zz[2])
     ─────────────────────────────────────────── }, {630, 337}},
                       139496
  {{39049290 - 64704 zz[1] - 50213 zz[2]   5 (-16558290 + 22579 zz[1] + 33363 zz[2])
    ────────────────────────────────────, ─────────────────────────────────────────── },
                   278992                                    139496
   {434, 675}}, {{-192, 670}, {zz[1], 246}}, {{-149, 990}, {zz[2], 99}}}}
```

Three-point kinematics

The three-point kinematic is somewhat special due to the fact that, based on the helicity of the involved particles, two different types of kinematics are manifested. The two scenarios, both of them satisfying $p_+p_2 + p_3 = 0$ and $s_{ij} = 0$ and both of them being allowed only because we consider complex kinematics[11], consist of $\langle i\, j\rangle = 0$ and $[i\, j] \neq 0$ for every i, j, or the opposite. Which three-point kinematics is numerically generated is controlled by the option `Type3pt` which admits values `$angle` and `$square`.

[11] Alternatively one could have considered a different signature of space time [12].

```
        (*Generate 3pt massless kinematics*)

In[2]:= GenSpinors[{1, 2, 3}, {AllMassless → True, Type3pt → $angle}];

In[3]:= {⟨1 2⟩, ⟨2 3⟩, ⟨3 1⟩} // ToNum

                    358166   16833802
Out[3]= {-358166, - ------, - --------}
                      13        13

In[4]:= {[1 2], [2 3], [3 1]} // ToNum

Out[4]= {0, 0, 0}

In[5]:= GenSpinors[{1, 2, 3}, {AllMassless → True, Type3pt → $square}];

In[6]:= {⟨1 2⟩, ⟨2 3⟩, ⟨3 1⟩} // ToNum

Out[6]= {0, 0, 0}

In[7]:= {[1 2], [2 3], [3 1]} // ToNum

                    30944   12470432
Out[7]= {-185664, - -----, - --------}
                     49        49
```

SetMomentum

In certain instances it might be convenient to set newly generated values of momenta to be equal (or opposite) to already generated momenta. This can be done through the option `SetMomentum`, which admits as a value a list of m multiples of momentum labels, and fixes the first m input labels through the given relation. An example is given below, where we consider a unitarity double cut of a four point amplitude.

```
(*Set momenta to specific values*)

GenSpinors[{p1, p2, q1, q2}, AllMassless → True];

GenSpinors[{"-q1", "-q2", p3, p4}, {AllMassless → True, SetMomentum → {-q1, -q2}}];

{S[p1, p2] == S[p3, p4], S[q1, q2] == S["-q1", "-q2"]} // ToNum
{True, True}
```

The given relationship between the momenta can be specified through any arbitrary number, but symbolic relations are not allowed and will prompt an error message.

```
           (*Any numeric multiple is allowed*)
In[11]:=   GenSpinors[{k1, k2, k3, k4}, {AllMassless → True, SetMomentum → {3 q1, 3 q2}}];
In[12]:=   S[k1, k2] == 9 * S[q1, q2] // ToNum
Out[12]=   True
```

Acknowledgments

We would like to thank Stefano De Angelis for his constant feedback on the early versions of the package, as well as for the collaboration on related topics, and Rodolfo Panerai for inspiring the original implementation of the code. We would also like to thank Gabriele Travaglini, Adreas Brandhuber and Sebastian Pögel for useful discussions and feedback, and Pierpaolo Mastrolia for discussions on `S@M`. This project has received funding by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 764850 *"SAGEX"*.

Bibliography

[1] M. L. Mangano and S. J. Parke, “Multiparton amplitudes in gauge theories,” *Phys. Rept.* **200** (1991) 301–367, arXiv:hep-th/0509223 [hep-th].

[2] D. A. Kosower, “Next-to-maximal helicity violating amplitudes in gauge theory,” *Phys. Rev. D* **71** (2005) 045007, arXiv:hep-th/0406175.

[3] N. Arkani-Hamed, T.-C. Huang, and Y.-t. Huang, “Scattering Amplitudes For All Masses and Spins,” arXiv:1709.04891 [hep-th].

[4] C. Cheung and D. O’Connell, “Amplitudes and Spinor-Helicity in Six Dimensions,” *JHEP* **07** (2009) 075, arXiv:0902.0981 [hep-th].

[5] D. Maitre and P. Mastrolia, “S@M, a Mathematica Implementation of the Spinor-Helicity Formalism,” *Comput. Phys. Commun.* **179** (2008) 501–574, arXiv:0710.5559 [hep-ph].

[6] T. Peraro, “Scattering amplitudes over finite fields and multivariate functional reconstruction,” *JHEP* **12** (2016) 030, arXiv:1608.01902 [hep-ph].

[7] M. Accettulli Huber and S. De Angelis, “Standard Model EFTs via on-shell methods,” *JHEP* **11** (2021) 221, arXiv:2108.03669 [hep-th].

[8] L. J. Dixon, “Calculating scattering amplitudes efficiently,” in *Theoretical Advanced Study Institute in Elementary Particle Physics (TASI 95): QCD and Beyond*, pp. 539–584. 1, 1996. arXiv:hep-ph/9601359.

[9] H. Elvang and Y.-t. Huang, “Scattering Amplitudes,” arXiv:1308.1697 [hep-th].

[10] J. M. Henn and J. C. Plefka, *Scattering Amplitudes in Gauge Theories*, vol. 883. Springer, Berlin, 2014.

[11] A. Brandhuber, J. Plefka, and G. Travaglini, “The SAGEX Review on Scattering Amplitudes Chapter 1: Modern Fundamentals of Amplitudes,” *J. Phys. A* **55** no. 44, (2022) 443002, arXiv:2203.13012 [hep-th].

[12] E. Witten, “Perturbative gauge theory as a string theory in twistor space,” *Commun. Math. Phys.* **252** (2004) 189–258, arXiv:hep-th/0312171.

[13] A. Hodges, “Eliminating spurious poles from gauge-theoretic amplitudes,” *JHEP* **05** (2013) 135, arXiv:0905.1473 [hep-th].