

Automation of analytical calculations in high energy physics with Redberry CAS

Stanislav Poslavsky

Institute for High Energy Physics NRC “Kurchatov Institute”, 142281 Protvino, Russia
Institute for Theoretical and Experimental Physics NRC “Kurchatov Institute”, 117218
Moscow, Russia

E-mail: stvlpos@mail.ru

Abstract. Here we present a brief report on recent developments of the **Redberry** computer algebra system through two practical examples: derivation of Feynman rules from arbitrary Lagrangians and computing NLO processes with heavy quarkonia.

1. Introduction

Redberry is a high performance computer algebra system [1, 2] which is specifically focused on applications in high energy physics (HEP). The main distinguishing feature of **Redberry** is that it considers all mathematical objects arising in HEP (scalars, spinors, matrices and in general arbitrary tensors) in a uniform way. This means that e.g. indices of tensors are automatically considered as patterns, all algebraic operations and functions are defined for objects with indices and the user need not worry about such things as dummy index clashes, relabelling of indices in substitutions etc. While **Redberry** comes with an extensive set of programming features which makes it easily extendable, out of the box it provides a wide range of functions for performing standard computations in HEP. Through the recent developments of the system, it is worth to mention the improved d -dimensional Dirac algebra for dimensional regularization, features for doing Fourier transform of the Lagrangian, a set of newly introduced transformations and language features and major performance improvements.

In the following two sections we will briefly illustrate the new features of **Redberry** by two practical examples: derivation of Feynman rules from the Lagrangian and computing next-to-leading order (NLO) contributions to processes with heavy quarkonia.

2. Deriving Feynman rules

For the first illustration let's consider an example which illustrates the derivation of Feynman rules for the gravitational field with the following action:

$$\mathcal{L} = \sqrt{-g} \left(-\frac{c}{\kappa^2} \mathcal{R} - b \mathcal{R}^2 + a \mathcal{R}_{ab} \mathcal{R}^{ab} + \gamma \mathcal{R} \phi^2 + g^{ab} \partial_a \phi \partial_b \phi - m^2 \phi^2 + \mathcal{L}_{g.f.} \right),$$

where ϕ is a scalar matter field and the standard definition of the Riemann tensor $\mathcal{R}^\lambda_{\rho\alpha\beta} = \partial_\alpha \Gamma^\lambda_{\rho\beta} - \partial_\beta \Gamma^\lambda_{\rho\alpha} + \Gamma^\lambda_{\alpha\sigma} \Gamma^\sigma_{\rho\beta} - \Gamma^\lambda_{\beta\sigma} \Gamma^\sigma_{\rho\alpha}$ is implied. The first step towards the derivation of Feynman rules is to perform a weak field expansion:

$$g_{ab}(x) = \eta_{ab} + h_{ab}(x), \quad g^{ab}(x) = \eta^{ab} - h^{ab}(x) + h^a_c(x) h^{cb}(x) + \dots, \quad \sqrt{-g} = 1 + h^a_a + \dots,$$

where η_{ab} is the Minkowski metric and $h_{ab}(x)$ is considered as a quantum perturbation (assumed to be also symmetric). With this definition, the gauge fixing term $\mathcal{L}_{g.f.}$ in the initial Lagrangian is chosen as $1/(2f)(\partial_a h^{ab}(x) - \partial^b h_a^a/2)^2$.

The following boilerplate code inputs all required definitions into Redberry:

```

1  setSymmetric 'h_ab[x_a]'
2  L = 'det * (Lg + Lf + Li + ...)'.t //total Lagrangian
3  //Lg - gravity, Lf - matter, Li - interaction
4  Lg = 'Lg = -c*R/k**2 - b*R**2 + a*R_ab*R_cd*m^ac[x_a]*m^bd[x_a]'.t
5  Lf = 'Lf = m^ab[x_a]*f^(1)_a[x_a]*f^(1)_b[x_a] - m**2*f[x_a]**2'.t
6  Li = 'Li = g*R*f[x_a]**2'.t
7  //Ri - Riemann, Chr - Christoffel, Ric - Ricci
8  Ri = 'R^a_bcd = G^(1)^{a}_{db c}[x_a] - ...'.t
9  Chr = ... ; Ric = ...; R = ...;

11 //WF - weak field, DT - sqrt(-g)
12 WF = 'm^ab[x_a] = g^ab - h^ab[x_a] + h^a_d[x_a]*h^db[x_a] + ...'.t
13 DT = 'det = 1 + h^a_a[x_a] + ...'.t

15 //subtitute all into the total Lagrangian
16 L <<= Lg & Lf & Li & R & Ric & Ri & Chr & WF & DT & ...

```

Here in the first line we specify that $h_{ab}(x)$ is a symmetric field; this will be automatically taken into account in all simplifications. The syntax `'...'.t` translates string expression into the computer object. For the original metric field $g_{ab}(x)$ the definition `m_ab[x_a]` is used while for the Minkowski metric there is a built-in definition `g_ab`. Partial derivatives are specified using special syntax `G^(1)^a_dbc[x_a]` which is treated as $\partial_c \Gamma_{db}^a(x)$. Finally, the join operator `&` used in the last line allows to join several substitutions into a single function and `expr<<=func` is a shorthand for the `expr = func(expr)`.

At the next step we need to translate the Lagrangian into momentum space. At this point we can select which terms to include into the expansion: we will keep terms with not higher than cubic interaction (i.e. $\sim hhh$, $\sim hh\phi$, $\sim h\phi\phi$).

```

17 $Degree = { expr -> Count(expr, 1, ['h_ab[x_a]', 'f[x_a]'].t, true) }
18 $SelectCubic = { expr ->
19     (expr.class == Product && $Degree(expr) > 3) ? 0.t : expr
20 }

22 //expand all brackets and drop higher interaction terms
23 L <<= ExpandAndEliminate[$SelectCubic]
24 //perform Fourier transform
25 L <<= LagrangeFourier

```

Here we define a new function `$SelectCubic` which takes `expr` and in the case if it is a product containing more than three field multipliers it returns zero, otherwise it returns `expr` as is. The transformation `ExpandAndEliminate` just expands out all brackets and simplifies contractions with Minkowski metric. Additionally it takes an optional argument – a function (in our case `$SelectCubic`) which will be applied at each intermediate step of the expand procedure; thus all higher interactions will be dropped out at the earliest possible step, which reasonably speeds up the calculation. After applying `LagrangeFourier`, the expression will look like

```

println L
▷ h_{ab}[k_a]*h_{cd}[-k_a]*k^a*k^c*g^bd...
  + h_{ab}[k_a]*f[p_a]*f[-p_a-k_a]*p^a*k^b... + ...

```

In order to find a propagator of e.g. a graviton, we need to select quadratic part of the Lagrangian. This can be easily done using functional derivatives, e.g.:

$$P_{hh}^{(-1)abcd}(p) = \frac{\delta}{\delta h_{ab}(p)} \frac{\delta}{\delta h_{cd}(-p)} \mathcal{L}$$

```

26 $hh = L
27 $hh <<= Differentiate['h_ab[p_a]', 'h_cd[-p_a]']
28 $hh <<= 'h_ab[p_a] = 0'.t & 'f[p_a] = 0'.t
29 $hh <<= ApplyDiracDeltas & 'DiracDelta[0] = 1'.t
30 //bind inverse propagator
31 $iP = 'iP^abcd[p_a]'.eq $hh

```

Differentiate will take into account symmetries and function arguments, so e.g.:

$$\frac{\delta h_{ab}[k_a]}{\delta h^{mn}[p_a]} = \frac{1}{2} * (g_{am} * g_{bn} + g_{an} * g_{bm}) * \text{DiracDelta}[k_a, p_a]$$

ApplyDiracDeltas just removes delta-functions in an appropriate way:

$$\text{DiracDelta}[k_a, p_a] * f[k_a] * \dots = f[p_a] * \dots$$

Having the inverse propagator, the propagator is determined from the equation:

$$P_{hh}^{(-1)mnab} P_{hh}^{abcd} = \left(\delta_a^c \delta_b^d + \delta_a^d \delta_b^c \right) / 2 .$$

This tensor equation can be solved in Redberry using the *Reduce* function:

```

32 //specify symmetries of propagator
33 addSymmetries 'P_abcd[k_a]', [[0,1]].p, [[0,2], [1,3]].p
34 $eq = 'iP^abcd[k_a]*P_abpq[k_a] = (d^c_p*d^d_q + d^c_q*d^d_p)/2'.t
35 $eq <<= iP //substitute the inverse propagator in equation

37 $opts= [Transformations: 'd^~n_n = 4'.t,
38           ExternalSolver: [Solver: 'Mathematica', Path: '/usr/bin/']]
39 $hhPropagator = Reduce([$eq], ['P_abcd[k_a]'], $opts)
40 println $hhPropagator
    ▷ P_{abcd}[k_{a}] = 2*(-c+(-6*k**2*b+2*a*k**2)*k_{g}*k^{g}...

```

Our unknown variable in the equation for the propagator is $P_{abcd}[k_a]$. First we should specify symmetries which this unknown variable should satisfy. Then we call *Reduce* with the list of equations, list of unknown variables and additional options. It will make an ansatz for the unknown variable and reduce the equations to the system of scalar equations. Additional options specify that the produced scalar system should be solved with *Mathematica* (without this *Reduce* will return just a system of scalar equations). Note that the spacetime dimension is controlled everywhere just by substituting $d^{\sim}n_n = 4$. The graviton propagator finally reads:

$$\begin{aligned}
P_{abcd}(k) = & \frac{a\kappa^2 k^2 + c - 2\kappa^2 f^2}{2f^2 k^4 (a\kappa^2 k^2 + c)} (g_{ac} k_b k_d + g_{bc} k_a k_d + g_{bd} k_a k_c + g_{ad} k_b k_c) \\
& + \frac{(g_{ac} g_{bd} + g_{ad} g_{bc})}{(a\kappa^2 k^2 + c) \kappa^2 k^2} + \frac{\kappa^2 (-\kappa^2 a k^2 + c + 4b\kappa^2 k^2) g_{ab} g_{cd}}{(\kappa^2 a k^2 + c) (2\kappa^2 (a - 3b) k^2 - c) k^2} \\
& + \frac{2\kappa^4 (a - 2b) (g_{cd} k_a k_b + g_{ab} k_c k_d + 2k_a k_b k_c k_d / k^2)}{(a\kappa^2 k^2 + c) (2\kappa^2 (a - 3b) k^2 - c) k^2}
\end{aligned}$$

All other Feynman rules can be found in the same way. For example, take the $h\phi\phi$ vertex:

```

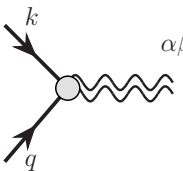
41 $hff = L
42 $hff <<= Differentiate['h_ab[k_a]', 'f[p_a]', 'f[q_a]']
43 $hff <<= 'h_ab[p_a] = 0'.t & 'f[p_a] = 0'.t
44 $hff <<= ApplyDiracDeltas

46 //bind vertex
47 $v3 = 'V3^ab[p_a, q_a, k_a]'.eq $hff
48 println $v3

  ▷ V3^ab[p_a, q_a, k_a] = (4*g-1)*p^{1}*q_{1}*g^{ab}-m**2...

```

which corresponds to



$$\begin{aligned}
 V^{ab}(p, q) = & -2g(q^a q^b + p^a p^b) + (1 - 2g)(q^a p^b + q^b p^a) + \\
 & + \left(2g(q^2 + p^2) + (4g - 1)q p^l - m^2\right) g^{ab}
 \end{aligned}$$

3. NLO processes with heavy quarkonia

Let us now turn to applications in QCD and consider the process $e^+e^- \rightarrow \gamma^* \rightarrow J/\psi + \eta_c$ at the one-loop level. In total, there are 210 Feynman diagrams which could be generated using e.g. FeynArts¹ [3]. Assuming that we already have the diagrams in some format, let's focus on the part which can be easily done with Redberry.

The basic boilerplate setup for QCD in Redberry looks like:

```

1 setAntiSymmetric 'e_abcd', 'f_ABC'
2 setSymmetric 'd_ABC'
3 defineMatrices 'T_A', Matrix2.matrix, //unitary matrices
4 'G_a', 'G5', Matrix1.matrix, //Dirac matrices
5 'v[p_a]', Matrix1.vector, Matrix2.vector, //final-state quark
6 'ubar[p_a]', Matrix1.covector, Matrix2.covector, //final-state antiquark
7 ...

```

The first lines just sets up symmetries for the Levi-Civita tensor and the SU(N) structure constants. Next, we specify rules for non-commuting objects: spinors, SU(N) and Dirac matrices. The definition Matrix1.matrix tells nothing but that G_a has two additional (invisible) matrix indices etc. In this sense e.g. the line $\bar{u}(p)T_A\gamma_a v(k)$ will be treated internally as

$$\text{ubar}[p_a]*T_A*G_a*v[k_a] \rightarrow \text{ubar}_{\{a'A'\}}[p_a]*T^{\{A'\}}_{\{B'A'\}}*G^{\{a'\}}_{\{b'a'\}}*v^{\{b'B'\}}[k_a].$$

The particular setup for our process of interest is then:

```

8 $proj = 'v[pPsi2_a]*ubar[pPsi1_a] =
9         (G^a*pPsi_a/2-MC)*G_m*epsPsi^m*(G^b*pPsi_b/2+MC)'.t & ...//onia projectors

11 $scalars = setMandelstam([k1_a: 0, k2_a: 0, pPsi_a: '2*MC', pEta_a: '2*MC'])
12 $qScalars = 'q_a*pPsi^a = qp'.t & 'q_a*pEta^a = qe'.t & 'q_a*q^a = qq'.t

14 $PaVe = PassarinoVeltman([2, 1], 'q_a', ['pPsi_a', 'pEta_a'])
15 $dSimplify = DiracSimplify[[Dimension: 'd', TraceOfOne: 4]]
16 $dTrace = DiracTrace[[Dimension: 'd', TraceOfOne: 4]]

```

¹ The details of the conversion from FeynArts can be found on the Redberry [website](#).

In the first line the projectors $v(p_1)\bar{u}(p_2) = (\hat{P}/2 - m_c)\hat{S}(\hat{P}/2 + m_c)$ of charmed quarks on the quantum numbers of the final quarkonia are specified, where $\hat{S} = \gamma_5$ for η_c and $\hat{S} = \hat{\epsilon}_\psi$ for the J/ψ and P is the total momentum of the quarkonia. Next, the function `setMandelstam` generates a list of replacements for the scalar products of momenta. The variable `$qScalars` holds auxiliary replacements of dot products with loop momenta, while `PassarinoVeltman(i, in, ext)` returns a list of replacements for loop momenta `in` and list of external momenta `ext`, e.g.:

```
println PassarinoVeltman(2, 'q_a', ['pPsi_a', 'pEta_a'])
▷ q_a*q_b = (...)*pPsi_a*pEta_b + (...)*g_ab + ...
```

Finally, `DiracSimplify` and `DiracTrace` are used with additional options in order to perform all algebra in dimensional regularization (Larin's scheme [4] for γ_5 is implied automatically).

After these preparations, the main calculation loop can be the following:

```
17 $amps = __CallFeynArts__.t //import diagrams from FeynArts
18 $answNLO = 0.t
19 for($amp in $amps){
20     $amp <<= $proj //apply quarkonia projectors
21     $amp <<= UnitaryTrace & UnitarySimplify // SU(N) algebra
22     $amp <<= $qScalars & $PaVe & ExpandAndEliminate // PaVe reduction
23     $amp <<= $dSimplify & $dTrace & LeviCivitaSimplify // dimensional regularization
24     $answNLO += Factor($amp) //sum amplitudes
25 }
```

We just iterate through all diagrams and sequentially do: substitutions of projectors, color algebra, Passarino-Veltman reduction and dimensional regularization. After all, the amplitudes `$answNLO` will be in the form:

$$(\dots) * e_{\{abcd\}} * p_{\Psi}^b * p_{\text{Eta}}^c * \epsilon_{\Psi}^d + (\text{scalar}) * (\text{tensor}) + \dots,$$

where all same Lorentz structures will be collected automatically and each scalar prefactor is a sum of scalar loop integrals that should be either directly calculated numerically or reduced to a set of master integrals. Presently, the latter step is deferred to external tools. In the case of the above example, it is easy to export processed amplitudes in e.g. `Mathematica` format with

```
new File('amps_nlo.m') << $answNLO.toString(OutputFormat.WolframMathematica)
```

and reduce the scalar integrals with e.g. the `$Apart` package [5] and `FIRE` [6]. The whole reduction of the above 210 diagrams with `Redberry` takes just about two minutes on a standard laptop.

4. Conclusions

In this paper we have illustrated some new important features of `Redberry` system, which facilitates automatization of calculations of many types required in HEP. More examples with many details can be found on the `Redberry` website <http://redberry.cc> [1].

Acknowledgments

I would like to thank Andrei Kataev and Andrei Onishchenko for stimulating discussions. The work was supported by the RFBR grant #16-32-60017 and by the FAIR-Russia Research Center.

References

- [1] Bolotin D A and Poslavsky S V 2013 (*Preprint* [1302.1219](https://arxiv.org/abs/1302.1219)) URL <http://redberry.cc>
- [2] Poslavsky S and Bolotin D 2015 *J. Phys. Conf. Ser.* **608** 012060
- [3] Hahn T 2001 *Comput. Phys. Commun.* **140** 418–431 (*Preprint* [hep-ph/0012260](https://arxiv.org/abs/hep-ph/0012260))
- [4] Larin S A 1993 *Phys. Lett.* **B303** 113–118 (*Preprint* [hep-ph/9302240](https://arxiv.org/abs/hep-ph/9302240))
- [5] Feng F 2012 *Comput. Phys. Commun.* **183** 2158–2164 (*Preprint* [1204.2314](https://arxiv.org/abs/1204.2314))
- [6] Smirnov A V 2008 *JHEP* **10** 107 (*Preprint* [0807.3243](https://arxiv.org/abs/0807.3243))