runDM v1.0 - Manual

May 4, 2016

Contents

1 Overview

The `runDM` code is a tool for calculating the low energy couplings of Dark Matter (DM) to the Standard Model (SM) in Simplified Models with vector mediators. By specifying the mass of the mediator and the couplings of the mediator to SM fields at high energy, the code outputs the couplings at a different energy scale, fully taking into account the mixing of all dimension-6 operators. Further details about the physics behind the code can be found in Appendix B of arXiv:1605.XXXXX.

At present, the code is written in two languages: *Mathematica* and *Python*. For installation instructions and example code in each language, skip straight to Sec. 3. Example files/notebooks for all versions of the code (along with the code itself) are available at: https://github.com/bradkav/runDM/releases/latest. If you are interested in an implementation in another language, please get in touch and we'll do what we can to add it. Please contact Bradley Kavanagh (bradkav@gmail.com) for any questions, problems, bugs and suggestions.

If you make use of `runDM` in your work, please cite it as:

F. D'Eramo, B. J. Kavanagh & P. Panci (2016). *runDM* (Version X.X) [Computer software]. Available at https://github.com/bradkav/runDM/ , making sure to include the correct version number. Please also cite the associated papers:

A. Crivellin, F. D'Eramo & M. Procura, *New Constraints on Dark Matter Effective Theories from Standard Model Loops*, Phys. Rev. Lett. **112** (2014) 191304 [arXiv:1402.1173 [hep-ph]].

F. D'Eramo & M. Procura, *Connecting Dark Matter UV Complete Models to Direct Detection Rates via Effective Field Theory*, JHEP **1504** (2015) 054 [arXiv:1411.3342 [hep-ph]] ,

F. D'Eramo, B. J. Kavanagh & P. Panci, *You can hide but you have to run: direct detection with vector mediators*, (2016) [arXiv:1605.XXXXX] .

2 General framework

This section describes the general framework of `runDM`, describing the general usage, inputs and outputs of `runDM`, as well as pseudocode for how to use the most important functions. For implementation-specific information, please see Sec. 3.

The core of `runDM` is the function `runCouplings`($\mathbf{c}$, E_1, E_2). This function accepts as input a vector of couplings $\mathbf{c}$, specified at some energy E_1. It returns a different vector of couplings, evaluated at some other energy E_2, taking into account RG evolution between the two energy scales. Full details about the numerical implementation can be found in Appendix B of arXiv:1605.XXXXX. Here, we simply outline which coupling values are required as input and output.

The input vector $\mathbf{c}$ has 16 elements, the coefficients of the dimension-6 DM-SM operators of the form: $\mathcal{O}_{\mathrm{DM},\mu}\,\mathcal{O}^{\mu}_{\mathrm{SM}}$. The structure of the DM operator does not affect the running of the coefficients and may take the following forms for fermionic DM χ:

$$\mathcal{O}_{\mathrm{DM},\mu} = \overline{\chi}\gamma_\mu\chi\,,\ \overline{\chi}\gamma_\mu\gamma^5\chi\,, \tag{1}$$

and for complex scalar DM ϕ:

$$\mathcal{O}_{\mathrm{DM},\mu} = \phi^\dagger \overleftrightarrow{\partial}_\mu \phi\,,\ \partial_\mu(\phi^\dagger\phi)\,. \tag{2}$$

The SM operators which appear at dimension-6, defined above the electroweak symmetry breaking (EWSB) scale, are:

$$\begin{aligned}
\mathcal{O}^{\mu}_{q^{(i)}} &= \overline{q}^{(i)}_L\gamma^\mu q^{(i)}_L\,, & \mathcal{O}^{\mu}_{l^{(i)}} &= \overline{l}^{(i)}_L\gamma^\mu l^{(i)}_L\,,\\
\mathcal{O}^{\mu}_{u^{(i)}} &= \overline{u}^{(i)}_R\gamma^\mu u^{(i)}_R\,, & \mathcal{O}^{\mu}_{e^{(i)}} &= \overline{e}^{(i)}_R\gamma^\mu e^{(i)}_R\,,\\
\mathcal{O}^{\mu}_{d^{(i)}} &= \overline{d}^{(i)}_R\gamma^\mu d^{(i)}_R\,, & \mathcal{O}^{\mu}_{H} &= iH^\dagger\overleftrightarrow{D}_\mu H\,,
\end{aligned} \tag{3}$$

where the (i) superscript labels the generation number. The vector of couplings which `runDM` accepts as input is then defined as

$$\mathbf{c} = \begin{pmatrix} c^{(1)}_q & c^{(1)}_u & c^{(1)}_d & c^{(1)}_l & c^{(1)}_e & c^{(2)}_q & c^{(2)}_u & c^{(2)}_d & c^{(2)}_l & c^{(2)}_e & c^{(3)}_q & c^{(3)}_u & c^{(3)}_d & c^{(3)}_l & c^{(3)}_e & c_H \end{pmatrix}. \tag{4}$$

A number of functions are available to initialise these coupling vectors - see Sec. 2.1.

If the energy E_2 at which to evaluate the couplings is above the EWSB scale (i.e. $E_2 > m_Z$), the output of `runCouplings` will be a vector of couplings as in Eq. 4, evaluated after RG evolution. However, if E_2 is below the EWSB scale (i.e. $E_2 < m_Z$), then `runCouplings` will return the vector $\mathcal{C}$, which are the couplings to the SM operators after EWSB. These operators are

$$\begin{aligned} \mathcal{O}^{\mu}_{Vu^{(i)}} &= \overline{u}^{(i)}\gamma^{\mu}u^{(i)}\,, & \mathcal{O}^{\mu}_{Au^{(i)}} &= \overline{u}^{(i)}\gamma^{\mu}\gamma^5 u^{(i)}\,, \\ \mathcal{O}^{\mu}_{Vd^{(i)}} &= \overline{d}^{(i)}\gamma^{\mu}d^{(i)}\,, & \mathcal{O}^{\mu}_{Ad^{(i)}} &= \overline{d}^{(i)}\gamma^{\mu}\gamma^5 d^{(i)}\,, \\ \mathcal{O}^{\mu}_{Ve^{(i)}} &= \overline{e}^{(i)}\gamma^{\mu}e^{(i)}\,, & \mathcal{O}^{\mu}_{Ae^{(i)}} &= \overline{e}^{(i)}\gamma^{\mu}\gamma^5 e^{(i)}\,, \end{aligned} \tag{5}$$

and the corresponding coupling vector is

$$\mathcal{C} = \left(\begin{matrix} \mathcal{C}^{(1)}_{Vu} & \mathcal{C}^{(1)}_{Vd} & \mathcal{C}^{(2)}_{Vu} & \mathcal{C}^{(2)}_{Vd} & \mathcal{C}^{(3)}_{Vd} & \mathcal{C}^{(1)}_{Ve} & \mathcal{C}^{(2)}_{Ve} & \mathcal{C}^{(3)}_{Ve} & \mathcal{C}^{(1)}_{Au} & \mathcal{C}^{(1)}_{Ad} & \mathcal{C}^{(2)}_{Au} & \mathcal{C}^{(2)}_{Ad} & \mathcal{C}^{(3)}_{Ad} & \mathcal{C}^{(1)}_{Ae} & \mathcal{C}^{(2)}_{Ae} & \mathcal{C}^{(3)}_{Ae} \end{matrix}\right). \tag{6}$$

The `runDM` code takes care of the matching between between **c** and $\mathcal{C}$ at the EWSB-scale. It also takes care of relative values of E_1 and E_2 compared to each other and to m_Z. The only possibility which is not allowed is to have $E_1 < m_Z$ and $E_2 > m_Z$. This is because the matching from the broken to the unbroken phase of the SM is not unique.

In the following subsections, we give more details on the key functions available in `runDM`, including `DDCoupling` (Sec. 2.3) which allows the user to directly calculate the couplings relevant at the direct detection scale.

2.1 Initialisation

The input coupling vector **c** must be in the form of an array with 16 elements. To help with initialisation, `runDM` includes the function `initCouplings()`, which returns such an array, filled with zeroes. The user is then free to specify each of the 16 couplings in Eq. 4.

Alternatively, `setBenchmark(benchmarkID)` returns a coupling vector **c** corresponding to one of a number of preset benchmarks, specified using the string `benchmarkID`. The available benchmarks are:

- `"Higgs"` - coupling only to the Higgs current operator:

  $c_H = 1$, all other couplings zero.

- `"UniversalVector"` - universal vector coupling to all fermions:

  $c_q^{(i)} = c_u^{(i)} = c_d^{(i)} = c_l^{(i)} = c_e^{(i)} = 1$, $c_H = 0$.

- `"UniversalAxial"` - universal axial-vector coupling to all fermions:

  $-c_q^{(i)} = c_u^{(i)} = c_d^{(i)} = -c_l^{(i)} = c_e^{(i)} = 1$, $c_H = 0$.

- `"QuarksVector"` - vector coupling to all quarks:
  $c_q^{(i)} = c_u^{(i)} = c_d^{(i)} = 1$, $c_l^{(i)} = c_e^{(i)} = c_H = 0$.
- `"QuarksAxial"` - axial-vector coupling to all quarks:
  $-c_q^{(i)} = c_u^{(i)} = c_d^{(i)} = 1$, $c_l^{(i)} = c_e^{(i)} = c_H = 0$.
- `"LeptonsVector"` - vector coupling to all leptons:
  $c_l^{(i)} = c_e^{(i)} = 1$, $c_q^{(i)} = c_u^{(i)} = c_d^{(i)} = c_H = 0$.
- `"LeptonsAxial"` - axial-vector coupling to all leptons:
  $-c_l^{(i)} = c_e^{(i)} = 1$, $c_q^{(i)} = c_u^{(i)} = c_d^{(i)} = c_H = 0$.
- `"ThirdVector"` - vector coupling to third generation fermions:
  $c_q^{(3)} = c_u^{(3)} = c_d^{(3)} = c_l^{(3)} = c_e^{(3)} = 1$, all remaining couplings zero.
- `"ThirdAxial"` - axial-vector coupling to third generation fermions:
  $-c_q^{(3)} = c_u^{(3)} = c_d^{(3)} = -c_l^{(3)} = c_e^{(3)} = 1$, all remaining couplings zero.

If the value of `benchmarkID` is not recognised, `setBenchmark` simply returns a coupling vector filled with zeroes.

2.2 runCouplings

The function `runCouplings(`$\mathbf{c}$, E_1, E_2`)` accepts as input a vector of couplings $\mathbf{c}$, defined at energy E_1 (in GeV) and returns a vector of couplings evaluated at energy E_2 (in GeV), taking into account the running, matching and mixing between the two energies. The energies E_1 and E_2 must be in the range $[1, 10^8]$ GeV (the evolution matrices have not been calculated outside of that range). The input vector $\mathbf{c}$ has the form of Eq. 4.

If $E_2 > m_Z$, `runCouplings` returns the elements of $\mathbf{c}$, the vector of couplings above the EWSB scale defined in Eq. 4. Alternatively, if $E_2 < m_Z$, `runCouplings` returns the elements of $\mathcal{C}$, the vector of couplings in the EFT below the EWSB scale, defined in Eq. 6. Note that `runCoupling` will return an error if $E_1 < m_Z$ and $E_2 > m_Z$ - in this case the matching at the EWSB-scale is not unique.

The function `runCouplings` is *vectorized*, meaning that it will accept E_1 and E_2 as arrays as well as single numbers. `runCouplings` will happily deal with one of E_1 and E_2 being an array and the other being a scalar. The output of *runCouplings* will be an array with dimensions {Length(E_1), 16} or {Length(E_2), 16}, depending on which of E_1 and E_2 is an array. If both E_1 and E_2 are arrays, they should have the same length and *runCouplings* will return an array with dimensions {Length(E_1), 16}. The function is *not vectorized* in $\mathbf{c}$ - i.e. $\mathbf{c}$ should always be a 1-dimensional array with 16 elements.

2.3 DDCouplings

The function `DDCouplings`($\mathbf{c}$, E_1) accepts as input a vector of couplings $\mathbf{c}$, defined at energy E_1 (in GeV). The input vector $\mathbf{c}$ has the form of Eq. 4. The running is performed down to the nuclear energy scale of 1 GeV and the function returns a vector of the low-energy couplings to light quarks (u, d, s) relevant for direct detection:

$$\mathcal{C}_q = \left(\mathcal{C}_V^{(u)}, \mathcal{C}_V^{(d)}, \mathcal{C}_A^{(u)}, \mathcal{C}_A^{(d)}, \mathcal{C}_A^{(s)}\right) . \tag{7}$$

As in `runCouplings`, E_1 must be in the range $[1, 10^8]$ GeV. `DDCouplings` is also vectorized, so it will accept E_1 as a scalar or an array. It will return an array with dimensions {Length(E_1), 5}.

3 Implementations

3.1 Mathematica

In order to use the *mathematica* implementation of `runDM`, simply copy the *mathematica* package file `mathematica/runDM.m` into the same directory as your notebook and then use `Get`:

```
Get[NotebookDirectory[] <> "runDM.m"];
```

Below is an example code snippet for calculating (and plotting) the low energy couplings to light quarks (relevant in direct detection experiments) at the nuclear energy scale, assuming a mediator of mass m_V which couples through the axial-vector current to all SM quarks:

```
Get[NotebookDirectory[] <> "runDM.m"];

(*Set benchmark*)
chigh = setBenchmark["QuarksAxial"];

clabels = {"cVu", "cVd", "cAu", "cAd", "cAs"};

(*Calculate and plot*)
Table[Plot[{DDCouplings[chigh, 10^lmv][[k]]}, {lmv, 0.0, 6.0},
  PlotTheme -> "Scientific",
  FrameLabel -> {"Log10[mv/GeV]", clabels[[k]]},
  PlotRangePadding -> .1, ImageSize -> 200,
  BaseStyle -> {FontSize -> 16}], {k, 5}]
```

For more detailed examples, see the notebook at `mathematica/runDM-examples.nb`.

3.2 Python

In order to use the *python* implementation of `runDM`, you will need to have the `numpy` and `scipy` packages installed. Once that's done, copy the module file `python/runDM.py` into the same directory as your python script. Make sure that the folder `python/data/` is in the same folder as the module file. Then simply import the module:

```python
import runDM
```

Below is an example code snippet for calculating (and plotting) the low energy couplings to light quarks (relevant in direct detection experiments) at the nuclear energy scale, assuming a mediator of mass m_V which couples through the axial-vector current to all SM quarks:

```python
import numpy as np
from matplotlib import pyplot as pl
import runDM

#Set benchmark
c_high = runDM.setBenchmark("QuarksAxial")

#Calculate the low energy couplings
mV = np.logspace(0, 6, 1000)
c_q = runDM.DDCouplings(c_high, mV)

#Now let's do some plotting
f, axarr = pl.subplots(5,figsize=(5,9))

clabels = ['c_V^u','c_V^d','c_A^u','c_A^d','c_A^s']

for k in range(5):
    ax = axarr[k]
    ax.semilogx(mV, c_q[:,k])
    ax.set_xlabel(r'$m_V$ [GeV]', fontsize=14.0)
    ax.set_ylabel(r'$'+clabels[k]+'$', fontsize=14.0)

pl.tight_layout()
pl.show()
```

For more detailed examples, see the script at `python/runDM-examples.py` or the ipython notebook at `python/runDM-examples.ipynb`.