Vector Spaces for Dark Matter (VSDM): Fast Direct Detection Calculations with Python and Julia

Benjamin Lillard[1]⋆ and Aria Radick[1,2]†

1 Institute for Fundamental Science, University of Oregon, Eugene, OR 97401, USA
2 Department of Physics, University of Oregon, Eugene, OR 97401, USA

⋆ blillard@uoregon.edu , † aradick@uoregon.edu

Abstract

Anisotropic target materials are promising candidates for dark matter direct detection experiments, providing a directional sensitivity that can be used to distinguish a dark matter (DM) signal from the various Standard Model backgrounds. In this paper we introduce the Julia and Python implementations of *Vector Spaces for Dark Matter* (VSDM), which handle the difficult scattering rate computation for these rotating, three-dimensional response functions by calculating a partial rate matrix for every combination of DM velocity distribution, material response function, and particle DM properties.

1   Introduction

In the field of dark matter direct detection, a need has arisen for a much faster scattering rate calculation. Refs. [1, 2] solve this problem by projecting the dark matter (DM) velocity distribution and the Standard Model (SM) detector response function onto vector spaces spanned by orthogonal wavelet-harmonic basis functions. Taking advantage of the symmetries of the DM–SM scattering operator, this vector space integration method calculates the scattering rate using a relatively small number of vector products, reducing the evaluation times for the hardest analyses by up to a factor of 10^8 [2]. In this paper, we present two numerical implementations of this method for public use: VectorSpaceDarkMatter, a Julia package, and vsdm, in Python. The two packages are designed with complementary priorities: the Python version emphasizes flexibility in the intermediate stages of the calculation, while the Julia package is designed for speed and simplicity.

If you, the reader, have found yourself spending too much of your time calculating nonrelativistic scattering rates, then there is a good chance that VSDM will be useful to you. By representing the input functions on an orthogonal basis of functions, and performing the difficult integrals using the basis functions instead, the difficult pieces of an analysis can be factorized from each other. This is particularly helpful when there are multiple versions of each input function. Due to some properties of the spherical harmonics outlined in [1], the event rate as

a function of detector orientation can be entirely represented by an object called the *partial rate matrix*. The main purpose of VSDM is to evaluate this object, and to use it to efficiently calculate the scattering rate.

The Julia version can be installed by opening a Julia REPL and typing

```
using Pkg
Pkg.add("VectorSpaceDarkMatter")
```

The Python version can be installed from PyPI via the command:

```
pip install vsdm
```

After Section 1.1, which provides a review of the scaling problem that VSDM is designed to solve, the remainder of this section is devoted to reviewing the standard literature on direct detection scattering rates. Sections 1.2–1.3 present the expression for the DM–SM scattering rate, and its dependence on the details of the DM particle model. Section 1.4 discusses the detector response function in the contexts of DM–nucleus, DM–electron, and DM–phonon scattering. Because several different notations are currently in use, we show explicitly how to convert each response function into our notation. Section 1.5 introduces the DM velocity distribution, the Standard Halo Model, and the integrated velocity distribution or "inverse average speed" η. Finally, Section 1.6 describes a few effects (like energy-dependent efficiency factors in the material or detector) that VSDM does not include, and that the user will have to implement themselves.

Section 2 introduces the vector space version of the rate calculation, largely following the notation of [2], but taking care to match the conventions used in the VSDM packages for all of the intermediate steps. Sections 3 and 4 document the Julia and Python packages, respectively, providing examples of how to complete the scattering calculation in each case. Section 5 compares the evaluation times for the two packages, at each stage of the calculation. The most time-consuming parts of the VSDM calculation are faster by a factor of 10^2 in the Julia version, making it the best choice for analyses with multiple momentum or velocity distributions, or detailed scans over the particle DM parameter space. Once the partial rate matrix of [1] has been evaluated, both versions of the code are extremely efficient at calculating the scattering rate as a function of detector orientation.

Units: Both implementations of the code use units of $\hbar = c = 1$. In this paper, we include the factors of c in relevant equations. The Python implementation vsdm uses $c = 1$ as the default option, but it is possible to change the fundamental velocity unit to something else (e.g. to km/s).

1.1 The Problem That VSDM Solves

Except for the very simplest detector systems, predicting the DM scattering rate in a detector requires repeated numerical integration. This is especially challenging for directionally sensitive target materials [3–15], but also isotropic semiconductor [16–20] and molecular targets [21–23]. Not only does the rate integrand depend on the details of the DM model (its mass, and the nature of its couplings to the SM), but it also depends on inputs from astrophysics and condensed matter or physical chemistry, all of which are generally subject to systematic uncertainties [24–29]. The scattering rate in an anisotropic detector is a function of the detector's orientation with respect to the DM velocity distribution, which is itself a function of time, varying annually according to the Earth velocity around the Sun (see e.g. [30] for a review). Taken together, a careful analysis may need to perform the rate calculation

$$N_{\text{calcs}} = N_{\mathcal{R}} N_{\text{DM}} N_{g_\chi} N_{f_S} \tag{1.1}$$
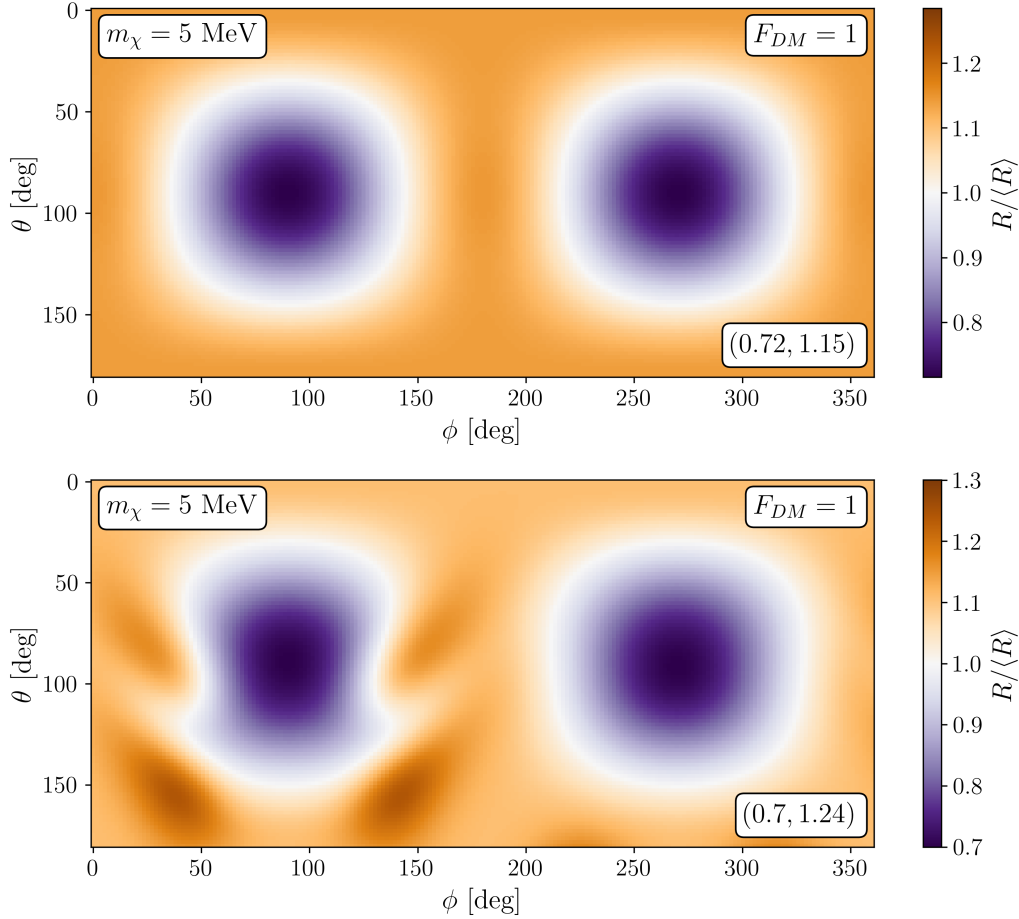
Figure 1: These two plots show a type of analysis that is easy with VSDM, and challenging with traditional methods. Each plot required a scan over 10^4 detector orientations: with VSDM this is effectively instantaneous. We show the scattering rate R for a set of detector orientations parameterized by θ and ϕ, the direction of the Earth velocity vector relative to the detector in spherical coordinates: $\vec{v}_E = (v_E, \theta, \phi)$. Both plots use the same material response function (from from [2]) and DM model (a 5 MeV DM candidate with a heavy mediator form factor, $F_{DM} = 1$), but the velocity distributions are different. The upper plot assumes the Standard Halo Model, while the lower plot takes 5% of the DM to be in an off-axis stream with relative speed of 238 km/s, with the SHM comprising the remaining 95% of the distribution. The bottom right hand corner of each figure denotes the minimum and maximum values of R divided by the angular average rate $\langle R \rangle$.

many times, where N_{DM} is the number of DM particle models, specifying its mass and its SM couplings; N_{g_χ} counts the number of distinct lab-frame DM velocity distributions, e.g. to account for its time dependence and its systematic uncertainties; N_{f_S} counts how many detector models are included in the analysis, e.g. including multiple target materials, or cases where several different SM final states contribute to the same experimental observable; and $N_{\mathcal{R}}$ is the number of detector orientations. For an isotropic detector material (e.g. unpolarized liquids or gases), the rate does not depend on the detector orientation, and $N_{\mathcal{R}} = 1$.

Figure 1 shows a relatively straightforward application of VSDM: predicting the scattering rate as a function of detector orientation, for two different velocity distributions. Both figures assume the material form factor for a particle-in-a-box from [2],

$$f_s^2(\vec{q}) = \prod_{j=x,y,z} \left[\frac{\mathrm{sinc}\left(\frac{|q_j L_j| - \pi(n_j - 1)}{2}\right)}{1 + \pi(n_j - 1)/|q_j L_j|} + \frac{\mathrm{sinc}\left(\frac{|q_j L_j| - \pi(n_j + 1)}{2}\right)}{1 + \pi(n_j + 1)/|q_j L_j|} \right]^2 \tag{1.2}$$

where L_j is the length of the side in the j-direction with $(L_x, L_y, L_z) = (4, 7, 10)\, a_0$ (where a_0 is the Bohr radius), and n_j is the excitation mode in the j-direction. This figure specifically looks at the $n_x = n_y = 1, n_z = 2$ mode, the lowest energy excited mode with an excitation energy of $\Delta E = 4.03$ eV.

The top panel of Figure 1 assumes that 100% of the DM velocity distribution follows the Standard Halo Model (see (1.42)), while in the bottom panel a 5% fraction of the DM is moved into a low-dispersion stream, modeled by a Gaussian with parameters

$$\vec{\mu} = (v, \theta, \phi) = (238 \text{ km/s}, \pi/4, -\pi/2) \tag{1.3}$$
$$\sigma = 23.3 \text{ km/s} \tag{1.4}$$

In both cases the DM mass is taken as $m_\chi = 5$ MeV, with a heavy mediator for the DM–SM scattering ($F_{\mathrm{DM}} = 1$). Each plot shows the scattering rate $R(\theta, \phi)$ for a two-dimensional slice of the possible $\mathcal{R} \in SO(3)$ detector orientations, with (θ, ϕ) indicating the direction of the Earth velocity $\vec{v}_E$ in spherical coordinates relative to the detector. For the azimuthally symmetric SHM this is a complete description of $R(\mathcal{R})$. In the second example, R is also a function of the rotation angle β about the $\vec{v}_E$ axis, not shown in the picture: a 3d scan over (θ, ϕ, β) would be required for a complete study of this system.

Even for this 2d subset of $SO(3)$, each plot required a scan over 10^4 different detector rotations. With traditional methods this would ordinarily require evaluating 10^4 numeric integrals, but VSDM reduces this to 10^4 dot products involving the partial rate matrix, vastly reducing the computation time. Repeating this analysis for other DM models ($N_{\mathrm{DM}} \gg 1$), other excited states for the detector, and other parameters for the DM stream, it is easy to see how N_{calcs} of (1.1) can grow too fast for the traditional methods to handle.

1.2 Review: Calculating the DM–SM Scattering Rate

In terms of the free particle DM–SM scattering cross section, $\sigma(\vec{q}, \vec{v})$, as a function of momentum transfer $\vec{q}$ and DM velocity $\vec{v}$ in the nonrelativistic limit; the number of SM target "particles" (e.g. atoms, molecules, or unit cells of a crystal) N_T; the DM mass m_χ and local density ρ_χ; and the reduced mass of the DM–SM system, μ_χ, the differential scattering rate as a function of deposited energy E is:

$$\left. \frac{dR}{dE} \right|_{E, \mathcal{R}} = N_T \frac{\rho_\chi}{m_\chi} \int d^3 q\, d^3 v\, g_\chi(\vec{v}, t)\, \delta\left(E + \frac{q^2}{2m_\chi} - \vec{q} \cdot \vec{v}\right) \frac{\sigma(\vec{q}, \vec{v})}{4\pi \mu_\chi^2} \mathcal{R} \cdot f_S^2(\vec{q}, E). \tag{1.5}$$

Here $g_\chi(\vec{v}, t)$ is the lab frame DM velocity distribution, normalized according to

$$\int d^3 v \, g_\chi(\vec{v}) \equiv 1. \tag{1.6}$$

For a terrestrial experiment, or any other lab that orbits the Sun, the lab frame $g_\chi(\vec{v})$ is time-dependent, varying over the course of the year. The momentum form factor $f_S^2(\vec{q}, E)$ is our general-purpose notation for the material response function. It has units of inverse energy. The rate R has units of events per unit time, and it is a function of the detector orientation $\mathcal{R} \in SO(3)$, written in (1.5) as a rotation operator acting on the detector form factor.

For transitions from the ground state (g) to states with discrete final energies, labeled by $E = \Delta E_s$ for some set of excited states s, the $g \to s$ scattering rate is:

$$R_{g \to s}(\mathcal{R}) = N_T \frac{\rho_\chi}{m_\chi} \int d^3q \, d^3v \, g_\chi(\vec{v}, t) \, \delta\left(\Delta E_s + \frac{q^2}{2m_\chi} - \vec{q} \cdot \vec{v}\right) \frac{\sigma(\vec{q}, \vec{v})}{4\pi \mu_\chi^2} \mathcal{R} \cdot f_{g \to s}^2(\vec{q}), \tag{1.7}$$

with a different dimensionless material form factor $f_{g \to s}^2(\vec{q})$ for each excited state s.

The total scattering rate R is given by integrating (1.5) over energy,

$$R(\mathcal{R}) = \int dE \frac{dR}{dE}\bigg|_{\mathcal{R}}, \tag{1.8}$$

or by summing (1.7) over all of the relevant excited states:

$$R(\mathcal{R}) = \sum_s R_{g \to s}(\mathcal{R}). \tag{1.9}$$

In an idealized, perfectly efficient detector, the expected signal rate is identical to the DM–SM scattering rate R. A more realistic model of the detector adjusts R by some efficiency factor, which may be a function of recoil energy E or the location of the scattering event inside the target material, to account for the attenuation of the signal as it propagates through the detector systems. VSDM does not attempt to account for these effects: it simply calculates the DM–SM scattering rates as presented in (1.5) and (1.7).

Kinematics: A simple fact about the kinematics of direct detection can be gleaned from the energy-conserving δ function: in order to deposit energy E into the detector, there must be some part of the phase space where $\vec{q} \cdot \vec{v} = E + q^2/(2m_\chi)$. For a given value of the momentum transfer q, the smallest $|\vec{v}|$ that satisfies this constraint is:

$$v_{\min}(q, E) \equiv \frac{E}{q} + \frac{q}{2m_\chi}. \tag{1.10}$$

This quantity turns out to be highly useful when simplifying the rate integrand. For fixed E but varying q, the minimum possible value of $v_{\min}$ occurs when $E/q = q/(2m_\chi)$: we label this special point as

$$q_\star(E) \equiv \sqrt{2m_\chi E}, \qquad\qquad v_\star(E) \equiv \frac{q_\star}{m_\chi} = \min(v_{\min}(q, E)). \tag{1.11}$$

This $v_\star(E)$ is related in turn to the lower limit of a detector's sensitivity to light DM masses. Let $E_{\min}$ be the lowest possible energy where the detector can still detect scattering events, and let $v_{\max}$ be the largest velocity where the lab frame velocity distribution $g_\chi(\vec{v})$ has support. If $v_\star(E_{\min}) > v_{\max}$, then the scattering rate is identically zero. This occurs for all DM masses below some threshold, $m_{\min}$, defined as

$$m_{\min}(v_{\max}) = \frac{2E_{\min}}{v_{\max}^2}. \tag{1.12}$$

For m_χ only slightly above this threshold, the event rate is highly suppressed: the only part of $g_\chi(\vec{v})$ that can contribute to the scattering must lie within the interval $v_\star(E) \leq v \leq v_{\min}$.

Rotations and Quaternions: The rotation operator $\mathcal{R} \in SO(3)$ is described here as an active rotation operator that acts on the detector response function. This is equivalent to acting on the momentum variable with the operator $\mathcal{R}^{-1}$:

$$\mathcal{R} \cdot f_S^2(\vec{q}, E) = f_S^2(\mathcal{R}^{-1}\vec{q}, E). \tag{1.13}$$

Because it is the relative orientation of the detector and the DM velocity distribution that matters, in some cases it is easier to use a coordinate frame where the detector material is held fixed, and the operator $\mathcal{R}^{-1}$ acts on $g_\chi(\vec{v})$:

$$\mathcal{R}^{-1} \cdot g_\chi(\vec{v}) = g_\chi(\mathcal{R}\vec{v}). \tag{1.14}$$

VSDM and one of its dependencies, spherical (SphericalFunctions.jl in the Julia version), specify elements of $SO(3)$ using quaternions. In this language a 3d Cartesian vector $\vec{u}$ is represented by the imaginary quaternion $\vec{u} = u_x \hat{i} + u_y \hat{j} + u_z \hat{k}$, and every unit quaternion Q maps onto a rotation operator $\mathcal{R}$. Any $\mathcal{R} \in SO(3)$ can be described as a (right-handed) rotation by an angle β about an axis $\hat{n}$ (a unit vector). In this notation, the mapping between Q and $\mathcal{R}$ is given by:

$$Q(\mathcal{R}) = \cos\frac{\beta}{2} + \sin\frac{\beta}{2}\hat{n}. \tag{1.15}$$

where $\hat{n}$ is an imaginary unit quaternion, and β is an angle $0 \leq \beta < 2\pi$. The action of $\mathcal{R}$ on a vector $\vec{u}$ is given by the quaternionic multiplication rule, $\hat{i}^2 = \hat{j}^2 = \hat{k}^2 = \hat{i}\hat{j}\hat{k} = -1$:

$$\mathcal{R} \cdot \vec{u} = Q^{-1}\vec{u}Q, \tag{1.16}$$

and the composition of multiple rotations is given by the product of their quaternions:

$$\mathcal{R}_2 \cdot (\mathcal{R}_1 \cdot \vec{u}) = Q_2^{-1}Q_1^{-1}\vec{u}Q_1Q_2 = (Q_1Q_2)^{-1}\vec{u}(Q_1Q_2). \tag{1.17}$$

Note that Q and $-Q$ encode identical rotations, and that for unit quaternions the inverse Q^{-1} is also the complex conjugate of Q,

$$Q^\star(\mathcal{R}) = \cos\frac{\beta}{2} - \sin\frac{\beta}{2}\hat{n}, \qquad\qquad Q^\star Q = 1. \tag{1.18}$$

Compared to 3×3 matrices, the quaternion representation is more compact and easier to interpret: i.e. β and $\hat{n}$ can be extracted directly from the real and imaginary parts of Q, respectively, while obtaining this information from a 3×3 matrix (or a set of Euler angles) is much more tedious.

1.3 DM Model Form Factor

For nonrelativistic DM–SM scattering, the cross section $\sigma(\vec{q}, \vec{v})$ is a function of Galilean invariants: $\vec{q}$, the momentum transfer; the DM and SM spins, $\vec{S}_\chi$ and $\vec{S}_i$; and a velocity quantity $\vec{v}^\perp$ [31]:

$$\vec{v}^\perp \equiv \vec{v} + \frac{\vec{q}}{2\mu_\chi}, \tag{1.19}$$

where μ_χ is the reduced mass of the DM–SM system and $\vec{v}$ is the velocity of the DM particle in the lab rest frame. Refs. [31, 32] have enumerated complete sets of DM–SM couplings in the limit of nonrelativistic DM–nucleon and DM–electron scattering. Adapting the notation

of [33], the free particle cross section can be written in terms of a constant, $\bar{\sigma}_0$, and a velocity- and momentum-dependent form factor, $F^2_{\text{DM}}(\vec{q}, \vec{v})$:

$$\sigma(\vec{q}, \vec{v}) = \frac{\mu_\chi^2}{16\pi m_\chi^2 m_{\text{SM}}^2} \overline{|\mathcal{M}_{\chi,\text{SM}}(\vec{q}, \vec{v})|^2} \tag{1.20}$$

$$\equiv \bar{\sigma}_0 \cdot F^2_{\text{DM}}(\vec{q}, \vec{v}). \tag{1.21}$$

Here $\overline{|\mathcal{M}|^2}$ is the spin-averaged squared amplitude for free DM–SM scattering (i.e. where the DM and SM incoming and outgoing states are eigenstates of momentum), and F^2_{DM} is defined such that $F^2_{\text{DM}}(\vec{q}_{\text{ref}}, \vec{v}_{\text{ref}}) \equiv 1$ at some reference momentum and velocity, $\vec{q}_{\text{ref}}$ and $\vec{v}_{\text{ref}}$. For nuclear scattering, if $q_{\text{ref}} \to 0$ does not produce a well-defined $\bar{\sigma}_0$, then the characteristic lab-frame momentum of an incoming DM particle provides a natural scale: $q_{\text{ref}} = m_\chi v_0$, where v_0 is the local galactic rotation speed. In DM–electron scattering, the Bohr momentum $q_{\text{ref}} = \alpha m_e c$ is a common choice for q_{ref}.

With the F^2_{DM} notation, (1.5) becomes:

$$\frac{dR}{dE}\bigg|_{E,\mathcal{R}} = N_T \rho_\chi \bar{\sigma}_0 \int d^3q\, d^3v\, g_\chi(\vec{v})\, \delta\Big(E + \frac{q^2}{2m_\chi} - \vec{q} \cdot \vec{v}\Big) \frac{F^2_{\text{DM}}(\vec{q}, \vec{v})}{4\pi \mu_\chi^2 m_\chi} \mathcal{R} \cdot f_S^2(\vec{q}, E). \tag{1.22}$$

If the spins of the DM and SM particles are unpolarized, then F^2_{DM} is rotationally invariant, even when including the most generic set of effective operators in the nonrelativistic theory:

$$F^2_{\text{DM}}(\vec{q}, \vec{v}) = F^2_{\text{DM}}(q, v, \vec{q} \cdot \vec{v}), \tag{1.23}$$

with $q = |\vec{q}|$ and $v = |\vec{v}|$. Noting the energy-conserving δ function, instances of $\vec{q} \cdot \vec{v}$ can be replaced with $E + q^2/(2m_\chi)$.

For spin-independent DM–SM interactions (and also the simple $\mathcal{O}_4$ spin coupling of [31, 32]), the form factor F^2_{DM} is a velocity-independent function of q and the scalar or vector mediator mass m_ϕ. At tree level:

$$F^2_{\text{DM}}(q) = \left(\frac{q^2_{\text{ref}} + m_\phi^2}{q^2 + m_\phi^2}\right)^2. \tag{1.24}$$

Experimental results and projected sensitivities are often shown using the "heavy mediator" ($m_\phi \gg q$) and "light mediator" ($m_\phi \ll q$) limits of this expression, in which $F^2_{\text{DM}}(q) \simeq 1$ and $F^2_{\text{DM}}(q) \simeq (q_{\text{ref}}/q)^4$, respectively.

In the limits where the mediator mass is either very light or very heavy compared to q, the spin–momentum and spin–velocity couplings of [31, 32] can all be converted into the form:

$$F^2_{\text{DM}}(q, v) = \sum_{a,b} d_{a,b}(E) \times \left(\frac{q}{q_{\text{ref}}}\right)^a \left(\frac{v}{c}\right)^b, \tag{1.25}$$

for some coefficients $d_{a,b}$, Ref. [2] derives a number of useful analytic results for the rate calculation. The current versions of VSDM include support for arbitrary values of a and b, including non-integer values. Full support for form factors at finite mediator mass m_ϕ,

$$F^2_{\text{DM}} \propto \left(\frac{1}{q^2 + m_\phi^2 c^2}\right)^2 q^a v^b, \tag{1.26}$$

may also be added in future versions of VSDM.

1.4 Detector Response Form Factors

The material response function describes SM physics: given some momentum and energy transfer $\vec{q}$ and E, what is the probability that the detector material will undergo a transition from the ground state into a detectable final state? In (1.5), the total material response is $N_T f_S^2(\vec{q}, E)$, where f_S^2 is the response for a single unit of SM material (e.g. one molecule, or one atom, or the unit cell of a crystal) and N_T is the total number of these units. As a function of the detector target mass M_T and the molar mass $m_T^{(\mathrm{mol})}$ of the target atom/molecule/crystal/etc.,

$$N_T = N_A \times \frac{M_T}{m_T^{(\mathrm{mol})}}, \tag{1.27}$$

where $N_A \simeq 6.022 \cdot 10^{23}$ is the Avogadro constant.

In the simplest case, where the SM detector target is modeled as a collection of free particles (e.g. atomic nuclei [34–39]), $f_S^2(\vec{q}, E)$ is simply an energy-conserving δ function:

$$f_S^2(\vec{q}, E) = \delta\left(\frac{q^2}{2m_A} + \Delta E - E \right) \tag{1.28}$$

Here E is the total energy transferred from the DM to the SM system, m_A is the mass of the SM particle, and ΔE is the kinetic energy converted into mass or potential energy (e.g. scattering an atomic nucleus into an excited nuclear state) in the case of inelastic scattering. The recoil energy of the SM particle is often written as $E_R \equiv q^2/(2m_A)$. The continuum of final states is labeled according to E_R, while ΔE may take discrete values (or it may simply be zero). For DM–nuclear scattering, this expression is modified to accommodate a nuclear form factor, $F^2(q)$:

$$f_S^2(\vec{q}, E) = F^2(q) \times \delta(E_R + \Delta E - E), \tag{1.29}$$

where following the notation of [30], $F^2(q)$ accounts for the finite size of the nucleus, r_n, in the limit where $q \gtrsim r_n^{-1}$.

For DM–electron or DM–phonon scattering, $f_S^2(\vec{q}, E)$ depends on the wavefunctions of the SM initial and final states. In terms of the dynamic structure factor $S(\vec{q}, \omega)$ of [40], normalized using the volume of the unit cell of the crystal V_{cell}:

$$f_S^2(\vec{q}, \omega) = \frac{V_{\mathrm{cell}}}{2\pi} S(\vec{q}, \omega). \tag{1.30}$$

In certain DM models, $S(\vec{q}, \omega)$ can be recast in terms of the material's dielectric function $\epsilon(\vec{q}, \omega)$ [13, 41, 42]:

$$f_S^2(\vec{q}, \omega) = \frac{V_{\mathrm{cell}} q^2}{4\pi^2} \frac{1}{\alpha} \mathrm{Im}\left(\frac{-1}{\epsilon(\vec{q}, \omega)} \right). \tag{1.31}$$

This result applies if the DM couples to the electron number density, and not e.g. the spins of the SM particles. If the DM–SM interaction is mediated by a dark photon kinetically mixed with the SM photon, this formalism in terms of the energy loss function (ELF) is exact [41], and so this is sometimes referred to as the dark photon model of DM–SM interactions.

For atomic and molecular systems with discrete spectra of final states s, with energies ΔE_s,

$$f_S^2(\vec{q}, E) = \sum_s f_{g \to s}^2(\vec{q}) \, \delta(E - \Delta E_s), \tag{1.32}$$

with a different momentum form factor $f^2_{g \to s}(\vec{q})$ for each excited state. In the case of single particle scattering from the ground state g to an excited final state s, $f^2_{g \to s}(\vec{q})$ is given by

$$f^2_{g \to s}(\vec{q}) = \left| \int d^3 r \, \Psi^\star_s(\vec{r}) e^{i\vec{q}\cdot\vec{r}} \Psi_g(\vec{r}) \right|^2 = \left| \int \frac{d^3 k}{(2\pi)^3} \psi^\star_s(\vec{k}+\vec{q}) \psi_g(\vec{k}) \right|^2, \tag{1.33}$$

where $\Psi_{g,s}(\vec{r})$ are position space wavefunctions (with normalization $\int d^3 r |\Psi(\vec{r})|^2 = 1$), and $\psi_{g,s}(\vec{k})$ are the momentum space wavefunctions (normalized $\int d^3 k |\psi(\vec{k})|^2 = (2\pi)^3$). Refs. [10, 23] describe how to calculate the momentum form factor $f^2_{g \to s}(\vec{q})$ using this type of expression when the states $\Psi_{g,s}$ are multi-electron wavefunctions.

The standard versions of the material form factors $f^2_S(\vec{q}, E)$ described above are sufficient for describing spin-independent DM–SM scattering, e.g. with the interaction mediated by a new scalar particle of mass m_ϕ. These f^2_S functions are also appropriate for some, but not all, of the spin-dependent interactions of Refs. [31, 32], e.g. the DM–SM spin coupling operator, $\mathcal{O}_4 = \vec{S}_{\mathrm{SM}} \cdot \vec{S}_{\mathrm{DM}}$ in a model of fermionic DM with a vector mediator. To include the other possible DM–SM interactions, however, one must calculate additional material response functions. In the case of Ref. [31], the full set of operators $\mathcal{O}_i$ couple to a total of six different types of nuclear response functions $F^2(q)$. For DM–electron scattering, Ref. [43] introduces a vectorial form factor $\vec{f}_{g \to s}(\vec{q})$,

$$\vec{f}_{g \to s}(\vec{q}) = \int d^3 r \, \Psi^\star_s(\vec{r}) e^{i\vec{q}\cdot\vec{r}} \frac{i\nabla_{\vec{r}}}{m_e} \Psi_g(\vec{r}) = \int \frac{d^3 k}{(2\pi)^3} \psi^\star_s(\vec{k}+\vec{q}) \frac{\vec{k}}{m_e} \psi_g(\vec{k}), \tag{1.34}$$

analogous to (1.33). Ref. [32] enumerates four relevant combinations of $f_{g \to s}$ and $\vec{f}_{g \to s}$ to form the material response functions $\mathcal{B}_i(\vec{q})$, e.g. $\mathcal{B}_1(\vec{q}) = |f_{g \to s}(\vec{q})|^2$ and $\mathcal{B}_3(\vec{q}) = (\vec{f}_{g \to s})^\star \cdot \vec{f}_{g \to s}$. Each $\mathcal{B}_i$ couples to a different "DM response function" $\mathcal{R}_i(q, v)$, which in our notation is proportional to $F^2_{\mathrm{DM}}(q, v)$. In terms of the $\mathcal{W}_i(\vec{q}, E)$ material response functions of [32],

$$\bar{\sigma}_0 F^2_{\mathrm{DM}}(q, v) f^2_S(\vec{q}, E) = \frac{\mu^2_\chi}{16 m^2_\chi m^2_e} \frac{1}{E} \sum_i \mathrm{Re}\left[\mathcal{R}^\star_i(q, v) \mathcal{W}_i(\vec{q}, E) \right]. \tag{1.35}$$

When using VSDM with multiple response functions $\mathcal{W}_i$ and operators $\mathcal{O}_j$, the product $F^2_{\mathrm{DM}} f^2_S$ should be expanded into at least one term per material response function: that is,

$$\frac{1}{E} \mathrm{Re}\left[\mathcal{R}^\star_i(q, v) \mathcal{W}_i(\vec{q}, E) \right] \equiv \frac{16 m^2_\chi m^2_e}{\mu^2_\chi} \bar{\sigma}^{(i)}_0 F^{2(i)}_{\mathrm{DM}}(q, v) f^2_{S(i)}(\vec{q}, E), \tag{1.36}$$

$$\bar{\sigma}_0 F^2_{\mathrm{DM}}(q, v) f^2_S(\vec{q}, E) = \sum_i \bar{\sigma}^{(i)}_0 F^{2(i)}_{\mathrm{DM}}(q, v) f^2_{S(i)}(\vec{q}, E). \tag{1.37}$$

Monomial Format in VSDM: The VSDM calculation is most efficient if $F^{2(i)}_{\mathrm{DM}}$ is expanded in monomials of q and v as in (1.38),

$$F^{2(i)}_{\mathrm{DM}}(q, v) = \sum_{a,b} d^{(i)}_{a,b}(E) \times \left(\frac{q}{q_{\mathrm{ref}}} \right)^a \left(\frac{v}{c} \right)^b. \tag{1.38}$$

After VSDM calculates the component of the rate generated by each (i, a, b) combination, these contributions can be added together:

$$\frac{dR}{dE}\bigg|_{E,\mathcal{R}} = \frac{N_T \rho_\chi}{4\pi \mu^2_\chi m_\chi} \sum_{i,a,b} \bar{\sigma}^{(i)}_0(E) \int d^3 q \, d^3 v \, g_\chi(\vec{v}) \delta\left(E + \frac{q^2}{2m_\chi} - \vec{q}\cdot\vec{v} \right) \frac{q^a v^b}{q^a_{\mathrm{ref}} c^b} \mathcal{R} \cdot f^2_{S(i)}(\vec{q}, E), \tag{1.39}$$

by absorbing the dimensionless coefficients $d_{ab}^{(i)}$ into the value of $\bar{\sigma}_0^{(i)}$ used in the exposure factor:

$$\bar{\sigma}_0^{(i)}(E) \equiv \bar{\sigma}_0^{(i)} d_{ab}^{(i)}(E). \tag{1.40}$$

1.5 The DM Velocity Distribution

The final ingredient in the rate calculation is the local DM velocity distribution. Much of the literature uses the notation $f_\chi(\vec{v})$ for this quantity. We feel, however, that there are already sufficiently many physical quantities represented by the letter f or F, so following [1,2] we use the notation $g_\chi(\vec{v})$.

VSDM will accommodate any 3d velocity distribution, as long as the DM speeds are comfortably non-relativistic. There are scenarios in which a subcomponent of the local dark matter could be boosted to relativistic speeds: cosmic ray upscattering [44,45], for example, or models in which the DM interacts with or is produced by a hot SM source (e.g. the Sun [46,47]). These relativistic effects are not currently included in VSDM.

The Earth Velocity and `earthspeed` Package: Any static DM velocity distribution in the galactic rest frame becomes time-dependent when observed in the lab frame, assuming the lab is orbiting the Sun. Following [1], we have organized the rate equation (1.5) so that this time-dependence is accounted for in the lab frame velocity distribution $g_\chi(\vec{v}, \vec{v}_E)$, where $\vec{v}_E(t)$ is the lab velocity relative to the galactic center. In terms of the Earth velocity with respect to the Sun, $v_\oplus(t)$, $\vec{v}_E(t)$ varies over the course of the year according to:

$$\vec{v}_E(t) \simeq \vec{v}_{\mathrm{LSR}} + \vec{v}_\odot + \vec{v}_\oplus(t), \tag{1.41}$$

where $\vec{v}_\odot$ is the peculiar velocity of the Sun relative to the LSR. On average, $\langle |\vec{v}_\oplus| \rangle \simeq 29.8\,\mathrm{km/s}$. Taking into account the angle between $\vec{v}_\odot$ and the plane swept out by $\vec{v}_\oplus(t)$, the annual variation on $|\vec{v}_E(t)|$ is approximately $\pm 15\,\mathrm{km/s}$. Refs. [30,48] provide simplified expressions for $\vec{v}_E(t)$, and Refs. [49,50] correctly account for the ellipticity of the Earth's orbit. Using an LSR speed of 238 km/s and the updated parameters for $\vec{v}_\odot$ and $\vec{v}_\oplus$ from [48,50], in 2024 the Earth speed reached a maximum of $v_E \simeq 266.2\,\mathrm{km/s}$ on May 31, 11:59 UTC, and a minimum of $v_E \simeq 237.2\,\mathrm{km/s}$ at Dec. 2, 4:07 UTC.

A public Python package, `earthspeed` [51], implements the precise version of $\vec{v}_\oplus(t)$ using the updated values of v_{LSR} and $\vec{v}_\oplus$ from [48]. For future reference, Figure 2 shows how the Earth speed $|\vec{v}_E(t)|$ and direction of motion $\hat{v}_E(t)$ change from month to month. Several studies of anisotropic detectors and daily modulation (e.g. [7]) define a $\hat{z}$ direction in the lab based on the DM wind velocity $(-\vec{v}_E)$ at some particular time of day. For convenience to the experimentalist, Figure 2 converts $\hat{v}_E(t)$ into the celestial coordinate system of right ascension (RA) and declination (dec). Identifying the $\hat{z} \propto \hat{v}_E(t=0)$ direction is then as simple as aligning a telescope to point at a particular star (from the basement of a building, or an underground cavern).

The Standard Halo Model: The DM velocity distribution in our neighborhood of the Milky Way is essentially unknown. For the purposes of comparing different DM experiments, however, it is convenient to approximate $g_\chi(\vec{v})$ using a simplified model. In the Standard Halo Model (SHM), $g_\chi(\vec{v})$ is a Maxwell–Boltzmann distribution, with a cutoff at the galactic escape velocity v_{esc}. In the galactic rest frame the SHM is isotropic, but in the lab frame it is

$$g_{\mathrm{SHM}}(\vec{v}, t) = \frac{1}{N_0} \exp\left(-\frac{|\vec{v} + \vec{v}_E(t)|^2}{v_\sigma^2} \right) \Theta(v_{\mathrm{esc}} - |\vec{v} + \vec{v}_E(t)|), \tag{1.42}$$
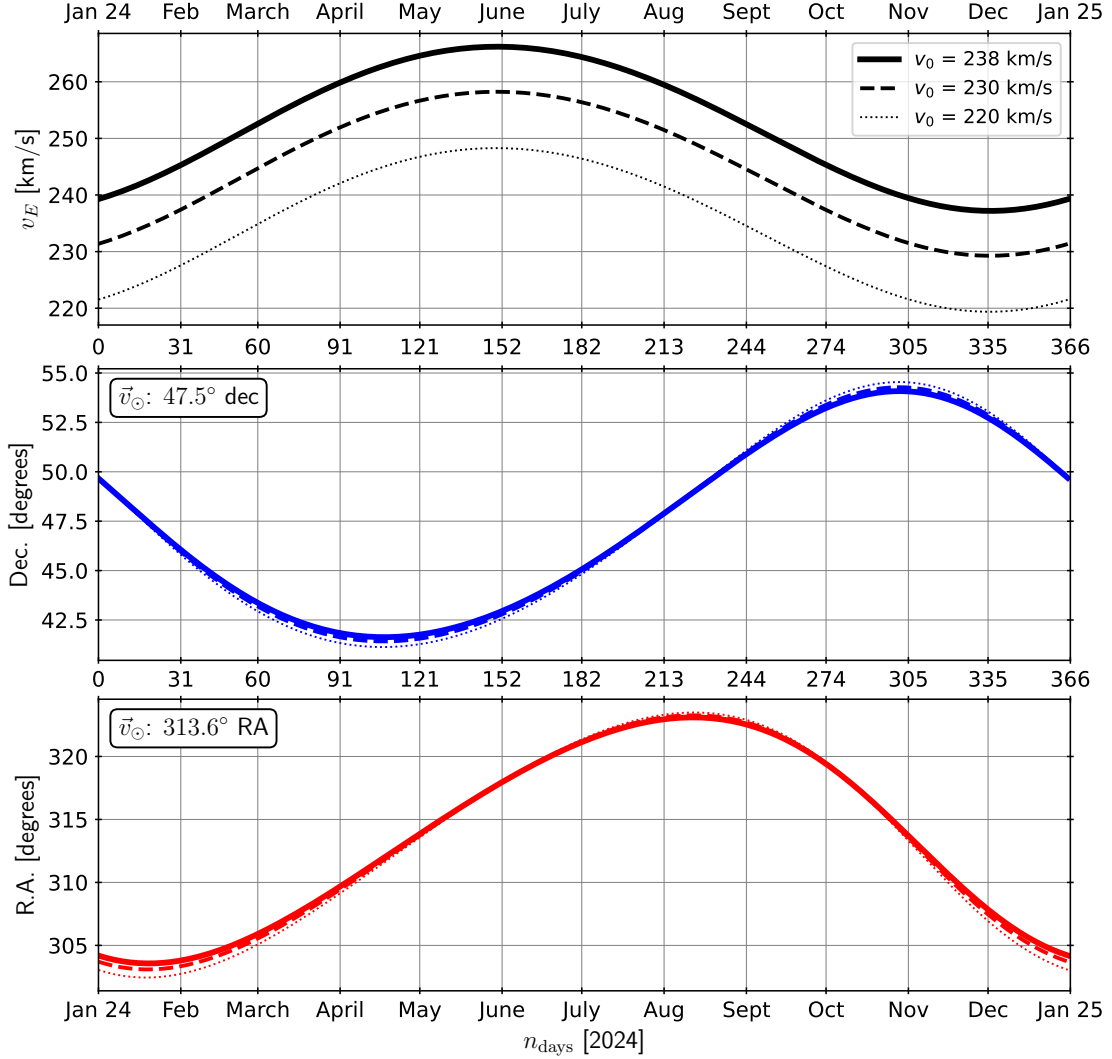
Figure 2: The Earth velocity $\vec{v}_E(t)$, in the spherical coordinate system of declination (dec) and right ascension (RA) familiar to astronomers, for three values of the LSR speed: $v_0 = (220, 230, 238)\,\mathrm{km/s}$. Time of year t is measured in the number of days since UTC midnight on January 1, 2024. The angle between the north pole and $\vec{v}_E$ is $\theta_N \equiv 90° - \mathrm{dec}$. Right ascension specifies the longitudinal position of $\vec{v}_E$ on the celestial sphere: this determines what time of day the DM wind reaches its highest and lowest points on the sky, for a lab location on or near the surface of the Earth.

where $\vec{v}_E(t)$ is the lab velocity (on Earth) relative to the galactic rest frame, and $N_0(v_\sigma, v_{esc})$ is a normalization factor. The Heaviside Θ function imposes a discontinuously sharp cutoff at the escape velocity. The velocity dispersion, v_σ, is taken to match the local circular speed of the galaxy, i.e. the local standard of rest (LSR). Many older references use $v_\sigma = 220$ km/s or $v_\sigma = 230$ km/s [30], but more recent measurements suggest $v_\sigma \approx 238$ km/s [48]. For the escape speed, Ref. [48] recommends $v_{esc} = 544$ km/s.

While the SHM is a useful standard for predicting scattering rates in direct detection experiments, it is not supposed to be a particularly accurate representation of the as-yet unknown velocity distribution in our galaxy. Data from the Milky Way (especially the SDSS and Gaia sky surveys) indicate that the local stellar velocity distribution contains unvirialized substructures [52, 53] from relatively recent galactic merger events, which may well be accompanied by matching kinematic substructures in the DM velocity distribution [54–56]. The SHM also neglects the gravitational effects from the Sun and Earth, which accelerate the incoming DM particles according to

$$v_f^2 - v_i^2 \simeq 2\frac{GM_\odot}{1\,\text{AU}} + 2\frac{GM_\oplus}{R_\oplus} \approx 2.113 \cdot 10^{-8} c^2, \qquad v_f \simeq \sqrt{v_i^2 + (43.58\,\text{km/s})^2}, \qquad (1.43)$$

with the Sun doing about 93% of the work. This skews the lab-frame velocity distribution towards speeds. For $v_i = 250$ km/s the effect is modest, increasing v_i by 3.7 km/s. For the slower part of the distribution the effect is more pronounced: an incoming $v_i = 100$ km/s would be boosted by 9.1 km/s, for example, while the entire $0 \le v_i \lesssim 44$ km/s part of the distribution shifts to 44 km/s $\lesssim v_f \lesssim 62$ km/s. Likewise, the gravitational potential of the Sun is known to perturb the local DM density, inducing an annual variation that impacts the predictions for annual modulation searches [57].

Additionally, the surface speed of the Earth slightly shifts the lab frame velocity distribution by an amount that decreases with latitude, from 0.465 km/s at the equator to zero at the poles. Compared to the effects from annual modulation and the gravitational potentials of the Sun and Earth, the surface speed is a tiny perturbation, altering $\vec{v}_E(t)$ by no more than 0.2%. However, the surface velocity is periodic over the sidereal day (23.93 hours), which does distinguish it from the annual effects. In this sense the Earth surface speed provides a minimum degree of daily modulation in materials that are highly isotropic. Anisotropic materials can have daily modulation amplitudes that are two orders of magnitude larger than this (e.g. [10, 12]).

The Integrated Velocity Distribution: It is common in the literature to define an integrated version of the velocity distribution, usually notated by η, sometimes referred to as the average inverse speed. The usual derivations assume either that the detector response is isotropic, or that there is some velocity frame in which the DM velocity distribution is isotropic. In these cases, some or all of the angular integrals in (1.5) can be completed analytically. Refs. [2, 40] provide a more general version of this quantity, that does not rely on either assumption. For spin-independent scattering, the quantity

$$\eta(\vec{q}, E) \equiv 2q \int d^3v\, g_\chi(\vec{v})\, \delta\left(E + \frac{q^2}{2m_\chi} - \vec{q} \cdot \vec{v}\right) \qquad (1.44)$$

is sufficient for calculating the scattering rate; but for velocity-dependent $F_{DM}^2(q, v)$ form factors $F_{DM}^2 \propto v^b$, one must also evaluate

$$\eta^{(b)}(\vec{q}, E) \equiv 2q \int d^3v\, \frac{v^b}{c^b}\, g_\chi(\vec{v})\, \delta\left(E + \frac{q^2}{2m_\chi} - \vec{q} \cdot \vec{v}\right). \qquad (1.45)$$

The factor of $2q$ here ensures that the angular average of our $\eta(\vec{q}, E)$ matches the isotropic version, $\eta(q, E)$, that appears elsewhere in the literature. For comparison with the related quantity $g(\vec{q}, \omega)$ of [40], note that $\eta(\vec{q}, E) = (q/\pi)g(\vec{q}, E)$. In the special case where $g_\chi(\vec{v})$ is isotropic in some other velocity frame, Ref. [2] shows how $\eta(\vec{q}, E)$ can be reduced to a 1d integral over speeds. For the SHM, even this integral can be completed analytically: the result is

$$\eta(\vec{q}, E) = \frac{2\pi v_\sigma^2}{N_0} \left(e^{-v_-(\vec{q}, E)^2/v_\sigma^2} - e^{-v_{\text{esc}}^2/v_\sigma^2} \right) \Theta(v_{\text{esc}} - v_-(\vec{q}, E)), \tag{1.46}$$

where

$$v_-(\vec{q}, E) \equiv \frac{E}{q} + \frac{q}{2m_\chi} + \frac{\vec{q} \cdot \vec{v}_E}{q}. \tag{1.47}$$

If a closed form analytic expression can be obtained for $\eta(\vec{q}, E)$, then this quantity can be quite helpful for calculating the scattering rate. Expanding $F_{\text{DM}}(q, v)$ in powers of v,

$$F_{\text{DM}}^2(q, v) = \sum_b d_b(E) \frac{v^b}{c^b} F_{\text{DM}}^{2(b)}(q), \tag{1.48}$$

the differential scattering rate becomes

$$\left. \frac{dR}{dE} \right|_{E, \mathcal{R}} = \sum_b d_b(E) \frac{N_T \rho_\chi \bar{\sigma}_0}{8\pi \mu_\chi^2 m_\chi} \int \frac{d^3q}{q} \eta^{(b)}(\vec{q}, E) F_{\text{DM}}^{2(b)}(q) \mathcal{R} \cdot f_S^2(\vec{q}, E). \tag{1.49}$$

If $\eta(\vec{q}, E)$ is obtained via numerical integration, however, this version of the rate calculation may become even more difficult than directly evaluating (1.5). Tabulating $\eta(\vec{q}, E)$ a single time would be bad enough, but this calculation needs to be repeated for every value of the DM mass and the Earth velocity. In such a case it is almost certainly better to use the VSDM method outlined in Section 2 instead.

1.6 What VSDM Does and Does Not Calculate

There are some physical effects that VSDM does not include, but which are straightforward for the user to implement themselves. Spatial or temporal variations in the local DM density ρ_χ are one such example: e.g. if gravitational focusing effects from the Sun cause $\rho_\chi(t)$ to vary over the course of the year. As ρ_χ simply rescales the overall event rate, this type of time dependence can be accommodated by multiplying the output of VSDM by a factor of $\rho_\chi(t)/\rho_\chi(t=0)$.

Similarly, there may be some efficiency factors that suppress the actual number of events measured by a detector. In this case, the predicted average rate of events with E in some range $E_1 \leq E \leq E_2$ would be

$$R(E_1 \leq E \leq E_2, \mathcal{R}) = \int_{E_1}^{E_2} dE \, \xi(E) \left. \frac{dR}{dE} \right|_{E, \mathcal{R}}, \tag{1.50}$$

where $\xi(E)$ is some efficiency factor (as a function of energy deposition E). For this reason and others, VSDM generally only calculates the differential rate dR/dE: it is left to the user to perform any integrals over energy. The exception to this rule is for discrete final states, where

$$f_S^2(\vec{q}, E) = f_{g \to s}^2(\vec{q}) \delta(E - \Delta E_s). \tag{1.51}$$

Given $f_{g \to s}^2(\vec{q})$, VSDM calculates the rate $R(\mathcal{R})$ directly using (1.7).

VSDM also does not calculate $f_S^2(\vec{q}, E)$ or $g_\chi(\vec{v})$ from first principles: these inputs should generally be provided by the user. Due to the popularity (and simplicity) of the SHM, however, the Python repository `vsdm` does include some SHM-specific tools for the user's convenience. For analyses with continuous energy spectra, calculating $dR/dE|_E$ using (1.5), the user should provide the form factor $f_S^2(\vec{q}, E)$ evaluated at the same set of $E = \{E_j\}$ values as dR/dE. One of the intermediate steps in the VSDM calculation involves calculating angular integrals of the form

$$\int d\Omega_q \, Y_{\ell m}(\hat{q}) f_S^2(\vec{q}, E), \qquad (1.52)$$

for real spherical harmonics $Y_{\ell m}$. If the user has previously tabulated $f_S^2(\vec{q}, E)$ on a grid of $(\vec{q}_i, E_j)$ values, then f_S^2 can be provided to VSDM as an interpolating function. Alternatively, if $f_S^2(\vec{q}, E)$ is tabulated on a spherical grid in $\vec{q} = (|q|, \theta, \phi)$, then the necessary angular integrals can be evaluated in large batches using a discrete spherical harmonic transformation instead. We highly recommend this approach when working with tabulated $f_S^2(\vec{q}, E)$: compared to numerical integration, the discrete harmonic transformation is extremely fast.

2 Vector Space Rate Calculation

VSDM implements the vector space integration method introduced in [1] and described in detail in [2]. In this Section we provide the key results and intermediate steps, describing how VSDM treats the units and dimensionful quantities at each stage. Compared to Ref. [2], the discussion in this section works backwards: we begin by introducing the final result for the rate in terms of the partial rate matrix $K_{mm'}^{(\ell)}$, then later in the section we go through the intermediate calculations that are designed to evaluate $K^{(\ell)}$ efficiently.

VSDM makes the following assumptions:

- the lab-frame $g_\chi(\vec{v})$ is entirely nonrelativistic, vanishing above some cutoff $v_{\max}$:

$$g_\chi(|\vec{v}| > v_{\max}) = 0. \qquad (2.1)$$

- the cross section $\sigma(\vec{q}, \vec{v})$ is a scalar function of q, v, and $\vec{q} \cdot \vec{v}$:

$$F_{\mathrm{DM}}^2(\vec{q}, \vec{v}) = F_{\mathrm{DM}}^2(q, v, E). \qquad (2.2)$$

The Partial Rate Matrix: Under these assumptions, Ref. [1] shows that the differential scattering rate as a function of detector orientation $\mathcal{R}$ is entirely described by the "partial rate matrix" $K_{mm'}^{(\ell)}$, actually a series of matrices that transform as representations of $SO(3)$. Like the scattering rate itself, the partial rate matrix is a function of the DM model $(m_\chi, F_{\mathrm{DM}}^2)$ and the physical inputs $g_\chi(\vec{v})$ and $f_S^2(\vec{q}, E)$. Its differential form is defined [1]:

$$\left. \frac{dK_{mm'}^{(\ell)}}{dE} \right|_E = \int \frac{d^3q \, d^3v}{qv} P_\ell\left(\frac{v_{\min}(q, E)}{v}\right) Y_{\ell m}(\hat{v}) g_\chi(\vec{v}) \frac{F_{\mathrm{DM}}^2(q, v)}{2\mu_\chi^2 m_\chi} Y_{\ell m'}(\hat{q}) f_S^2(\vec{q}, E), \qquad (2.3)$$

where $P_\ell(x)$ is the ℓth Legendre polynomial, $Y_{\ell m}(\theta, \phi)$ are the real spherical harmonics (see Sec. 2.1), and $v_{\min}$ is the function defined in (1.10). The velocity integral is restricted to the region where $v_{\min} \leq v$, and the argument of P_ℓ is less than or equal to 1.

Although the value of dK/dE is a function of the initial orientation of $f_S^2(\vec{q}, E)$, it is not a function of $\mathcal{R}$. Instead, the scattering rate $R(\mathcal{R})$ is found by multiplying $K_{mm'}^{(\ell)}$ by the "Wigner

G matrix" of [2]:

$$\frac{dR}{dE}\bigg|_{E,\mathcal{R}} = N_T \rho_\chi \bar{\sigma}_0 \sum_{\ell=0}^{\infty} \sum_{m=-\ell}^{\ell} \sum_{m'=-\ell}^{\ell} G_{mm'}^{(\ell)}(\mathcal{R}) \cdot \frac{dK_{mm'}^{(\ell)}}{dE}\bigg|_E, \tag{2.4}$$

where $G_{mm'}^{(\ell)}(\mathcal{R})$ is defined as

$$G_{mm'}^{(\ell)}(\mathcal{R}) \equiv \int d\Omega_u \, Y_{\ell m}(\hat{u}) \, \mathcal{R} \cdot Y_{\ell m'}(\hat{u}) = \int d\Omega_u \, Y_{\ell m}(\hat{u}) \, Y_{\ell m'}(\mathcal{R}^{-1}\hat{u}). \tag{2.5}$$

Although (2.5) is a clear way to define G, this integral is not the most convenient way to calculate $G(\mathcal{R})$. Instead, after calculating the standard Wigner $D^{(\ell)}(\mathcal{R})$ matrix using the package spherical (SphericalFunctions.jl), VSDM uses the relationship between $G^{(\ell)}$ and $D^{(\ell)}$ from [2] to determine $G^{(\ell)}(\mathcal{R})$.

VSDM works by truncating the expansion in ℓ at some finite $\ell_{\max}$:

$$\frac{dR}{dE}\bigg|_{E,\mathcal{R}} \simeq \sum_{\ell=0}^{\ell_{\max}} \frac{dR_{(\ell)}}{dE}\bigg|_{E,\mathcal{R}}, \tag{2.6}$$

$$\frac{dR_{(\ell)}}{dE}\bigg|_{E,\mathcal{R}} = N_T \rho_\chi \bar{\sigma}_0 \sum_{m,m'=-\ell}^{\ell} G_{mm'}^{(\ell)}(\mathcal{R}) \cdot \frac{dK_{mm'}^{(\ell)}}{dE}\bigg|_E. \tag{2.7}$$

For each value of ℓ, the sum over m, m' is equivalent to taking the dot product of two vectors of length $(2\ell+1)^2$. Including all $\ell \leq \ell_{\max}$, the total number of terms in (2.4) grows as $4\ell_{\max}^3/3$ in the limit of large $\ell_{\max}$.

Given a particular precision goal, the appropriate value for $\ell_{\max}$ can be estimated *a priori* by inspecting the sizes of $g_{\ell m}(v)$ and $f_{\ell m'}^2(q, E)$ input functions. To verify that $\ell_{\max}$ is large enough, however, the user should inspect the relative magnitudes of $dR_{(\ell)}/dE$ as a function of ℓ at the end of the calculation, to verify that this series does converge to within the desired precision.

For discrete final states, $f_S^2(\vec{q}, E) = f_{g \to s}^2(\vec{q})\delta(E - \Delta E)$, the differential form of K is replaced by:

$$K_{mm'}^{(\ell)}(\Delta E) = \int \frac{d^3q \, d^3v}{qv} P_\ell\left(\frac{v_{\min}(q, \Delta E)}{v}\right) Y_{\ell m}(\hat{v}) g_\chi(\vec{v}) \frac{F_{\mathrm{DM}}^2(q, v)}{2\mu_\chi^2 m_\chi} Y_{\ell m'}(\hat{q}) f_{g \to s}^2(\vec{q}), , \tag{2.8}$$

$$R(\mathcal{R}, \Delta E) = N_T \rho_\chi \bar{\sigma}_0 \sum_{\ell=0}^{\infty} \sum_{m=-\ell}^{\ell} \sum_{m'=-\ell}^{\ell} G_{mm'}^{(\ell)}(\mathcal{R}) \cdot K_{mm'}^{(\ell)}(\Delta E). \tag{2.9}$$

Result: The dot product version of (2.4) needs to be evaluated for each of the $N_{\mathrm{DM}} N_{\mathcal{R}} N_{g_\chi} N_{f_S}$ combinations of model parameters and physical inputs described in (1.1). With typical values of $\ell_{\max} \sim 10^1$, each rate calculation might take some number of microseconds to evaluate. Using an earlier version of the Python implementation vsdm, for example, Ref. [2] found that the evaluation time in seconds was approximately $T_{\mathrm{eval}} \sim \ell_{\max}^3 \times 10^{-9}$ s. Given $dK_{mm'}^{(\ell)}/dE$ for each combination of models, a typical personal computer using vsdm could calculate the DM scattering rate for billions of combinations of $\mathcal{R}$, g_χ, f_S^2, m_χ and F_{DM} per hour. The Julia version is faster.

Now that the importance of the partial rate matrix has been established, the remainder of this section describes the steps VSDM takes to calculate it. Sections 2.1 and 2.2 begin by defining a basis of orthogonal 3d functions, in terms of real spherical harmonics and the spherical Haar wavelets introduced in Ref. [2]. A user familiar with [2] can skip directly to Section 2.3, which specifies the conventions that we use to keep track of the dimensionful quantities.

2.1 Spherical Harmonics

VSDM uses real spherical harmonics for its rate calculation, because the input functions $g_\chi(\vec{v})$ and $f_S^2(\vec{q}, E)$ are real-valued. As in [2], for polar and azimuthal angles θ, ϕ, $Y_{\ell m}(\theta, \phi)$ is defined explicitly as:

$$
Y_{\ell m}(\theta, \phi) = \begin{cases} \sqrt{2}(-1)^m \sqrt{\dfrac{2\ell+1}{4\pi} \dfrac{(\ell-|m|)!}{(\ell+|m|)!}} P_\ell^{|m|}(\cos\theta)\sin(|m|\phi) & \text{for } m < 0, \\[2em] \sqrt{\dfrac{2\ell+1}{4\pi}} P_\ell(\cos\theta) & \text{for } m = 0, \\[2em] \sqrt{2}(-1)^m \sqrt{\dfrac{2\ell+1}{4\pi} \dfrac{(\ell-m)!}{(\ell+m)!}} P_\ell^m(\cos\theta)\cos(m\phi) & \text{for } m > 0, \end{cases} \tag{2.10}
$$

where $P_\ell^m(x)$ on $-1 \le x \le 1$ are the associated Legendre polynomials,

$$
P_\ell^m(x) \equiv \frac{(-1)^m}{2^\ell \ell!}(1-x^2)^{m/2}\frac{d^{\ell+m}}{dx^{\ell+m}}(x^2-1)^\ell. \tag{2.11}
$$

To avoid numerical instabilities at large ℓ, VSDM uses the efficient iterative method outlined in Appendix A to calculate $\sqrt{(\ell-m)!/(\ell+m)!}P_\ell^m(x)$ directly.

Properties: Under central inversion, $\hat{u} \to -\hat{u}$, i.e. $\theta \to \pi - \theta$ and $\phi \to \phi \pm \pi$, the harmonics transform as

$$
Y_{\ell m}(-\hat{u}) = Y_{\ell m}(\pi - \theta, \phi \pm \pi) = (-1)^\ell Y_{\ell m}(\theta, \phi), \tag{2.12}
$$

while under reflections along the x axis ($\hat{x} \to -\hat{x}$, or $\phi \to -\phi$),

$$
Y_{\ell m}(\theta, -\phi) = (-1)^m Y_{\ell m}(\theta, \phi). \tag{2.13}
$$

Under rotations, a spherical harmonic $Y_{\ell m}$ mixes with the other $Y_{\ell m'}$ harmonics with matching ℓ:

$$
\mathcal{R} \cdot Y_{\ell m}(\hat{u}) = \sum_{m'} G_{m'm}^{(\ell)}(\mathcal{R}) Y_{\ell m'}(\hat{u}). \tag{2.14}
$$

For the trivial rotation $\mathcal{R} = \mathbb{1}$, the identity operator, $G^{(\ell)}$ is just the Kronecker δ function:

$$
G_{mm'}^{(\ell)}(\mathbb{1}) = \delta_{mm'}. \tag{2.15}
$$

For the relationship between $Y_{\ell m}$ and the complex spherical harmonics Y_ℓ^m, we refer the reader to Ref. [2]. Likewise, this reference also provides the explicit expression for $G_{mm'}^{(\ell)}(\mathcal{R})$ as a function of $D_{mm'}^{(\ell)}(\mathcal{R})$.

The spherical harmonics provide an orthonormal basis for real-valued functions on the sphere S^2: their inner product is given by

$$
\int d\Omega\, Y_{\ell m}(\theta, \phi)Y_{\ell', m'}(\theta, \phi) = \delta_\ell^{\ell'}\delta_m^{m'}. \tag{2.16}
$$

A function $f(\vec{u})$ of a 3d variable $\vec{u}$ can be written as a series expansion in spherical harmonic modes:

$$
f(\vec{u}) = \sum_{\ell=0}^{\infty}\sum_{m=-\ell}^{\ell} f_{\ell m}(u)Y_{\ell m}(\hat{u}), \tag{2.17}
$$

defining the (ℓ, m) radial function $f_{\ell m}(u)$ as

$$
f_{\ell m}(u) \equiv \int d\Omega_u\, Y_{\ell m}(\hat{u})f(\vec{u}). \tag{2.18}
$$

Revisiting the Partial Rate Matrix: With this notation, the partial rate matrix (2.3) simplifies to a series of 2d integrals on q and v:

$$\frac{dK_{mm'}^{(\ell)}}{dE}\bigg|_E = \int_0^\infty q\,dq \int_{v_{\min}(q,E)}^{v_{\max}} v\,dv\, P_\ell\left(\frac{v_{\min}(q,E)}{v}\right) g_{\ell m}(v) \cdot \frac{F_{\mathrm{DM}}^2(q,v)}{2\mu_\chi^2 m_\chi} \cdot f_{\ell m'}^2(q,E), \quad (2.19)$$

where

$$g_{\ell m}(v) \equiv \int d\Omega_v\, Y_{\ell m}(\hat{v}) g_\chi(\vec{v}), \qquad f_{\ell m'}^2(q,E) \equiv \int d\Omega_q\, Y_{\ell m'}(\hat{q}) f_S^2(\vec{q},E). \quad (2.20)$$

If the input functions $g_\chi(\vec{v})$ and $f_S^2(\vec{q},E)$ allow (2.19) to be evaluated analytically, then our work is done: dR/dE can be found directly from (2.4). If the number of g_χ and f_S^2 models is small, then it might be feasible to integrate (2.19) numerically. That said, following (1.1), there are

$$N_{\mathrm{DM}} N_{g_\chi} N_{f_S} \times \left(\frac{4}{3}\ell_{\max}^3 + \mathcal{O}(\ell_{\max}^2)\right) \quad (2.21)$$

many of these 2d integrals to evaluate. To solve this problem, Section 2.2 introduces the spherical wavelets, replacing $g_{\ell m}(v)$ and $f_{\ell m'}^2(q,E)$ with a set of simpler basis functions, for which (2.19) has a closed-form solution.

2.2 Spherical Haar Wavelets

In this section we introduce the last major component of VSDM: a basis of orthogonal functions in $v = |\vec{v}|$ and $q = |\vec{q}|$. As far as the integration method of [1] is concerned, nearly any type of basis function would work. In VSDM, we use the spherical wavelets derived in [2]. Their major advantages are:

- spherical Haar wavelets are piecewise-constant, making it easy to integrate (2.19);

- the wavelet expansion of 1d functions converge predictably and generically;

- given tabulated values of $f(x)$, there are efficient discrete transformations for calculating the coefficients of the wavelet expansion.

Together, these properties of wavelets greatly reduce the number of integrals that need to be calculated.

Definition: The d-dimensional spherical Haar wavelets defined in [2] form a set of orthogonal basis functions on $x \in [0,1]$, satisfying

$$\int_0^1 x^{d-1} dx\, h_n(x) h_{n'}(x) = \delta_{nn'}. \quad (2.22)$$

For all applications in this paper, we specialize to $d = 3$. As the radial index $n = 0, 1, 2, \ldots$ becomes larger, the $h_n(x)$ functions become narrower, with $h_n(x) = 0$ for x outside of an n-dependent region. The $n = 0$ and $n = 1$ wavelets have support over the full region, $0 \le u \le 1$, starting with:

$$h_{n=0}(0 \le x \le 1) = \sqrt{3}. \quad (2.23)$$

All other wavelets ($n \geq 1$) are organized by a "generation" label, $\lambda = 0, 1, 2, \ldots$, and a position label, $\mu = 0, 1, \ldots, 2^\lambda - 1$:

$$n = 2^\lambda + \mu. \tag{2.24}$$

The generation label λ controls the width of the wavelet: with each generation, the width of the wavelet decreases by a factor of 2. Within each generation, the wavelets are non-overlapping, with their relative positions within the interval $[0, 1]$ determined by μ.

Precisely:

$$h_n(x) = \begin{cases} +A_n & x_1(n) \leq x < x_2(n), \\ -B_n & x_2(n) < x \leq x_3(n), \\ 0 & \text{otherwise,} \end{cases} \tag{2.25}$$

where

$$x_1(n) = 2^{-\lambda}\mu, \qquad x_2(n) = 2^{-\lambda}(\mu + \tfrac{1}{2}), \qquad x_3(n) = 2^{-\lambda}(\mu + 1), \tag{2.26}$$

and

$$A_n = \sqrt{\frac{3}{x_3^3 - x_1^3} \frac{x_3^3 - x_2^3}{x_2^3 - x_1^3}} \qquad\qquad B_n = \sqrt{\frac{3}{x_3^3 - x_1^3} \frac{x_2^3 - x_1^3}{x_3^3 - x_2^3}}. \tag{2.27}$$

For the $n = 0$ wavelet, $x_1(0) \equiv 0$ and $x_2(0) \equiv 1$. Although there is no second region $[x_2, x_3]$, we define

$$A_0 \equiv \sqrt{3}, \qquad\qquad B_0 \equiv 0 \tag{2.28}$$

for uniformity.

For functions of velocity or momentum, we take $x = v/v_{\max}$ or $x = q/q_{\max}$, for velocity and momentum cutoffs $v_{\max}$ and $q_{\max}$, respectively.

Setting $q_{\max}$: This is one possible disadvantage of the spherical Haar wavelets: because they are defined on a finite interval, $x \in [0, 1]$, we must impose some cutoff on the momentum q. We have already established that g_χ is only nonzero over a finite range, $0 \leq v \leq v_{\max}$, so for functions of velocity this is no problem; $f_S^2(\vec{q}, E)$, on the other hand, does not necessarily vanish identically at large q.

In the context of direct detection, there is a strict upper limit on which values of q contribute to the scattering rate. Thanks to the lab-frame velocity cutoff $|\vec{v}| \leq v_{\max}$ and the nature of $v_{\min}(q, E)$, the rate integrand (2.19) vanishes for

$$q \geq 2m_\chi v_{\max}, \tag{2.29}$$

because at these q the value of $v_{\min}$ is always larger than $v_{\max}$:

$$v_{\min}(q = 2m_\chi v_{\max}) > v_{\max}. \tag{2.30}$$

So, the momentum integral of (2.19) can be cut off at $q_{\max} = 2m_\chi v_{\max}$ without any approximation.

The form factor $f_S^2(\vec{q}, E)$ may also have a natural cutoff, some $q_{\max}$ at which f_S^2 vanishes or is at least negligibly small. In DM–electron scattering, for example, the Bohr momentum

$$q_{\text{Bohr}} \equiv \alpha m_e c \simeq 3.729 \, \text{keV} \tag{2.31}$$

is the characteristic scale for the momentum transfer, and $f_S^2(\vec{q}, E)$ can become quite small in the $q \gg \alpha m_e$ limit. In [23], for example, $f_{g \to s}^2(\vec{q}) \propto (q_{\text{Bohr}}/q)^7$ at high momenta. For such cases, setting $q_{\text{max}} \sim 10 q_{\text{Bohr}}$ can capture all the relevant dynamics with a negligible loss of precision, even when this q_{max} is orders of magnitude smaller than $2 m_\chi v_{\text{max}}$.

Whatever the physical system, before proceeding with VSDM, the user should identify the physically relevant range of momenta $0 \leq q \leq q_{\text{max}}$ that should be included in the rate integral.

2.3 Basis Functions In VSDM

In VSDM, functions of $\vec{v}$ and $\vec{q}$ are expanded in a series of 3d wavelet-harmonic basis functions,

$$\phi_{n\ell m}(\vec{u}) \equiv h_n\left(\frac{u}{u_{\text{max}}}\right) Y_{\ell m}(\hat{u}). \tag{2.32}$$

At this stage of the calculation it is more convenient to work with the bra-ket notation of Ref. [2], with $g_\chi(\vec{v}) \to \left|g_\chi\right\rangle$, $f_S^2(\vec{q}, E) \to \left|f_S^2\right\rangle$. Basis functions of velocity or momentum are written in terms of their indices, $\phi_{n\ell m}(\vec{u}) \to |n\ell m\rangle$. For real-valued functions, there is no distinction between "bra" and "ket" vectors. In this notation,

$$g_\chi(\vec{v}) = \sum_{n=0}^\infty \sum_{\ell=0}^\infty \sum_{m=-\ell}^\ell \langle n\ell m|g_\chi\rangle \phi_{n\ell m}(\vec{v}), \qquad f_S^2(\vec{q}, E) = \sum_{n\ell m} \langle n\ell m|f_S^2\rangle \phi_{n\ell m}(\vec{q}), \tag{2.33}$$

where the inner products $\langle f|g\rangle$ are defined following the Ref. [2] convention:

$$\langle f|g\rangle \equiv \int \frac{d^3 u}{u_{\text{max}}^3} f(\vec{u}) \cdot g(\vec{u}), \tag{2.34}$$

where $\vec{u}$ in this expression is either $\vec{v}$ or $\vec{q}$. Note that $\langle n\ell m|f_S^2\rangle$ is a function of E, while $\langle n\ell m|g_\chi\rangle$ is a constant. With these conventions, the $|n\ell m\rangle$ are dimensionless and orthonormal,

$$\langle n\ell m|n'\ell'm'\rangle = \delta_{nn'}\delta_{\ell\ell'}\delta_{mm'}, \tag{2.35}$$

and the coefficients $\langle n\ell m|f\rangle$ have the same units as the function $f(\vec{u})$.

Tensor operators $\hat{\mathcal{O}}$ map vectors from one space onto another, e.g. velocity space onto momentum space, or vice versa. In a given basis, the coefficients of a generic $\hat{\mathcal{O}}(\vec{v}, \vec{q})$ are:

$$\langle n\ell m|\hat{\mathcal{O}}|n'\ell'm'\rangle \equiv \int \frac{d^3 v}{v_{\text{max}}^3} \frac{d^3 q}{q_{\text{max}}^3} \phi_{n\ell m}(\vec{v}) \, \hat{\mathcal{O}}(\vec{v}, \vec{q}) \, \phi_{n'\ell'm'}(\vec{q}). \tag{2.36}$$

After shuffling around some factors of v_{max} and q_{max}, the scattering rate R from (1.22) can be written in this form:

$$\left.\frac{dR}{dE}\right|_{E,\mathcal{R}} = \frac{N_T \rho_\chi \bar{\sigma}_0}{4\pi \mu_\chi^2 m_\chi} (q_{\text{max}} v_{\text{max}})^3 \left\langle g_\chi \left| F_{\text{DM}}^2(\vec{q}, \vec{v}) \, \delta\left(E + \frac{q^2}{2m_\chi} - \vec{q} \cdot \vec{v}\right) \right| \mathcal{R} \cdot f_S^2 \right\rangle. \tag{2.37}$$

2.4 The Kinematic Scattering Matrix

Ref. [2] shows that the tensor operator $F_{\text{DM}}^2 \delta(E \ldots - \vec{q} \cdot \vec{v})$ is diagonal in the basis of spherical harmonics: i.e.

$$\left\langle n\ell m \left| \frac{F_{\text{DM}}^2(q, v)}{4\pi \mu_\chi^2 m_\chi} \delta\left(E + \frac{q^2}{2m_\chi} - \vec{q} \cdot \vec{v}\right) \right| n'\ell'm' \right\rangle \equiv \delta_{\ell\ell'}\delta_{mm'} \times \frac{v_{\text{max}}^2}{q_{\text{max}}^4} \mathcal{I}_{nn'}^{(\ell)}(E), \tag{2.38}$$

where $\mathcal{I}^{(\ell)}$ is the *kinematic scattering matrix*. In the wavelet basis, it is:

$$\mathcal{I}_{nn'}^{(\ell)}(E) \equiv \frac{q_{\max}^3/v_{\max}^3}{2m_\chi \mu_\chi^2} \int_0^{q_{\max}} \frac{qdq}{q_{\max}^2} h_{n'}\left(\frac{q}{q_{\max}}\right) \int_{v_{\min}(q,E)}^{v_{\max}} \frac{vdv}{v_{\max}^2} P_\ell\left(\frac{v_{\min}(q,E)}{v}\right) h_n\left(\frac{v}{v_{\max}}\right) F_{\mathrm{DM}}^2(q,v).$$
(2.39)

A factor of $q_{\max}^4/v_{\max}^2$ has been absorbed into $\mathcal{I}$ to make it dimensionless. More importantly, the factors of $m_\chi \mu_\chi^2$ are included as well. This way, $\mathcal{I}^{(\ell)}$ contains all of the terms that depend on the DM particle model, m_χ and F_{DM}^2.[1]

In terms of $\mathcal{I}^{(\ell)}$, the scattering rate (2.37) is:

$$\left.\frac{dR}{dE}\right|_{E,\mathcal{R}} = N_T \rho_\chi \bar{\sigma}_0 \frac{v_{\max}^5}{q_{\max}} \sum_{n,n'} \sum_{\ell m} \langle g_\chi | n\ell m\rangle \mathcal{I}_{nn'}^{(\ell)}(m_\chi, F_{\mathrm{DM}}^2; E) \langle n'\ell m | \mathcal{R} | f_S^2 \rangle.$$
(2.40)

The m_χ dependence of $\mathcal{I}$ enters through its dependence on $v_{\min}(q,E)$.

In VSDM: In the current version of VSDM, if the user wishes to evaluate an F_{DM}^2 that is not of the form $F_{\mathrm{DM}}^2 \propto q^a v^b$, they must provide the solution to (2.39) themselves. Otherwise, for F_{DM}^2 given by (1.38), VSDM provides the contribution to $\mathcal{I}^{(\ell)}$ from each combination of (a,b):

$$F_{\mathrm{DM}}^2(q,v) = \sum_{ab} d_{ab}(E)(q/q_{\mathrm{ref}})^a (v/c)^b,$$
(2.41)

$$\mathcal{I}_{nn'}^{(\ell)}(E) = \sum_{ab} d_{ab}(E) \times \mathcal{I}_{nn'}^{(\ell)}(m_\chi, a, b; E),$$
(2.42)

where $\mathcal{I}_{nn'}^{(\ell)}(a,b;E)$ is defined as in (2.39) but with $F_{\mathrm{DM}}^2 \to (q/q_{\mathrm{ref}})^a(v/c)^b$.

For any pair of velocity and momentum basis functions, there are up to four regions that contribute to the integral, within which $h_n(v)h_{n'}(q)$ is constant. Defining $q_i(n) = x_i(n)q_{\max}$ and $v_i(n) = x_i(n)v_{\max}$, each of the intervals $[v_1, v_3]$ and $[q_1, q_3]$ is subdivided at v_2 and q_2, respectively, and within each region:

$$h_n(v/v_{\max})h_{n'}(q/q_{\max}) = \{A_n A_{n'}, -A_n B_{n'}, -B_n A_{n'}, +B_n B_{n'}\},$$
(2.43)

where A_n, B_n are as defined in (2.27). Referring to these regions as AA, AB, BA, and BB, we define $\mathcal{I}^{(\ell)}$ in terms of the convenient $I_\star$ defined in [2]:

$$\mathcal{I}_{nn'}^{(\ell)}(m_\chi, \alpha, \beta; E) = \sum_{AA...BB} (h_n h_{n'}) \times \frac{q_{\max} q_\star^2 v_\star^2}{2m_\chi \mu_\chi^2 v_{\max}^5} \times I_{\alpha,\beta}^{(\ell)}(E, [q_{ab}, v_{ab}]),$$
(2.44)

where $v_\star$ and $q_\star$ are as defined in (1.11). In each term AA, AB, etc., the limits of integration $u_{a,b}$ are determined by n and n'. For the A-type intervals, $[u_a, u_b] = u_{\max} \times [x_1(n), x_2(n)]$, while for B-type intervals $[u_a, u_b] = u_{\max} \times [x_2(n), x_3(n)]$. The integral function $I_{a,b}^{(\ell)}$ is defined as:

$$I_{a,b}^{(\ell)}(E, [q_{ab}, v_{ab}]) \equiv \int_{q_a}^{q_b} \frac{qdq}{q_\star^2} \left(\frac{q}{q_\star}\right)^a \int_{v_a}^{v_b} \frac{vdv}{v_\star^2} \left(\frac{v}{v_\star}\right)^b P_\ell\left(\frac{v_{\min}}{v}\right) \Theta(v - v_{\min}(q,E)),$$
(2.45)

which has a closed form solution derived in Appendix B of Ref. [2], for integer or non-integer values of a and b.

---

[1] Note that Ref. [2] defines a kinematic scattering matrix $I^{(\ell)}$ using (q,E)-dependent radial basis functions. Our $\mathcal{I}^{(\ell)}$ in this work is proportional to $dI^{(\ell)}/dE$.

2.5 The Partial Rate Matrix

By changing the order of the sums in (2.40), and applying (2.14) to evaluate $\mathcal{R} \left| f_S^2 \right\rangle$, the kinematic scattering matrix can be used to calculate $K_{mm'}^{(\ell)}$ or $dK_{mm'}^{(\ell)}/dE$ from the wavelet-harmonic representations of $\left| g_\chi \right\rangle$ and $\left| f_S^2 \right\rangle$. Defining the *reduced partial rate matrix* as

$$\mathcal{K}_{mm'}^{(\ell)}(g_\chi, f_S^2, F_{\mathrm{DM}}^2; E) \equiv v_{\max}^3 \sum_{nn'} \langle g_\chi | n\ell m \rangle \, \mathcal{I}^{(\ell)}(F_{\mathrm{DM}}^2; E) \, \langle n'\ell m' | f_S^2 \rangle, \qquad (2.46)$$

and assuming that $F_{\mathrm{DM}}^2 \to \sum_{ab}(a, b)$ is given by (2.41),

$$\left. \frac{dR}{dE} \right|_{E,\mathcal{R}} = N_T \rho_\chi \frac{v_{\max}^2}{q_{\max}} \sum_{\alpha,\beta} \bar{\sigma}_0 d_{\alpha,\beta}(E) \sum_{\ell m m'} \mathcal{K}_{mm'}^{(\ell)}(g_\chi, f_S^2, (m_\chi, \alpha, \beta); E) \cdot G_{mm'}^{(\ell)}(\mathcal{R}), \qquad (2.47)$$

where $G_{mm'}^{(\ell)}$ is the Wigner G matrix defined in (2.5). The factor of $v_{\max}^3$ is included in $\mathcal{K}$ to make it nearly dimensionless: that is, it has the same units as f_S^2.

The $\mathcal{K}$ notation accommodates either continuous or discrete final states. In the former case, $\mathcal{K}$ has units of inverse energy, like $f_S^2(\vec{q}, E)$, and

$$\mathcal{K}_{mm'}^{(\ell)}(E) = \frac{q_{\max}}{v_{\max}^2} \left. \frac{dK_{mm'}^{(\ell)}}{dE} \right|_E, \qquad (2.48)$$

in the limit where the sums over n, n' are complete. For discrete final states, we simply define $\mathcal{K}$ in terms of $f_{g \to s}^2(\vec{q})$ rather than $f_S^2(\vec{q}, E)$:

$$\mathcal{K}_{mm'}^{(\ell)}(g_\chi, f_{g \to s}^2, F_{\mathrm{DM}}^2; \Delta E) \equiv v_{\max}^3 \sum_{nn'} \langle g_\chi | n\ell m \rangle \, \mathcal{I}^{(\ell)}(F_{\mathrm{DM}}^2; \Delta E) \, \langle n'\ell m' | f_{g \to s}^2 \rangle. \qquad (2.49)$$

In this case $\mathcal{K}_{mm'}^{(\ell)}$ is dimensionless, and equal to

$$\mathcal{K}_{mm'}^{(\ell)}(\Delta E) = \frac{q_{\max}}{v_{\max}^2} K_{mm'}^{(\ell)}(\Delta E). \qquad (2.50)$$

For convenience, the remaining dimensionful constants can be combined into a dimensionless, basis-dependent quantity k_0 proportional to $\bar{\sigma}_0$ and the experimental exposure time $T_{\exp}$:

$$k_0 \equiv N_T T_{\exp} \bar{\sigma}_0 \rho_\chi \frac{v_{\max}^2}{q_{\max}}. \qquad (2.51)$$

This way the total (differential) rate is given by

$$\left. \frac{dR}{dE} \right|_{E,\mathcal{R}} = \frac{k_0}{T_{\exp}} \sum_{ab} d_{ab}(E) \sum_{\ell m m'} \mathcal{K}_{mm'}^{(\ell)}(g_\chi, f_S^2, (m_\chi, a, b); E) \cdot G_{mm'}^{(\ell)}(\mathcal{R}). \qquad (2.52)$$

For discrete final states (1.7), the total rate is given directly by a sum over final states s:

$$R(\mathcal{R}) = \frac{k_0}{T_{\exp}} \sum_s \sum_{ab} d_{ab}(\Delta E_s) \sum_{\ell m m'} \mathcal{K}_{mm'}^{(\ell)}(g_\chi, f_{g \to s}^2, (m_\chi, a, b); \Delta E_s) \cdot G_{mm'}^{(\ell)}(\mathcal{R}). \qquad (2.53)$$

Both the Python and Julia implementations store $\mathcal{K}_{mm'}^{(\ell)}$ and $G_{mm'}^{(\ell)}$ as 1d vectors

$$\vec{G}_i = G_{mm'}^{(\ell)}, \qquad\qquad\qquad \vec{\mathcal{K}}_i = \mathcal{K}_{mm'}^{(\ell)}, \qquad (2.54)$$

of length

$$\text{len}(\vec{\mathcal{K}}) = \text{len}(\vec{G}) = \frac{(\ell_{\max} + 1)(2\ell_{\max} + 1)(2\ell_{\max} + 3)}{3}, \tag{2.55}$$

with i indexed by

$$\text{index}(\ell, m, m') = \frac{\ell}{3}(4\ell^2 - 1) + (2\ell + 1)(\ell + m) + (\ell + m') \tag{2.56}$$

for $\text{index} = 0, 1, 2, \ldots, \text{len}(G) - 1$.

2.6 Summary

A VSDM calculation proceeds in the following steps:

1. The user selects basis function parameters $v_{\max}$ and $q_{\max}$ that are appropriate for the g_χ and f_S^2 functions in question. The user should also determine which values to use for continuous parameters m_χ, the excitation energy (for dR/dE), and possibly the time of year (if including annual variation in $g_\chi(t)$). It is not necessary to specify the list of $\mathcal{R} \in SO(3)$ detector orientations at this time.

2. The functions g_χ and f_S^2 (or $f_{g \to s}^2$) are projected onto the wavelet-harmonic basis, saving the coefficients $\langle n\ell m|g_\chi\rangle$ and $\langle n\ell m|f_S^2(E)\rangle$ for every version of g_χ and $f_S^2(E)$. At this stage, the user should determine the largest values of $n_{\max}$ and $\ell_{\max}$ that are actually needed in order to represent the functions $|g_\chi\rangle$ and $|f_S^2\rangle$ within the desired precision.

3. The kinematic scattering matrix $\mathcal{I}_{nn'}^{(\ell)}(m_\chi, F_{\text{DM}}^2; E)$ is evaluated for every combination of $(m_\chi, F_{\text{DM}}^2)$ and E that appear in the problem, up to some predetermined values of $\ell_{\max}$ and $n_{\max}^{(q,v)}$.

4. For each combination of g_χ, $f_S^2(E)$, F_{DM}^2, and m_χ, evaluate the reduced partial rate matrix $\mathcal{K}$.

Once the partial rate matrices are known, it is very fast to calculate the scattering rate as a function of detector orientation: after evaluating $G_{mm'}^{(\ell)}(\mathcal{R})$ for every $\mathcal{R}$, the scattering rate is given by the dot product between the vectors $\vec{G}$ and $\vec{\mathcal{K}}$:

$$\left.\frac{dR}{dE}\right|_{E,\mathcal{R}} = \frac{k_0}{T_{\exp}} \sum_{ab} d_{ab}(E) \times \vec{\mathcal{K}} \cdot \vec{G}(\mathcal{R}). \tag{2.57}$$

For a transition to a single discrete final state $g \to s$, the expression for $R(\mathcal{R})$ is identical:

$$R_s(\mathcal{R}) = \frac{k_0}{T_{\exp}} \sum_{ab} d_{ab}(E) \times \vec{\mathcal{K}} \cdot \vec{G}(\mathcal{R}). \tag{2.58}$$

Compared to numerical integration of (1.5), evaluating the $\vec{\mathcal{K}} \cdot \vec{G}$ dot product is extremely fast. Section 5 quantifies the improvement. Step 2 is often the slowest step in this pipeline, at least if $\langle n\ell m|g_\chi\rangle$ and $\langle n\ell m|f_S^2\rangle$ are found by numeric integration. However, if $g_\chi(\vec{v})$ or $f_S^2(\vec{q}, E)$ have already been tabulated, then the wavelet coefficients can be obtained more quickly using discrete transformations. Section 2.7 describes this method, as well as the analytic shortcut from Ref. [2] that can be used for functions (e.g. g_χ) that are defined as the sum of multiple Gaussians.

2.7   Faster Wavelet-Harmonic Transformations

If a function of a 3d variable $f(\vec{u})$ is tabulated on a regular grid of points in spherical coordinates, $\vec{u} = (u, \theta, \phi)$, then the user can rapidly evaluate $\langle n\ell m | f \rangle$ without using the numeric integration methods included in VSDM. Bypassing the numeric integration step can drastically reduce the total VSDM evaluation time. The only nonstandard step in the procedure is the wavelet transformation for $d > 1$ spherical wavelets, which we provide in this section. The Julia implementation of VSDM uses this method as the default for evaluating $\langle f | n\ell m \rangle$.

For calculating $\langle f | n\ell m \rangle$ with discretized 3d functions $f(\vec{u})$, we apply a two-step process. First, for a sequence of spherical shells $|\vec{u}| = u_i$, evaluate

$$f_{\ell m}(u) \equiv \langle \ell m | f \rangle = \int d\Omega \, Y_{\ell m}(\hat{u}) f(\vec{u}) \tag{2.59}$$

for all of the relevant ℓm with $\ell \leq \ell_{\max}$. This can be done by using a discrete spherical harmonic transform, for which there are many pre-existing packages and typically requires tabulating $f(u_i, \theta, \phi)$ on a grid of (θ, ϕ). The sampling density of the grid determines the largest value of ℓ that can be accurately recovered.

Next, for each $f_{\ell m}(u)$ function (evaluated at $u = u_i$ for all u_i on the radial grid), evaluate

$$\langle n\ell m | f \rangle = \langle n | f_{\ell m} \rangle = \int u^2 du \, h_n(u/u_{\max}) f_{\ell m}(u) \tag{2.60}$$

for each $n \leq n_{\max}$, with $n_{\max}$ determined by the total number of u_i points. If the values of u_i are sufficiently finely spaced, then $\langle n | f_{\ell m} \rangle$ can be found using the discrete wavelet transformation: otherwise, a finer grid can be generated from $f_{\ell m}(u_i)$ by interpolation.

It is not necessary to use the same grid of (θ, ϕ) points for each u_i. Near the origin $u \sim 0$, for example, the higher-ℓ moments are expected to vanish, and there is no need for a particularly fine angular grid. In spherical harmonics, the Taylor series of $f(\vec{u})$ has the form

$$f(\vec{u}) \approx \sum_{\ell m} c_{\ell m} u^\ell Y_{\ell m}(\hat{u}), \tag{2.61}$$

so the expected size of $f_{\ell m}(u_i)$ scales as u_i^ℓ for small u_i (assuming that f and its derivatives are finite at the origin).

Unless it is essentially effortless to evaluate $f(\vec{u})$ on a fine grid of points, we recommend the following procedure:

1. identify a list of spherical shells $|\vec{u}| = u_i$ to evaluate $f_{\ell m}(u_i)$. This grid can be irregularly spaced, to increase the sampling density in the places where $f(\vec{u})$ varies most quickly.

2. For each u_i:

   (a) evaluate $f(u_i, \theta, \phi)$ on a grid of (θ, ϕ) points, with the angular grid spacing chosen according to the chosen spherical harmonic transform algorithm and the desired $\ell_{\max}$ for this i

   (b) find $f_{\ell m}(u_i)$ from the tabulated $f(\vec{u})$ via the discrete spherical harmonic transformation.

3. In preparation for the wavelet transformation, evaluate each $f_{\ell m}$ on a finer grid of 2^L regularly-spaced values of $u = \bar{u}_i$, interpolating between the evaluated $f_{\ell m}(u_i)$ if $\bar{u}_i \notin \{u_i\}$.

4. Apply the discrete spherical wavelet transformation (defined below) to obtain $\langle n\ell m | f \rangle$ from the tabulated values of $f_{\ell m}(\bar{u}_i)$, for wavelet generations up to $\lambda_{\max} = L - 1$.

If it is easy to evaluate $f(\vec{u})$ on a denser grid of points, then the interpolation step can be skipped by sampling $f(\vec{u})$ on the same $|\vec{u}| = \bar{u}_i$ spherical shells used by the wavelet transformation.

If interpolation is necessary, it is much faster to evaluate $f_{\ell m}(u_i)$ first and interpolate second (to get $f_{\ell m}(\bar{u}_i)$), rather than interpolating first (to get $f(\bar{u}_i, \theta_j, \phi_k)$ for all $\bar{u}_i$) and getting $f_{\ell m}(\bar{u}_i)$ directly from the spherical harmonic transformation on the interpolated data.

Discrete Spherical Wavelet Transformation: Here we generalize the discrete Haar transformation to define the discrete spherical wavelet transformation for $d \geq 2$.

A 1d function $f(x)$, for $x \in [0, 1]$, is discretized on a regularly spaced grid,

$$f(x_i \leq x < x_{i+1}) \approx \bar{f}_i, \qquad\qquad x_i = i \times \Delta, \qquad\qquad \Delta \equiv 2^{-L}, \qquad (2.62)$$

where $x_0, x_1, \ldots, x_{i_{\max}}, x_{i_{\max}+1}$ mark the boundaries between the different intervals, Δ is the interval width, $i = 0, 1, \ldots, 2^L - 1$, and $\bar{f}_i$ is the volume-weighted average of f in the ith interval:

$$\bar{f}_i \equiv \int_{x_i}^{x_{i+1}} x^{d-1} dx \, f(x). \qquad (2.63)$$

This is equivalent to approximating $f(x)$ by a sum of orthogonal "tophat" basis functions $|\Theta_i\rangle$,

$$\Theta_i(x) \equiv \begin{cases} 1 & \text{if } x_i < x < x_{i+1} \\ 0 & \text{otherwise.} \end{cases} \qquad (2.64)$$

vanishing everywhere outside the ith interval. For this discussion it is simpler not to normalize $|\Theta\rangle$, so we can write

$$f(x) \approx \sum_i \bar{f}_i |\Theta_i\rangle. \qquad (2.65)$$

The error in $f(x) - \sum_i \bar{f}_i |\Theta_i\rangle$ is proportional to Δ and the first derivative of f,

$$f(x) - \sum_i \bar{f}_i |\Theta_i\rangle \lesssim f'(x)\Delta. \qquad (2.66)$$

Rather than finding $\bar{f}_i$ from the integral (2.63), it can be sufficiently well approximated by sampling $f(x)$ at a particular point, $\bar{x}_i$: for arbitrary d,

$$\bar{x}_i \equiv \frac{d}{d+1} \frac{x_{i+1}^{d+1} - x_i^{d+1}}{x_{i+1}^d - x_i^d} \simeq x_i + \frac{1}{2}\Delta + \frac{d-1}{12}\frac{\Delta^2}{x_i} + \mathcal{O}(\Delta^3). \qquad (2.67)$$

For the 3d spherical wavelets, $d = 3$. With this choice, $f(\bar{x}_i) \simeq \bar{f}_i$ to second order in the small Δ expansion:

$$f(\bar{x}_i) - \bar{f}_i \sim \Delta^2 \cdot f''(\bar{x}_i). \qquad (2.68)$$

At any other value of $\bar{x}_i$, the correction to the approximation above would have been proportional to $\Delta \cdot f'(x_i)$, but $\bar{x}_i$ is chosen so that this linear correction cancels itself out.

After evaluating the function f at each $\bar{x}_i$ for $i = 0, 1, \ldots, i_{\max}$, the coefficients of f in the tophat basis are approximated by:

$$f_i = f(\bar{x}_i) \simeq \langle f | \Theta \rangle, \qquad\qquad f(x) \simeq \sum_i f_i |\Theta_i\rangle. \qquad (2.69)$$

By applying a basis transformation, these coefficients can be used to find the wavelet coefficients. Writing the spherical wavelet basis functions as $|n\rangle$, the coefficients of the basis transformation $\langle n|\Theta_i\rangle$ are given by:

$$\langle n|\Theta_i\rangle = \int_{x_i}^{x_{i+1}} x^2 dx \, h_n(x), \tag{2.70}$$

which for the 3d spherical Haar wavelets simplifies to:

$$\langle n < 2^{\lambda_{\max}}|\Theta_i\rangle = \frac{\Delta}{3}\left(3x_i^2 + 3x_i\Delta + \Delta^2\right) \times \begin{cases} A_n & \text{if } x_1(n) < \bar{x}_i < x_2(n) \\ -B_n & \text{if } x_2(n) < \bar{x}_i < x_3(n) \\ 0 & \text{otherwise,} \end{cases} \tag{2.71}$$

$$\langle n \geq 2^{\lambda_{\max}}|\Theta_i\rangle = 0, \tag{2.72}$$

for the $x_{1,2,3}$ defined in (2.26), and $x_i = \Delta \cdot i$. A_n and B_n are defined in (2.27). Because x_i and x_{i+1} are aligned with the wavelet grid, $\langle n|i\rangle$ vanishes for most pairings of n and i. In particular, the wavelet expansion terminates at finite $n \leq i_{\max}$: for higher-frequency wavelets $n > i_{\max}$, $|\phi_i\rangle$ is constant over the full range of support of the wavelet, and so $\langle n|\phi_i\rangle = 0$ by the same orthogonality condition that sets $\langle n|0\rangle$ for the h_0 wavelet. For $n \leq i_{\max}$, the tophat function ϕ_i is narrower than any of the wavelets, and it overlaps with exactly one wavelet from each generation. Consequently, there are only $(i_{\max} + 1) \cdot (\log_2(i_{\max} + 1) + 1)$ many nonzero terms, rather than $(i_{\max} + 1)^2$.

With this result, the wavelet transformation of $f(x)$ can be read off from:

$$f(x) \simeq \sum_{i=0}^{i_{\max}} f_i |\Theta_i\rangle = \sum_{i=0}^{i_{\max}} f_i \sum_{n=0}^{i_{\max}} \langle n|\Theta_i\rangle |n\rangle, \tag{2.73}$$

and the nth wavelet coefficient $\langle f|n\rangle$ is:

$$\langle n|f\rangle \simeq \sum_i f(\bar{x}_i)\langle n|\Theta_i\rangle, \tag{2.74}$$

where the sum over i can be restricted to only those i for which $x_1(n) < \bar{x}_i < x_3(n)$. In terms of the $n \to (\lambda, \mu)$ wavelet indexing, there are $2^{L-\lambda}$ many terms in this sum: as few as 2 for the narrowest wavelets, at $2^{\lambda_{\max}} \leq n \leq 2^{\lambda_{\max}} - 1$, or as many as 2^L for the $n = 0, 1$ wavelets.

Discrete Spherical Harmonic Transformation: A function $f(\theta, \phi)$ is defined on the S^2 sphere, with $0 \leq \theta \leq \pi$ and $0 \leq \phi < 2\pi$. While there exist methods for extracting the spherical harmonic coefficients $\langle \ell m|f\rangle$ from irregular sets of (θ_i, ϕ_j) values, many discrete transformations use regular grids in (θ, ϕ), e.g. an equiangular $N \times (2N-1)$ rectangular grid in θ, ϕ with

$$\theta_i = \frac{i + \frac{1}{2}}{N}\pi, \qquad\qquad \phi_j = \frac{j}{(2N-1)} \times 2\pi \tag{2.75}$$

for $i = 0, 1, \ldots, N-1$ and $j = 0, 1, \ldots, 2(N-1)$. In this scheme, the maximum ℓ for a given N is $\ell_{\max} = N-1$, thus the function $f(u_i, \theta, \phi)$ would need to be evaluated M times for a desired $\ell_{\max}$, where $M = 2\ell_{\max}^2 + 3\ell_{\max} + 1$. Generally, the number of function evaluations needed for a particular $\ell_{\max}$ will scale with $\ell_{\max}^2$, but the exact grid that $f(u_i, \theta, \phi)$ should be evaluated on will depend directly on the choice of algorithm for performing the discrete spherical harmonic transform.

3 Julia Implementation: VectorSpaceDarkMatter.jl

`VectorSpaceDarkMatter.jl` is the Julia implementation of the wavelet-harmonic integration method. The Julia programming language [58] was chosen mostly for its just-in-time compilation, which allows for an even larger speedup over the Python implementation. Julia also offers a few additional benefits, including multiple dispatch, which allows for a more simplified set of functions, and built-in parallelization, allowing for additional speedup proportional to the number of threads.

In order to calculate the rate, there are two main steps: projecting velocity distribution(s) and material form factor(s) onto a basis, and then calculating the partial rate matrix $\mathcal{K}$. This section outlines the steps that one needs to take in order to do these calculations. An example notebook is available at https://github.com/ariaradick/VectorSpaceDarkMatter.jl.

3.1 Projecting a Function

The first step in projecting a function is choosing a basis for the radial part. There are two options included in `VectorSpaceDarkMatter.jl`, spherical Haar wavelets and spherical tophat functions. These are described in more detail in Section 2.2 and Appendix B respectively. We can initialize a basis by running the appropriate constructor function,

```
Wavelet(umax)
Tophat(xi, umax)
```

where `umax` is the (dimensionful) maximum value of the magnitude of the velocity (momentum) in the chosen velocity distribution (material form factor), and `xi` is a list of boundaries for the tophat function on the range [0,1]. Both of these are subtypes of the abstract type `RadialBasis`. Importantly, `umax` should be defined for each function in natural units (units of c for the velocity, and eV for the momentum) for consistency with the rate calculation. It is possible to omit the `umax` option from either, which results in a default value of `umax=1.0`.

In order to calculate the coefficients, one simply needs to use

```
ProjectF(f, nl_max::Tuple{Int,Int}, radial_basis::RadialBasis;
         dict=false, use_measurements=false,
         integ_method=:discrete, integ_params=(;))
```

where `f` is the function to be projected, which can be a plain function, a `GaussianF`, or an `f_uSph`. `nl_max` $= (n_{\max}, \ell_{\max})$ is a tuple containing the number of coefficients in the radial basis, n, and the number of ℓ modes in the spherical harmonics – all m will be calculated for each ℓ. `radial_basis` is either an instance of `Wavelet` or `Tophat`. `use_measurements` will return the numerical integration error along with the values of the coefficients stored in a `Measurements` object from the `Measurements.jl` package [59]. `integ_method` chooses between four options: `:discrete`, which applies the discrete wavelet–harmonic transformation from [60], relying on `FastSphericalHarmonics.jl` for the spherical harmonics part; `:cubature`, which uses the integration method hcubature from the package `HCubature.jl`[61]; and `:vegas` or `:vegasmc`, both options for Monte-Carlo integration from `MCIntegration.jl`[62]. Note that the discrete transformation can only be used with `Wavelet`-type radial bases! `integ_params` is a named tuple containing keyword arguments for the chosen integration method, relevant for the cubature and monte-carlo methods, see the documentation for each package for available options.

If `dict` is false (default), `ProjectF` will return a `ProjectedF` object that stores the values of the coefficients in a matrix,

```
struct ProjectedF{A, B<:RadialBasis}
    fnlm::Matrix{A}
    lm::Vector{Tuple{Int,Int}}
    radial_basis::B
end
```

where `lm` stores the corresponding (ℓ, m) values as a vector and `radial_basis` is a copy of the supplied radial basis object. An instance of this `ProjectedF` can be called as $p(u, \theta, \phi)$ and use the values of the coefficients to evaluate the function at the specified point.

If `dict` is true, `ProjectF` will return an `FCoeffs` object that stores the values of the coefficients in a dictionary,

```
struct FCoeffs{A, B<:RadialBasis}
    fnlm::Dict{Tuple{Int,Int,Int}, A}
    radial_basis::B
end
```

where the `fnlm` values are stored as $(n, \ell, m) \Rightarrow f_{n\ell m}$ and again, `radial_basis` is a copy of the supplied radial basis object. Note that an `FCoeffs` object *cannot* be called as a function, and *cannot* be used in functions calculating the rate. The main advantage to an `FCoeffs` is the ability to update individual coefficients via the function

```
update!(fc::FCoeffs, f, nlm; integ_method=:cubature,
        integ_params=(;))
```

which updates the $\texttt{nlm} = (n, \ell, m)$ coefficient of `fc` with the coefficient obtained by projecting `f` onto the stored radial basis in `fc`. `nlm` can also be a vector of tuples and this will update each coefficient. There are also convenience functions that convert between `ProjectedF` and `FCoeffs` objects,

```
FCoeffs(pf::ProjectedF)
ProjectedF(fcoeffs::FCoeffs)
```

If one wishes to use numerical integration rather than a discrete method, we can speed up the calculation of coefficients greatly by skipping coefficients that we know are zero based on the symmetries of the function. In order to specify these, we include a type

```
f_uSph(f::Function; z_even=false, phi_even=false,
       phi_cyclic=1, phi_symmetric=false, center_Z2=false)
```

The options are

- `z_even` : if $f(x, y, z) = f(x, y, -z)$. Implies $\langle \ell m | f \rangle = 0$ if $(\ell + m)$ is odd.

- `center_Z2` : if $f(\vec{u}) = f(-\vec{u})$ is symmetric under central inversion. Implies $\langle \ell m | f \rangle = 0$ if ℓ is odd.

- `phi_even` : if $f(u, \theta, \phi) = f(u, \theta, -\phi)$. Implies $\langle \ell m | f \rangle = 0$ if m is negative.

- `phi_cyclic` : $f(u, \theta, \phi) = f(u, \theta, \phi + 2\pi/\alpha)$ for integer $\alpha = \texttt{phi\_cyclic}$. Implies $\langle \ell m | f \rangle = 0$ unless m is divisible by α.

- `phi_symmetric` : if $f(u, \theta)$ is independent of ϕ. Takes precedence over other ϕ options. Sets $\langle \ell m | f \rangle = 0$ for all $m \neq 0$.

If one wishes to project a Gaussian function, there are special routines implemented to handle these efficiently. Primarily, we have included a `GaussianF` type, which stores the relevant parameters of the Gaussian,

```
GaussianF(c::Float64, uSph::Vector{Float64}, sigma::Float64)
```

where uSph is the spherical vector indicating the center of the Gaussian, uSph = (u, θ, ϕ), `sigma` is the standard deviation, and c is the overall amplitude. uSph and `sigma` should be in natural units. Instances of this type can be called as $g(u, \theta, \phi)$ to return the value of the Gaussian at the specified point.

In order to check the accuracy of the projection, we include a function that calculates the L^2 norm,

```
f2_norm(f[, a, b])
```

where f can be a function, `ProjectedF`, `FCoeffs`, `GaussianF`, or `Vector{GaussianF}`. In each case, the appropriate norm will be returned:

$$L^2[f] = \int_a^b d^3u\, f^2(\mathbf{u}) = \left(\sum_{n\ell m} f_{n\ell m} \right)^2 \tag{3.1}$$

a and b represent the bounds in spherical coordinates when integrating a function. Using `f2_norm` with a Gaussian will return the analytic integral over all space, unless the bounds a and b are specified. It should be noted that accuracy in the L^2 norm is not necessarily equivalent to accuracy in the rate. *E.g.*, if one has a material with high ℓ support, but a velocity distribution that falls off with ℓ, it may be the case that the rate can be reproduced with high accuracy but the L^2 norm for the material can be off by 50%. So, L^2 accuracy is a sufficient, but not necessary, condition for convergence in the VSDM rate calculation.

Finally, as it can be computationally expensive to calculate coefficients, we include the ability to save and load coefficients to and from CSV files via

```
writeFnlm(outfile, pf)
readFnlm(infile[, radial_basis]; dict=false, use_err=true)
```

where pf is either a `ProjectedF` or `FCoeffs`. In `readFnlm` the `radial_basis` argument is optional if using `Wavelet` for the radial basis as `writeFnlm` saves this information along with the coefficients. If the coefficients were calculated with tophats, it *is* necessary to specify the basis. `use_err` is equivalent to `use_measurements`, choosing whether or not to load the uncertainty values (if present).

3.2   Calculating a Rate

It is necessary to supply some parameters that specify the DM mass m_χ, form factor F_{DM}, SM target mass m_{SM}, and the difference in energy between the initial and final SM states ΔE. We store these parameters in a `ModelDMSM` object,

```
struct ModelDMSM
    a::Int
    b::Int
    mX::Float64
    mSM::Float64
```

```
      deltaE::Float64
end
```

where a and b are parameters of the dark matter form factor such that $F_{\mathrm{DM}}^2 \sim v^a (q/q_{\mathrm{ref}})^b$ (Eqn. 1.38) and the other three parameters should be in units of eV. q_{ref} is taken to be $q_{\mathrm{ref}} = \alpha m_e$ by default (users can change this by editing the relevant variable in the `units.jl` file and installing the package locally). One can construct a `ModelDMSM` either by supplying all five arguments, or by providing `fdm_n = n` where $F_{\mathrm{DM}}^2 \sim (\alpha m_e/q)^{2n}$ as an alternative way of defining F_{DM},

```
ModelDMSM(a, b, mX, mSM, deltaE)
ModelDMSM(fdm_n, mX, mSM, deltaE)
```

Given that our primary concern is anisotropic detectors, the last important factor is the relative rotation between the material and the velocity distribution. This can be specified using a `Quaternion` or `Rotor` object from the `Quaternionic.jl` package [63]. If one wishes to use Euler angles or another way of specifying the rotation, `Quaternionic.jl` provides the conversion functions (see their documentation).

Calculating the rate once all of these ingredients have been assembled is fairly simple,

```
rate(R, model::ModelDMSM, pfv::ProjectedF, pfq::ProjectedF;
     ell_max=nothing, use_measurements=false)
```

where R is the rotation and `ell_max` specifies the maximum ℓ used when both ℓ_{max} for `pfv` and `pfq` are larger than `ell_max`. To scan over rotations it is possible to pass an array of quaternions or rotors for R. For continuous final state energies this returns dR/dE from Eqn. 2.57 while for discrete final state energies this returns R from Eqn. 2.58. Note that this will *not* include the constant prefactor k_0/T_{exp}.

The only part of the computation that takes a significant amount of time is calculating the reduced partial rate matrix $\mathcal{K}$, specifically the intermediate step of calculating the kinematic scattering matrix $\mathcal{I}$. It is possible to pre-compute $\mathcal{I}$ with

```
kinematic_I(lnvnq_max, model::ModelDMSM,
            v_basis::RadialBasis, q_basis::RadialBasis)
```

with `lnvnq_max` $= (\ell_{\mathrm{max}}, n_{\mathrm{max}}^v, n_{\mathrm{max}}^q)$. This is useful if one has many velocity distributions and/or material form factors projected with the same v_{max}, q_{max}, and ΔE. From $\mathcal{I}$ it is straightforward to obtain $\mathcal{K}$,

```
get_mcalK(mcI, pfv::ProjectedF, pfq::ProjectedF;
          ell_max=nothing, use_measurements=true)
```

where `mcI` is the precomputed $\mathcal{I}$ from `kinematic_I`. It is also possible to calculate $\mathcal{K}$ from a `ModelDMSM` instead, which will calculate $\mathcal{I}$ as an intermediate step,

```
get_mcalK(model::ModelDMSM, pfv::ProjectedF, pfq::ProjectedF;
          ell_max=nothing, use_measurements=true)
```

Both versions will return the values in a flattened vector in an `McalK` object,

```julia
struct McalK{A<:Union{Measurement,Float64}}
    K::Vector{A}
    ell_max::Int64
    vmax::Float64
    qmax::Float64
end
```

and these values can be saved and loaded via

```julia
writeK(outfile, mcK::McalK)
readK(infile[, vmax, qmax];
      use_err=true)
```

where the basis maximums can optionally be specified when reading the $\mathcal{K}$ matrix. The dimensionful, basis-independent partial rate matrix K (or dK/dE in the case of continuous final states) only differs from this $\mathcal{K}$ by a factor of $v_{\max}^2/q_{\max}$ and can be computed with

```julia
partial_rate(K::McalK)
```

An `McalK` object can be passed to the `rate` function as

```julia
rate(R, mcK::McalK)
```

where R is the rotation, equivalent to the case when not providing a pre-computed $\mathcal{K}$ matrix.
    The constants in Eqn 2.58 can be incorporated via an `ExposureFactor` struct,

```julia
struct ExposureFactor
    N_T::Float64
    sigma0::Float64
    rhoX::Float64
    total::Float64
end
expf = ExposureFactor(N_T, sigma0, rhoX)
```

where `N_T` is the number of individual targets, $\mathtt{N\_T} = M_T/m_{\mathrm{cell}}$ with M_T the target mass and m_{cell} the mass of the unit cell, `sigma0` is the reference cross-section in cm^2, $\mathtt{rhoX} = \rho_\chi$ is the local DM density in GeV / cm^3. `total` is computed based on the given inputs and is used to calculate the rate; it is equal to $\mathtt{total} = k_0 q_{\max}/(T_{\mathrm{exp}} v_{\max}^2)$, where the constant k_0 is defined in Eqn. 2.51. This can be passed as the second argument in any of the rate functions,

```julia
rate(R, exp::ExposureFactor, model::ModelDMSM, pfv::ProjectedF,
    pfq::ProjectedF; ell_max=nothing, use_measurements=false,
    t_unit='s')
rate(R, exp::ExposureFactor, mcK::McalK; t_unit='s')
```

which will return the rate for the given parameters in the specified units according to `t_unit`, which can either be 's' or 'y' for seconds or years.
    Finally, to get the total event count for a specified parameter point, we provide `N_events`,

```
N_events(R, T_exp_s, exp::ExposureFactor, model::ModelDMSM,
         pfv::ProjectedF, pfq::ProjectedF; ell_max=nothing,
         use_measurements=false)
N_events(R, T_exp_s, exp::ExposureFactor, mcK::McalK)
```

which is similar to rate called with a provided `ExposureFactor`, but also includes the total detector run time `T_exp_s`, which should be in units of seconds.

4 Python Implementation: VSDM

The Python implementation is available on the Python Package Index (PyPI) as `vsdm`, and on Github at https://github.com/blillard/vsdm. It can be installed via `pip`:

$$\texttt{pip install vsdm}$$

Beyond the core package, the Github repository includes several example notebooks for the $\langle f | n\ell m \rangle$ and rate calculations, as well as an efficient program for evaluating $\langle g_\chi | n\ell m \rangle$ for Standard Halo Model velocity distributions, `SHM_gX.py`.

A rate calculation proceeds in two stages: first, the functions g_χ and f_S^2 are projected onto velocity and momentum vector spaces; then, the scattering rate as a function of detector orientation is found by evaluating the partial rate matrix $K_{mm'}^{(\ell)}$. In the Python implementation, a few fundamental `class` objects handle each stage of the calculation:

- `Fnlm`: calculates and saves the coefficients $\langle f | n\ell m \rangle$ for functions $f(\vec{u})$ of a 3d variable $\vec{u}$

- `McalI`: calculates and saves the kinematic scattering matrices $\mathcal{I}^{(\ell)}(E)$ for each DM model $(m_\chi, F_{\mathrm{DM}}^2)$ and energy transfer E

- `McalK`: calculates the dimensionless partial rate matrices $\mathcal{K}^{(\ell)}(E)$, for combinations of $\langle g_\chi | n\ell m \rangle$, $\langle f_S^2(E) | n\ell m \rangle$, and $\mathcal{I}^{(\ell)}(E)$.

`McalK` includes several functions (class methods) for calculating the rate $R \propto G_{mm'}^{(\ell)} \mathcal{K}_{mm'}^{(\ell)}$ from (2.4), with or without the exposure factor k_0.

Below, we provide a brief description of the contents of the `vsdm` package. Sections 4.2 and 4.3 describe the usage of each object and its variants in detail. Appendix C summarizes the workflow for a `vsdm`, showing in a single page how to perform all steps in a standard computation. More detailed demonstrations are available on the Github repository, in the `tools` directory.

Constants and Basic Functions:

- `units.py`: defines the units for velocity and energy used in the internal calculations.[2] This file also defines convenient dimensionful parameters $\texttt{km\_s} = 1\,\mathrm{km/s}$ and $\texttt{g\_c} = c$ for velocities; $\texttt{qBohr} = \alpha m_e c$ for momenta; and `eV`, `keV`, `MeV`, etc. for energies. This file also defines the $q_{\mathrm{ref}} = \texttt{q0\_fdm}$ and $v_{\mathrm{ref}} = \texttt{v0\_fdm}$ parameters that appear in (1.38). By default, $q_{\mathrm{ref}} = \texttt{qBohr}$ and $v_{\mathrm{ref}} = c$.

---

[2]All built-in choices use $\hbar = 1$, but the fundamental unit for velocity can be changed from its default of $c = 1$ by editing `VUNIT_c`.

- `utilities.py`: includes several general-purpose functions, including our implementation of spherical harmonics and associated Legendre polynomials; the exposure factor `g_k0` $= k_0$; the DM scattering form factor functions `fdm2_n(n,q)` $= (q_{\mathrm{ref}}/q)^{2n}$ and `fdm2_ab(a,b,q,v)` $= (q/q_{\mathrm{ref}})^a (v/c)^b$; transformations between Cartesian and spherical coordinate systems; the numerical integration function `NIntegrate`; and the interpolation objects `Interpolator1d` and `Interpolator3d`, for functions of u or $\vec{u}$.

- `portfolio.py`: defines a consistent format for saving and reading HDF5 files.

Projection and Basis Functions:

- `basis.py`: defines the `Basis` and `Basis1d` classes, for generic $|n\ell m\rangle$ and $|n\rangle$ basis functions (respectively). Used by `Fnlm`: see Section 4.2 for detail.

- `haar.py`: contains the wavelet-specific functions for Haar and spherical Haar wavelets, including the wavelet transformation (`haar_transform`) and its inverse transformation (`haar_inverse`) for arbitrary d dimensional spherical wavelets, reverting to the standard Haar wavelets at $d = 1$. Used by `Basis` for the wavelet-harmonic $|n\ell m\rangle$ functions.

- `projection.py`: includes the `Fnlm` class, which stores, manipulates and saves the $\langle f | n\ell m \rangle$ coefficients; and `EvaluateFnlm`, which automatically calculates $\langle f | n\ell m \rangle$ for a predetermined list of (n, ℓ, m) coefficients.

- `Gaussian.py`: implements an analytic shortcut for calculating $\langle f | n\ell m \rangle$ for functions $f(\vec{u})$ that are provided as the sum of several Gaussians.

- `adaptive.py`: applies a wavelet-specific adaptive technique in `WaveletFnlm`. This method estimates the values of $\langle f | n'\ell m \rangle$ coefficients from the $n \approx n'/2$ previously evaluated coefficients $\langle f | n\ell m \rangle$, and calculates additional $\langle f | n'\ell m \rangle$ from numerical integration until the wavelet-harmonic expansion converges to within a specified absolute accuracy goal.

Rate Calculation and Rotations:

- `analytic.py`: provides the $\mathcal{I}^{(\ell)}$ integrals for piecewise-constant radial basis functions, following (2.45).

- `matrixcalc.py`: defines the `McalI` object, which evaluates the kinematic scattering matrix $\mathcal{I}^{(\ell)}$ for arbitrary radial basis functions. For wavelets and tophat basis functions, `McalI` uses the `analytic.py` results for the (2.45) integral. `McalI` can also be used with arbitrary radial basis functions, by integrating (2.39) numerically.

- `wigner.py`: evaluates the Wigner G matrix for real spherical harmonics in `WignerG`

- `ratecalc.py`: defines the `McalK` object, which calculates and stores the partial rate matrices $\mathcal{K}^{(\ell)}$. The class `McalK` includes methods for calculating the scattering rate as a function of detector orientation $\mathcal{R} \in SO(3)$, for rotation operators $\mathrm{R} = \mathcal{R}$ provided in the quaternionic representation of (1.15).

In this section we use the `vsdm` built-in units, e.g. `km_s` $= 1\,\mathrm{km/s}$, but otherwise we keep the `vsdm` prefix for functions and classes from the `vsdm` namespace, as if the package has been loaded via:

```
import vsdm
from vsdm.units import *
```

4.1 Defining a Function

As the first step of the calculation, the user must provide the input distributions $g_\chi(\vec{v})$ and $f_S^2(\vec{q}, E)$ as functions of 3d velocity $\vec{v}$ and momentum $\vec{q}$, in spherical coordinates: $\vec{u} = (u, \theta, \phi)$, where θ is the polar angle and $u = |\vec{u}|$. After defining a function $f(\vec{u})$, the symmetries of $f(\vec{u})$ can be specified by adding decorations to the Python function. Just as in the Julia version, the options are:

- `z_even = True` : if $f(x, y, z) = f(x, y, -z)$. Sets $\langle \ell m | f \rangle = 0$ for odd $(\ell + m)$.

- `phi_even = True` : if $f(u, \theta, \phi) = f(u, \theta, -\phi)$. Sets $\langle \ell m | f \rangle = 0$ for negative m.

- `phi_cyclic = n` : if $f(u, \theta, \phi) = f(u, \theta, \phi + 2\pi/n)$. Sets $\langle \ell m | f \rangle = 0$ unless m is divisible by `n`.

- `phi_symmetric` : if $f(u, \theta)$ is independent of ϕ. Sets $\langle \ell m | f \rangle = 0$ for all $m \neq 0$.

- `center_Z2` : if $\vec{f}(\vec{u}) = \vec{f}(-\vec{u})$ is symmetric under central inversion. Sets $\langle \ell m | f \rangle = 0$ for odd ℓ.

For example:

```
def f_uSph(uSph):
        ...
        return f_u
f_uSph.z_even = True
f_uSph.phi_even = True
f_uSph.phi_cyclic = 2
f_uSph.phi_symmetric = False
f_uSph.center_Z2 = True
```

defines a function that has the symmetries of a rectangular box. It is symmetric under the reflections $z \to -z$ and $\phi \to -\phi$ (a.k.a. $y \to -y$), and also has a Z_2 rotational symmetry about the z axis.

The combination of `z_even` and `phi_cyclic = 2` means that f is also center-symmetric. VSDM would set `center_Z2 = True` in this case automatically, but for clarity we have added it explicitly to `f_uSph`'s list of discrete symmetries. By default, VSDM assumes that f has no symmetries, so the line `f_uSph.phi_symmetric = False` can be omitted without consequence.

If $f(\vec{u})$ can be written as a sum of Gaussian functions, `vsdm` uses some analytic shortcuts to simplify the evaluation of $\langle f | n \ell m \rangle$. We use the normalization:

$$f(\vec{u}) = \sum_i c_i \frac{1}{\sigma_i^3 (2\pi)^{3/2}} \exp\left(-\frac{|\vec{u} - \vec{u}_i|^2}{2\sigma_i^2}\right), \tag{4.1}$$

with each term denoting a Gaussian of width σ_i, centered at $\vec{u}_i$, with L^1 norm c_i. In `vsdm`, a function of this type can be expressed as a list of $\{c_i, \vec{u}_i, \sigma_i\}$ parameters, with $\vec{u}_i$ in spherical coordinates:

```
v_1 = (250*km_s, 0, 0)
sigma_1 = 238*km_s / math.sqrt(2)
v_2 = (320*km_s, 1.2*math.pi, 4.0)
sigma_2 = 188*km_s / math.sqrt(2)
gvec_list = [(0.8, v_1, sigma_1), (0.2, v_2, sigma_2)]
gX = vsdm.GaussianF(gvec_list)
```

Each entry in `gvec_list` can be provided as a tuple, list, or `numpy` array. This excerpt defines a Gaussian-type function, `gX`, with `len(gvec_list) = 2` many Gaussian contributions.

Just like `f_uSph` above, the function `gX` can be decorated with the optional attributes `z_even`, `center_Z2`, etc. In this example, even though $f(\vec{u})$ is symmetric under reflections along the $\vec{v}_1 \times \vec{v}_2$ axis, the plane of symmetry is not aligned with the $\vec{u}$ coordinate system, so there are no sets of $\langle f | \ell m \rangle$ coefficients that systematically vanish.

4.2 Projecting a Function

Once $g_\chi(\vec{v})$ and $f_S^2(\vec{q}, E)$ are defined, we must specify the velocity and momentum basis functions $|n\ell m\rangle$,

$$|n\ell m\rangle = r_n(u) \cdot Y_{\ell m}(\theta, \phi). \tag{4.2}$$

The built-in options for r_n include d-dimensional spherical Haar wavelets, and similarly normalized spherical "tophat" basis functions, for $x \equiv u/u_{\max}$ defined on $0 \le x \le 1$. Both of these options are piecewise-constant, allowing $\mathcal{I}$ to be calculated analytically. It is possible, however, for the user to add their own radial basis functions $r_n^{(\ell)}$ to the `class Basis1d`: see for example the `laguerre` option included in `Basis1d._r_n_x`. We do not recommend using any set of basis functions that forces $\mathcal{I}^{(\ell)}$ to be evaluated via numerical integration.

In `vsdm`, the basis functions are specified by a dictionary object, that at a minimum defines the function type (wavelet or tophat) and the value of $u_{\max}$. For example,

```
basisV = dict(uMax=820*km_s, type='wavelet')
```

identifies a wavelet basis with $v_{\max} = 820 \, \text{km/s}$. This dictionary can be used to create a `Basis` or `Fnlm` type class instance.

Given a function $f(\vec{u})$ and a set of basis functions, any of the classes `Fnlm`, `EvaluateFnlm`, and `WaveletFnlm` can be used to calculate the coefficients $\langle f | n\ell m \rangle$:

- `EvaluateFnlm` converts a function $f(\vec{u})$ into $\langle f | n\ell m \rangle$ coefficients for arbitrary basis functions $\phi_{n\ell m}(\vec{u})$ using numerical integration.

- `WaveletFnlm` uses properties of Haar wavelets to enhance and reduce evaluation time of `EvaluateFnlm`.

- `Fnlm` is the underlying class for both methods, and can write and read the $\langle f | n\ell m \rangle$ coefficients to/from CSV and HDF5 files.

Each class has a `self.f_nlm` dictionary, which saves the value of $\langle f | n\ell m \rangle$ with the key `(n, l, m)`:

$$\texttt{self.f\_nlm[(n, l, m)]} = \langle f | n\ell m \rangle. \tag{4.3}$$

When calculating the partial rate matrix, or saving the coefficients to an HDF5 file, `Fnlm` converts this dictionary into a more convenient 2d array, `self.f_lm_n`, discussed in subsection 4.2.3.

4.2.1 Integration

The actual evaluation of $\langle f | n\ell m \rangle$ inner products is implemented in the class `Basis`, which is a parent class of Fnlm, via the `Basis.getFnlm()` method:

$$\text{Basis.getFnlm(f\_uSph, nlm, integ\_params)} = \int \frac{d^3 u}{u_0^3} \phi_{\text{nlm}}(\vec{u}) \, \text{f\_uSph}(\vec{u}), \quad (4.4)$$

with the dictionary `integ_params` providing the instructions for evaluating the integral numerically. Here $\phi_{n\ell m} = |n\ell m\rangle$ and $u_0 = \text{Basis.u0}$, which for wavelet and tophat basis functions is equivalent to u_{max}.

There are two numerical integration methods implemented in `vsdm.NIntegrate`: a Gaussian quadrature method, from `scipy.integrate`; and the adaptive Monte-Carlo method, `vegas`. The dictionary `integ_params` contains an item 'method', which is set to either 'gquad' or 'vegas', respectively. In the former case:

```
integ_params = dict(method='gquad', rtol_f=1e-5, atol_f=1e-8)
```

tells NIntegrate to use quadrature, with a relative accuracy goal of `rtol = rtol_f` and an absolute accuracy goal of `atol = atol_f`.

Alternatively, with method='vegas', the dictionary must specify the number of iterations, `nitn`, and the number of integrand evaluations per iteration, `n_eval`:

```
integ_params = dict(method='vegas', nitn=10, neval=1e4,
                    nitn_init=3, neval_init=1e3)
```

This instructs `vegas` to use 10^4 evaluations for each of 10 iterations of the adaptive method. The optional `_init` parameters add a lower-resolution initialization step, allowing `vegas` to adjust its sampling density to match the integrand before performing the main `nitn`, `neval` integration.

For low-dimensional integrals, including the 3d $\langle f | n\ell m \rangle$ inner products, we find that quadrature is typically faster. With either method, the dictionary can include an optional `integ_params['verbose'] = True` entry, which tells NIntegrate to print certain intermediate results.

The output of NIntegrate includes an estimate of the uncertainty from the numerical integration. For `vegas` this is the standard deviation from the Monte Carlo integration; for `scipy.integrate`, the uncertainty is calculated from the `rtol` and `atol` parameters. In both cases, we use the `gvar` Python package to combine the mean value and standard deviation into a single Gaussian variable. The `gvar` package adjusts the 1σ widths as needed when adding or multiplying `gvar` variables, making it a convenient way to propagate the uncertainty from the numerical integration. The mean value and standard deviation can be extracted from a `gvar` object via `.mean` and `.sdev`:

```
result = vsdm.NIntegrate(integrand, volume, integ_params)
value = result.mean
error = result.sdev
```

Here `value` and `error` are float-valued, and `result` is an object of type `_gvarcore.GVar`. Arithmetic with `gvar` objects can be substantially slower, so Fnlm includes a boolean parameter `use_gvar` that can be set to `False` to replace `result` with `result.mean` for the relevant calculations.

Note that the `gquad` integration method sometimes remains accurate to around one part in 10^8 even when `rtol` is substantially larger than 10^{-8}. In these cases, the reported error (`result.sdev` from `NIntegrate`) is substantially overestimated. For this reason, the `use_gvar` option is more useful when using the Monte Carlo integrator `vegas`.

4.2.2 Evaluation

The $\langle f | n\ell m \rangle$ calculation can be carried out using `Fnlm`, by running `getFnlm()` for each value of `nlm`:

```
gXnlm = vsdm.Fnlm(basisV, f_type='gX', use_gvar=bool)
for nlm in nlmlist:
    gXnlm.updateFnlm(f_uSph, nlm, integ_params,
                     csvsave_name='filename.csv')
```

given some `integ_params` and some list of (n, ℓ, m) indices. The `updateFnlm` function automatically adds $\langle f | n\ell m \rangle$ to `self.f_nlm`, either as a gvar variable or as a float, according to whether `use_gvar` is True or False. Here we have included an optional label `f_type`, to indicate that this function is a $g_\chi(\vec{v})$ velocity distribution, and we provided a CSV file destination (`filename.csv`) for saving the $\langle f | n\ell m \rangle$ coefficients. The `f_type` is only relevant when saving the results to HDF5 files.

If a file name is provided for `csvsave_name`, then `updateFnlm` automatically saves each new value of $\langle f | n\ell m \rangle$ to the end of the CSV file immediately after evaluation. This option is particularly convenient whenever the $\langle f | n\ell m \rangle$ integration time is relatively slow, as the coefficients are saved even if `Fnlm` is interrupted before it has processed the full `nlmlist`.

`EvaluateFnlm` and `WaveletFnlm` both inherit the class methods of `Fnlm`, and can evaluate batches of (n, ℓ, m) coefficients upon initialization. With `EvaluateFnlm`, this is done by supplying a list `nlmlist` of (n, ℓ, m) tuples:

```
gXnlm = vsdm.EvaluateFnlm(basisV, f_uSph, integ_params,
                          nlmlist=[(0,0,0), (1,0,0), ...],
                          csvsave_name='filename.csv',
                          f_type='gX', use_gvar=bool)
```

This example initializes an `Fnlm` type class instance called `gXnlm`, and calculates $\langle f | n\ell m \rangle$ in series for all of the (n, ℓ, m) coefficients supplied by `nlmlist`. Here $f(\vec{u})$ given by `f_uSph`, and the basis functions $\phi_{n\ell m}(\vec{u})$ are defined according to the `basisV` dictionary.

Additional $n\ell m$ coefficients can be calculated (and added to `self.f_nlm`) using the class method `Fnlm.updateFnlm()`. In this example, running

```
gXnlm.updateFnlm(f_uSph, (0,14,2), integ_params,
                 csvsave_name='filename.csv')
```

would calculate $\langle f | 0, 14, 2 \rangle$, add its value to `gXnlm.f_nlm`, and append the result to the end of the CSV file `filename.csv`. The values of `integ_params` (and even `f_uSph`) can be changed. Running `updateFnlm` for an `nlm` already present in `self.f_nlm` will overwrite the older value.

WaveletFnlm: When using wavelet basis functions, we recommend using `WaveletFnlm` instead of `EvaluateFnlm` whenever high precision in $\langle f | n\ell m \rangle$ is needed (e.g. better than 10^{-8} relative precision). `WaveletFnlm` calculates the $\langle f | n\ell m \rangle$ coefficients in batches of fixed

(ℓ, m), for all $n < 2^p$ for some power p. Rather than integrating over $0 \leq x \leq 1$ for $n = 0$ and $n = 1$, and $0 \leq x \leq 0.5$ for $n = 2$ (etc.), `WaveletFnlm` calculates the $\langle f | N\ell m \rangle$ coefficients for 2^p many tophat basis functions $r_N(x)$, each of width 2^{-p}, and applies a discrete transformation to convert $\langle f | N\ell m \rangle$ into wavelet coefficients $\langle f | n\ell m \rangle$. This can reduce the number of integrand evaluations by a factor of p without negatively impacting the integration accuracy. `EvaluateFnlm`, on the other hand, calculates the integral $\langle f | n\ell m \rangle$ directly using wavelet functions, requiring relatively more integrand evaluations to reach the same precision as `WaveletFnlm`.

In the sample implementation of `WaveletFnlm` below, the $\langle f | n\ell m \rangle$ batch calculation is performed in a separate line:

```
gXnlm = vsdm.WaveletFnlm(basisV, f_uSph, integ_params, f_type='gX',
                         csvsave_name='filename.csv')
gXnlm.initialize_lm((0,0), 9)
```

The `initialize_lm(lm, power2)` function evaluates all $\langle f | n\ell m \rangle$ coefficients with $n < 2^9$ and `lm = (0,0)`. If `self.csvsave_name` is not `None`, then `WaveletFnlm` will also save the results to the specified CSV file. `WaveletFnlm` can be parallelized simply by assigning different (ℓ, m) modes to different processors.

The complete wavelet-harmonic projection can be performed (in series) with a single line, by supplying a non-empty dictionary `power2_lm` when initializing `WaveletFnlm`: for example,

```
lm_p = {(0,0):10, (2,0):9, (2,2):9, (4,0):8, ...}
gXnlm = vsdm.WaveletFnlm(basisV, f_uSph, integ_params, power2_lm=lm_p)
```

sets $p = 10$ for $(\ell, m) = (0, 0)$, $p = 9$ for $\ell = 2$, and so on. One expects the magnitudes of $\langle f | n\ell m \rangle$ to decrease as ℓ becomes large, so the user may wish to keep somewhat larger values of p at small ℓ.

`WaveletFnlm` also includes a `refine_lm` function, that implements the polynomial wavelet extrapolation method from [2] to predict the values of higher-n wavelet coefficients from the set of previously calculated $\langle f | n\ell m \rangle$. The main reason to use `refine_lm` is for functions $f(\vec{u})$ that may have large, isolated fluctuations at small scales in u: `refine_lm` subdivides this region of u with progressively narrower basis functions, until the wavelet expansion converges.

If all values of $\langle f | n\ell m \rangle$ for $n = 2^{p-1}, 2^{p-1} + 1, \ldots, 2^p - 1$ match the predictions from the ref. [2] wavelet extrapolation method to within a tolerance set by `atol_fnlm`, then there is nothing for `refine_lm` to do. Otherwise, for each n where this difference is larger than `atol_fnlm`, the `refine_lm` function evaluates the descendant wavelets $n \to 2n$ and $n \to 2n+1$ via numerical integration, and checks again to see if the wavelet extrapolation method has converged. The process continues until all $\langle f | n\ell m \rangle$ coefficients match the predictions from the polynomial extrapolation, with the maximum number of wavelet subdivisions set by `max_depth`. For example,

```
gXnlm.refine_lm((0,0), max_depth=3)
```

applies this process to the $(\ell, m) = (0, 0)$ harmonic mode, allowing up to three stages of wavelet subdivision. This optional part of the calculation can also be completed upon initialization of `WaveletFnlm`, by setting `refine_at_init = True` and supplying values for `max_depth` and the absolute and relative tolerances `atol_fnlm` and `epsilon`. By default, `refine_lm` applies the cubic order wavelet extrapolation, but the polynomial order can be changed by setting `p_order` to e.g. 1 for linear extrapolation.

4.2.3   Array Format for $\langle f | n\ell m \rangle$

The dictionary format for `Fnlm.f_nlm` is convenient when adding $\langle f | n\ell m \rangle$ coefficients one at a time, particularly for sparse lists. However, when calculating the partial rate matrix, it is much faster to convert this data into an array, so that the double sum $\sum_{n,n'}$ in (2.46) can be implemented via matrix multiplication. Fnlm organizes these coefficients into a 2d array, `Fnlm.f_lm_n`, where each row corresponds to a different (ℓ, m), and each column corresponds to n. The mapping between the row label and (ℓ, m) is stored in `Fnlm.lm_index`.

To populate the `self.f_lm_n` array with values from `self.f_nlm`, simply run the line:

```
gXnlm._makeFarray(use_gvar=bool)
```

with the option to set `use_gvar = False` for this array even if `self.use_gvar = True` for the class instance. For large sets of coefficients (e.g. `len(nlmlist)` $\gtrsim 10^5$) this step may take several seconds, so `vsdm` usually waits to calculate `f_lm_n` until it is needed, e.g. when saving an HDF5 file or before calculating the partial rate matrix.

4.2.4   Reading and Writing CSV Files

There are two built-in file formats for saving $\langle f | n\ell m \rangle$ coefficients from Fnlm: CSV and HDF5. The CSV format is simpler: each row saves the values of

```
n, l, m, fnlm.mean, fnlm.sdev
```

for each value of `fnlm` in the `Fnlm.f_nlm` dictionary. The indices n, ℓ, and m are not necessarily in any sort of order. Every Fnlm type object has a class method `writeFnlm_csv()`,

```
Fnlm.writeFnlm_csv(csvfile, nlmlist=None)
```

which when called with `nlmlist=None` will save all values in `self.f_nlm` to the CSV file. Alternatively, supplying a nonempty list for `nlmlist` $= \{(n, \ell, m)\}$ will save only those values of $\langle f | n\ell m \rangle$. Rather than overwriting any existing lines of `csvfile`, `writeFnlm_csv()` appends the new values to the end of the file.

Once a CSV file has been created with this format, it can be used to populate a new Fnlm object. For example:

```
gXnlm = vsdm.Fnlm(basisV, f_type='gX', use_gvar=bool)
gXnlm.importFnlm_csv(filename.csv, use_gvar=bool)
```

The second line reads the values in the CSV file `filename.csv`, and adds each $\langle f | n\ell m \rangle$ to the `self.f_nlm` dictionary. If the file includes multiple rows with the same value of `nlm` $= (n, \ell, m)$, then the last entry (the most recent) is the one ultimately saved to `f_nlm[nlm]`.

4.2.5   Reading and Writing HDF5 Files

The CSV format is human-readable and highly convenient when working with one or two functions at a time, but it becomes increasingly challenging to organize larger sets of g_χ or f_S^2 distributions. In this scenario the HDF5 format is preferable: all of the data can be saved to the same file, and each dataset within the file can store the relevant metadata needed to interpret the tabulated coefficients, e.g. the description of the $|n\ell m\rangle$ basis functions, and any parameter values ϑ that went in to the definition of $f(\vec{u}, \vartheta)$.

By running the function `Fnlm.writeFnlm`,

```
gXnlm.writeFnlm(filename.hdf5, modelName, use_gvar=bool)
```

the array `self.f_lm_n` and the ℓm index `self.lm_index` are both saved as datasets within the HDF5 subgroup `typeName/modelName`, where `typeName` is given by the function's `f_type` (e.g. 'gX' for the gXnlm example), and `modelName` is a name string chosen by the user. Unless otherwise specified, the `f_lm_n` dataset name is 'fnlm', and the (ℓ, m) index dataset is saved with the name 'lm_index'. The default dataset names `fnlm`, `lm_index` are defined in `portfolio.py`.

The `modelName` can be organized with additional subgroups: for example, setting

```
modelName = 'SHM/vEsc_544/vE_250'
```

would generate a subgroup SHM under the group gX, with nested subgroups vEsc_544 and vE_250, e.g. for an SHM velocity distribution with escape velocity 544 km/s and Earth speed 250 km/s.

Every HDF5 dataset comes with a `.attrs` dictionary to store auxiliary data. By default, `writeFnlm` includes all entries in its `self.basis` dictionary, such as the type and the values of uMax and/or u0. The easiest way to feed additional data to `.attrs` is to add them as items in `self.basis` before running `writeFnlm`. For the gXnlm example above, one might add:

```
gXnlm.basis['vEsc'] = 544.*km_s
gXnlm.basis['vCirc'] = 238.*km_s
gXnlm.basis['vE'] = 250.*km_s
gXnlm.writeFnlm(filename.hdf5, 'SHM/vE_Feb15', alt_type=None)
```

to ensure that the SHM parameter values are saved with the data. The optional `alt_type` argument can be used to save the data under a different `typeName = alt_type`, rather than the default value `typeName = self.f_type`. This capability is included for cases such as the one encountered in (1.35), where an analysis includes multiple types of response functions for the same material.

To import the $\langle f | n\ell m \rangle$ coefficients from an HDF5 file, the user only needs to specify the `modelName` and the file name:

```
gXnlm = vsdm.Fnlm(basisV, f_type='gX', use_gvar=bool)
dataFnlm, basis_info = gXnlm.importFnlm(filename.hdf5, modelName)
```

By default, `importFnlm` looks for the dataset 'fnlm' under `f_type/modelName`, but if a string is provided for `alt_type` then Fnlm will import the coefficients from `alt_type/modelName` instead.

The class method `self.importFnlm` does not automatically populate `self.basis` with the values from the HDF5 dataset's `.attrs` dictionary. Similarly, running `importFnlm` fills `self.f_nlm` with the saved $\langle f | n\ell m \rangle$ coefficients, but not the array `self.f_lm_n`. However, both types of information can be recovered from the `return` value of `importFnlm`, as demonstrated above. Here, the first item `dataFnlm` is the `f_lm_n` array from the dataset, and `basis_info` is the `.attrs` dictionary.

Automatic Dataset Naming: The function `Fnlm.writeFnlm()` uses a naming scheme to avoid overwriting existing datasets. If a dataset by the name `fnlm` already exists in the subgroup `typeName/modelName`, then `writeFnlm` saves the newest dataset under the modified names `fnlm__1`, `fnlm__2`, etc., appending a double underscore and an integer to the original dataset name.

The same procedure is used for the `lm_index` dataset. Each `lm_index.attrs` dictionary (`lm_index__1.attrs`, etc.) saves the name of its matching `fnlm` dataset, under the key `dname`. So, the $\langle f | n\ell m \rangle$ dataset matching a given `lm_index` can be recovered via:

```
dsetname = lm_index__2.attrs['dname']
```

If all edits to the HDF5 file have been made using the `Portfolio` class, then one expects `dsetname = fnlm__2` to match the suffix of `lm_index`.

To import datasets that are not named "fnlm" and "lm_index," run `importFnlm` with the dataset names specified via the `d_fnlm` and `lm_ix` options:

```
gXnlm.importFnlm(filename.hdf5, modelName,
                 d_fnlm='fnlm__2', lm_ix='lm_index__2')
```

4.3 Calculating A Rate

Following (2.53) and (2.52), the scattering rate R, or the differential scattering rate dR/dE, can be rapidly evaluated from the partial rate matrix $\mathcal{K}$ defined in (2.46). The precursor to $\mathcal{K}$ is the kinematic scattering matrix $\mathcal{I}_{nn'}^{(\ell)}(E, m_\chi, \ldots)$ of Section 2.4, which must be evaluated for every DM particle model (m_χ, F_{DM}^2). In `vsdm` this part of the calculation is handled by the class `McalI` from `matrixcalc.py`.

Section 4.3.1 calculates $\mathcal{I}$, and describes how it can be saved in the HDF5 format. Section 4.3.2 shows how to calculate the partial rate matrix $\mathcal{K}$ from $\mathcal{I}$ and the vector space representations of $g_\chi(\vec{v})$ and $f_S^2(\vec{q}, E)$. After assembling the Wigner G matrices for each rotation $\mathcal{R} \in SO(3)$ in Section 4.3.3, Section 4.3.4 evaluates the scattering rate (or differential rate) for every detector orientation.

4.3.1 Kinematic Scattering Matrix $\mathcal{I}$

Like the Fnlm classes, `McalI` is a basis-dependent object, and it defines the velocity and momentum basis functions using the same `basisV` and `basisQ` dictionaries. Unlike Fnlm, `McalI` depends on the details of the DM particle model, via the DM mass m_χ and the DM–SM scattering form factor $F_{DM}^2(q, v)$. This information, and the energy transfer E, are provided to `McalI` via another dictionary, `modelDMSM`, so that an `McalI` class instance is defined via:

```
modelDMSM = dict(mX=..., fdm=(a,b), DeltaE=..., mSM=...)
mI = vsdm.McalI(basisV, basisQ, modelDMSM, mI_shape=(...),
               do_mcalI=bool, center_Z2=bool, use_gvar=bool)
```

The DM–SM scattering model depends on four parameters:

- `mX`: the DM particle mass, e.g. `1e9*eV` or `15*MeV` (using `units.py`)

- `fdm`: can be a tuple `(a, b)`, specifying the form factor parameters a, b for $F_{DM}^2 \propto q^a v^b$, following (1.38). Or, can be an integer or float, `fdm = n`, in which case $a = -2n$ and $b = 0$.

- `DeltaE`: the energy transfer from DM to the SM system, provided in `eV`, `keV`, etc.

- `mSM`: this is the mass of the SM target particle, e.g. m_e for DM–e^- scattering, so that the reduced mass of the DM–SM system is $\mu_{\chi SM} = m_{SM} m_\chi / (m_{SM} + m_\chi)$.

So, the first two arguments of `McalI` define the basis for the scattering matrix, and the third argument provides the physical parameters for the scattering process.

The values of $\mathcal{I}^{(\ell)}_{n_v, n_q}$ are saved as a 3d array,

$$\texttt{McalI.mcalI[l,nv,nq]} = \mathcal{I}^{(\ell)}_{n_v, n_q}(\Delta E, m_\chi, F^2_{\text{DM}}), \tag{4.5}$$

with the same normalization as (2.39). The fourth argument in the `McalI` initialization, the tuple `mI_shape`, sets the size of this array, so that it includes all values $\ell = 0, 1, \ldots, \ell_{\max}$, where $\ell_{\max} = \texttt{mI\_shape[0]} - 1$; likewise for the radial indices n_v and n_q.

When $\ell_{\max}$ and $n^{(v,q)}_{\max}$ are large, `McalI` can be one of the most time-consuming parts of the calculation, and so the class is structured to allow for parallel computation of its components. Initializing `McalI` with `do_mcalI = True` will instruct it to calculate all elements of $\mathcal{I}^{(\ell)}_{nn'}$ in series; with `do_mcalI = False`, the `McalI.mcalI` array is initialized with each value equal to zero. Then, different sets of (ℓ, n_v, n_q) indices can be assigned to multiple processors, and evaluated using the `McalI.updateIlvq` class method.

As long as both `basisV` and `basisQ` are wavelet or tophat basis functions, $\mathcal{I}^{(\ell)}_{n_v, n_q}$ can be integrated analytically. Each coefficient can be evaluated by running:

```
mI.updateIlvq((l,nv,nq), analytic=bool, integ_params={'verbose':bool})
```

Running `McalI` with the default values `analytic=True` and `integ_params=None` will replace `integ_params` with the dictionary `dict(verbose=False)`. Setting `verbose=True` causes `McalI` to print the intermediate integration results even for the analytic calculation.

For non-piecewise-constant basis functions, `McalI` must integrate each coefficient numerically (`analytic=False`). Tn this case, `integ_params` holds the parameters needed for the numerical integrator, with the same options as `Fnlm`. Note that compared to the analytic calculation, numerical integration for $\mathcal{I}$ will be extremely slow.

Once $\mathcal{I}^{(\ell)}$ has been calculated, it can be saved to an HDF5 file for future use, using the class method

```
mI.writeMcalI(hdf5file, modelName, alt_type=None)
```

Here `hdf5file` can be the same file that saves the `Fnlm` results for the g_χ and f^2_S models. By default, `McalI` arrays are stored under the type name `mcalI`, but the type name can be edited by passing a string to the `alt_type` argument. In either case, the dataset is saved under a subgroup name given by the `modelName` argument. The details of the `basisV` and `basisQ` basis functions, and `modelDMSM`, are saved to the `.attrs` dictionary. When running `McalI` with `use_gvar = True`, the central values and error bars are saved to two separate datasets, with names ending with "_mean" and "_sdev," respectively.

To read in the values of $\mathcal{I}^{(\ell)}_{nn'}$ from an HDF5 file, simply run:

```
mI.importMcalI(hdf5file, modelName, d_pair=['Ilvq_mean'])
```

providing an additional `alt_type` keyword argument if needed. When using the recommended wavelet or tophat basis functions, $\mathcal{I}$ is found analytically, so there is no `Ilvq_sdev` dataset with error bars. However, if $\mathcal{I}$ is found using numerical integration, then the error bars can be imported by setting:

```
d_pair=['Ilvq_mean', 'Ilvq_sdev']
```

when running `importMcalI`.

4.3.2 The Reduced Partial Rate Matrix $\mathcal{K}$

Once $\langle g_\chi | n\ell m \rangle \to$ gV, $\langle f_S^2 | n'\ell m' \rangle \to$ fsQ, and $\mathcal{I}_{nn'}^{(\ell)} \to$ mI are all known, the partial rate matrix $\mathcal{K}$ can be calculated directly using the RateCalc class:

```
mK = vsdm.RateCalc(gV, fsQ, mI, ellMax=None, lmod=1)
```

RateCalc automatically finds $\mathcal{K}$ from gV, fsQ and mI upon initialization. By default, the cutoff ℓ_{max} is taken to be the smallest value from the three physics inputs gV, fsQ, mI, but RateCalc can be initialized with a different value via the optional ellMax argument.

For convenience, the parent class of RateCalc, McalK, keeps track of two versions of the partial rate matrix: the "dimensionless" $\mathcal{K}$ defined in (2.46),

$$\text{mK.vecK[indexLMvMq]} = \mathcal{K}_{m_v, m_q}^{(\ell)}, \tag{4.6}$$

which has the same units as the momentum space form factor f_S^2; and the original partial rate matrix $K_{mm'}^{(\ell)}$ or $dK_{mm'}^{(\ell)}/dE$, defined in (2.3):

$$\text{mK.PartialRate[indexLMvMq]} = \frac{v_{max}^2}{q_{max}} \mathcal{K}_{m_v, m_q}^{(\ell)}. \tag{4.7}$$

Recall from (2.3) that PartialRate $= K$ is basis-independent, but not dimensionless, while vecK $= \mathcal{K}$ depends on the value of v_{max} and q_{max}. For discrete final states, $\left| f_S^2 \right\rangle$ is dimensionless, and so is vecK; for continuum final states, $\left| f_S^2 \right\rangle$ and vecK have units of inverse energy.

Both objects are stored as 1d arrays, indexed according to the function Gindex(l, mv, mq, lmod=1) from wigner.py: i.e.

$$\text{indexLMvMq} = \text{vsdm.Gindex}(\ell, m_v, m_q). \tag{4.8}$$

The lmod option in Gindex is used to skip all values of ℓ that are not divisible by lmod. The default lmod = 1 includes all $\ell = 0, 1, 2, \ldots$, but for systems where either f_S^2 or g_χ is center-symmetric, setting lmod = 2 will skip all odd values of ℓ. With lmod = 1, the entries of $\mathcal{K}_{m_v, m_q}^{(\ell)}$ are taken in the order:

$$(\ell, m_v, m_q) = (0, 0, 0), (1, -1, -1), (1, -1, 0), \ldots, (1, 1, 1), (2, -2, -2), \ldots, \tag{4.9}$$

while with lmod = 2 the odd ℓ (but not odd m) entries are skipped:

$$(\ell, m_v, m_q) = (0, 0, 0), (2, -2, -2), (2, -2, -1), \ldots, (2, 2, 2), (4, -4, -4), \ldots. \tag{4.10}$$

Saving K or $\mathcal{K}$ to CSV/HDF5: The partial rate matrices contain all of the information needed to evaluate dR/dE or R for arbitrary detector orientations. This data is also quite concise: the number of coefficients, len(PartialRate), simply scales as $\frac{4}{3}\ell_{max}^3$/lmod, which is typically orders of magnitude smaller than $\mathcal{I}_{nn'}^{(\ell)}$.

Because each $\mathcal{K}$ depends on the values of E and m_χ, as well as the versions of $g_\chi(\vec{v})$, $F_{DM}^2(q, v)$ and $f_S^2(\vec{q}, E)$ under consideration, we leave it to the reader to save and organize their partial rate matrices according to their own preferences.

When reading in $\mathcal{K}$ or K from tabulated vecK or PartialRate data, it is simplest to use McalK, the parent class of RateCalc. For example:

```
mK = vsdm.McalK(ellMax, lmod=lmod)
mK.PartialRate = ... # import from CSV/HDF5 data with matching lmod
# alternatively, if vecK was tabulated instead:
mK.vecK = ... # import from CSV/HDF5
mK.PartialRate = VMAX**2 / QMAX * mK.vecK
```

The first line initializes an `McalK` object with empty `vecK` and `PartialRate` vectors, which are then filled in by hand in the subsequent lines.

Unlike `PartialRate`, $\texttt{vecK} = \mathcal{K}$ depends on the values of $v_{\max}$ and $q_{\max}$, which must be saved independently when storing `vecK` data. On the other hand, `PartialRate` has dimensions of

$$[\texttt{PartialRate}] \propto \frac{\text{velocity}}{\text{mass}} \cdot [f_S^2] = \frac{\text{velocity}^3}{\text{energy}} \cdot \begin{cases} 1 & \text{discrete} \\ \text{energy}^{-1} & \text{continuous} \end{cases}, \qquad (4.11)$$

so in this case the user must ensure consistency in their unit conventions. By default, `vsdm` represents velocities in units of c, and energies in units of electronvolts. These defaults can only be changed by editing `VUNIT_c` and `EUNIT_eV` in `units.py`.

4.3.3 The Wigner G Matrix

Converting K into a scattering rate requires one final ingredient: the Wigner G matrix (2.5), which is a function of detector orientation $\mathcal{R} \in SO(3)$. We represent rotation operators using the standard quaternion convention, (1.16), and calculate $G_{mm'}^{(\ell)}$ from the Wigner $D^{(\ell)}$ matrix for complex spherical harmonics, following [2].

The class `WignerG` saves the values of each $G_{mm'}^{(\ell)}(\mathcal{R})$ as a 1d array, with the same indexing as `vecK` and `PartialRate`. If the list of rotation operators is known ahead of time, then `WignerG` can be initialized to evaluate G for this list:

```
wG = vsdm.WignerG(ellMax, rotations=[Q1, Q2, Q3, ...], lmod=lmod)
```

for quaternions Q1, Q2, etc., and for some values of `ellMax` and `lmod`.

`WignerG` saves the values of $G(\mathcal{R})$ to a 2d array, `G_array`. It can also be used to evaluate additional values of $G(\mathcal{R})$ for $\mathcal{R}$ not included in the initial `rotations` list, via the class method `G()`:

```
newG_R = wG.G(R, save=bool)
```

If `save==True`, then the newest G vector is also appended to the `G_array` list, and the corresponding value of R is appended to the list `wG.rotations` of saved rotation quaternions.

4.3.4 Calculating Rates

The partial rate class `McalK` includes several methods for evaluating the scattering rate for lists of $\mathcal{R}$ rotations. The simplest of these options is the function `mu_R`, which calculates the dot product

$$\texttt{mu\_R}(\mathcal{R}) = \sum_{\ell,m,m'} G_{mm'}^{(\ell)}(\mathcal{R}) \cdot \mathcal{K}_{mm'}^{(\ell)} \qquad (4.12)$$

for every $\mathcal{R}$ in the `rotations` list. This is quite fast: it is as simple as multiplying the 2d array `wG.G_array` by the 1d vector `mK.vecK`. The output, a list of $\mu(\mathcal{R})$, is proportional to the rate R or dR/dE without the exposure factor k_0 from (2.51).

To include the exposure factor, one can multiply `mu_R` by the function `vsdm.g_k0`. For discrete final states, this returns the total number of expected events, based on the target exposure (in kg yr), cross section $\bar{\sigma}$, and local DM density ρ_χ:

$$N_{\text{events}} = \text{g\_k0} \cdot \text{mu\_R}, \tag{4.13}$$

$$\text{g\_k0} = k_0 = \frac{M_T}{m_{\text{cell}}^{(\text{mol})}} N_A T_{\text{exp}} \bar{\sigma}_0 \rho_\chi \frac{v_{\text{max}}^2}{q_{\text{max}}} \tag{4.14}$$

Here $N_A M_T / m_{\text{cell}}^{(\text{mol})}$ is the number of unit cells in the detector, written in terms of the target mass M_T and the molar mass of the unit cell, $m_{\text{cell}}^{(\text{mol})}$, as in (1.27). For continuum final states, the product `g_k0 * mu_R` is the number of expected events per unit energy, i.e.

$$T \cdot \frac{dR}{dE}\bigg|_E = \text{g\_k0} \cdot \text{mu\_R}(E), \tag{4.15}$$

for exposure time T.

For either case, `McalK` provides a convenient class method, `Nevents`, that evaluates the total expected number of events for each rotation in `wG`:

$$\text{mK.Nevents(wG)} = \begin{cases} T \cdot R(\mathcal{R}) & \text{discrete} \\ T \cdot \frac{dR}{dE}\big|_{E,\mathcal{R}} & \text{continuous} \end{cases} \tag{4.16}$$

The exposure factor is determined by the values supplied in the optional arguments of `Nevents`: `exp_kgyr`, $M_T T$ in units of kg yr; `mCell_g`, the molar cell mass in grams; `sigma0_cm2`, the cross section $\bar{\sigma}_0 d_{\alpha\beta}$ in cm^2; and `rhoX_GeVcm3`, the local DM density in GeV/cm^3.

5 Performance

In this section we provide some specific results for the evaluation times of both the Python and Julia versions of VSDM to compare to the full multi-dimensional numeric integration. The total evaluation time for a set of velocity distributions, material form factors, dark matter models (masses and form factors), and rotations is given by

$$T_{\text{tot}} = N_{g_\chi} N_v T_{\text{proj}} + N_{f_S} N_q T_{\text{proj}} + N_{\mathcal{I}} T_{\mathcal{I}} + N_{g_\chi} N_{f_S} N_{\text{DM}} \left(N_{\mathcal{K}} T_{\mathcal{K}} + N_{\mathcal{R}} T_{\text{tr}} \right), \tag{5.1}$$

where

- N_{g_χ} is the number of different velocity distributions

- N_{f_S} is the number of different material form factors

- N_v and N_q are the total number of coefficients computed for each velocity distribution and material form factor respectively

- T_{proj} is the average time to project a single coefficient

- $T_{\mathcal{I}}$ is the average time per coefficient to compute the $\mathcal{I}_{n,n'}^{(\ell)}$ matrix

- $T_{\mathcal{K}}$ is the average time per coefficient to compute a reduced partial rate matrix $\mathcal{K}_{m,m'}^\ell$ matrix from a pre-computed $\mathcal{I}$

- N_{DM} is the number of dark matter models

- $N_{\mathcal{R}}$ is the number of rotational configurations

- T_{tr} is the average time to compute the final dot product between the Wigner G matrix and the partial rate matrix K.

N_v (N_q) is determined by the number of radial coefficients $N_n^{(v)}$ ($N_n^{(q)}$), $\ell_{\max}$, and the symmetries of the given function to be projected. For a function with *no* symmetries the total number of coefficients is $N_x = N_n^{(x)}(\ell_{\max}+1)^2$. If `z_even = true` or `phi_even = true` this number is approximately halved and if `phi_symmetric = true` divide the total number of coefficients by a factor of $\ell_{\max} + 1$. $N_{\mathcal{I}}$ is equal to the number of coefficients in $\mathcal{I}$ times the number of $\mathcal{I}$ matrices that need to be computed, $N_{\mathcal{I}} = N_{\mathrm{DM}}N_{\Delta E}N_n^{(v)}N_n^{(q)}\ell_{\max}$ where $N_{\Delta E}$ is the number of different excitation energies. $N_{\mathcal{K}}$ depends on the total number of coefficients over both the velocity distribution and material form factor, $N_{\mathcal{K}} = N_n^{(v)}N_n^{(q)}\ell_{\max}^3$.

Language	T_{proj}	$T_{\mathcal{I}}$	$T_{\mathcal{K}}$	$T_{\mathrm{tr}}/\ell_{\max}^3$
Python	0.74	1.4×10^{-4}	1.1×10^{-10}	9.8×10^{-10}
Julia	7.0×10^{-3}	9.1×10^{-7}	3.4×10^{-11}	1.6×10^{-10}

Table 1: All listed times are in seconds. T_{proj} was computed using the example material form factor from [2] with $n_{\max} = 2^8 - 1$, $\ell_{\max} = 4$. $T_{\mathcal{I}}$ and $T_{\mathcal{K}}$ were computed with same $n_{\max}$, $\ell_{\max} = 10$, for $m_\chi = 100$ MeV, $\Delta E = 4.03$ eV, and $F_{\mathrm{DM}} = 1$. T_{tr} is also computed for $\ell_{\max} = 10$ and only depends on $\ell_{\max}$.

Table 1 shows a concrete example of evaluation times on a Ryzen R5 3600 processor (with hyperthreading disabled in the Julia version). For reference, it takes ~ 30 sec to numerically calculate the full integral rate for a single parameter point using the `MCIntegration.jl` package in Julia, with options `solver=:vegas`, `niter=10`, and `neval=` 10^6 (everything else default). In Python, using the package `vegas` it takes ~ 100 sec to perform the same integral (with `nitn=` 10, `neval=` 10^6). Both packages end up with about $\mathcal{O}(0.1\%)$ error with these options.

If we take $N_{g_\chi} = N_{f_s} = 10$, $N_{\Delta E} = 10$, $N_n^{(v)} = N_n^{(q)} = 2^8$, $\ell_{\max} = 10$, $N_{\mathrm{DM}} = 60$, and $N_{\mathcal{R}} = 10^4$ (a reasonable number of parameter points) and assume that neither the velocity distribution nor the material form factor have any symmetries, the *first time* we calculate everything the total evaluation times would be:

$$\text{Monte Carlo (Julia):} \sim 1.8 \times 10^9 \text{ sec} \simeq 57 \text{ yr}$$

$$\text{Python:} \sim 5.1 \times 10^5 \text{ sec} \simeq 142 \text{ hr}$$

$$\text{Julia:} \sim 4.7 \times 10^3 \text{ sec} \simeq 79 \text{ min.}$$

On subsequent analyses, however, we can make use of the fact that we have already computed the projection coefficients, and load them from a file. For this number of coefficients and models, the total time to load projected coefficients and assemble them into a matrix is about 4 sec (92 sec) for Julia (Python) (coefficients stored on a WD SN770 drive). This decreases the evaluation times for VSDM,

$$\text{Python:} \sim 5.5 \times 10^4 \text{ sec} \simeq 15.3 \text{ hr}$$

$$\text{Julia:} \sim 3.8 \times 10^2 \text{ sec} \simeq 6.3 \text{ min.}$$

The majority of the computation time here is taken up by $\mathcal{I}$. There are two options for saving this: 1. you can directly save $\mathcal{I}$, although it is a large object ($N_n^{(v)} \times N_n^{(q)} \times (\ell_{\max} + 1)$), or 2.

you can save $\mathcal{K}$, which is in general much smaller. It is useful to save $\mathcal{I}$ when you have many different velocity distributions and/or material form factors that all use the same basis (and ΔE), but if you do not anticipate needing to add more of these, saving $\mathcal{K}$ will save a lot of space. If we have saved all of the $\mathcal{K}$ matrices to a file the $R(\mathcal{R})$ evaluation time is even faster,

$$\text{Python: } \sim 59 \text{ sec}$$
$$\text{Julia: } \sim 9.6 \text{ sec},$$

where in both cases it would take about 20 sec to load all of the $\mathcal{K}$ matrices. Calculating the scattering rate from a tabulated partial rate matrix is faster by factors of

$$\text{Python: } \sim 3.0 \times 10^7$$
$$\text{Julia: } \sim 1.9 \times 10^8,$$

compared to brute-force numerical integration.

6 Conclusion

VSDM is an invaluable tool for researchers in dark matter direct detection. By projecting the DM velocity distributions and the material form factors onto their respective bases of wavelet-harmonic functions, the three-dimensional input functions from astrophysics, condensed matter and/or physical chemistry are represented concisely in a format that is easily shared between researchers. VSDM easily assembles the partial rate matrix for DM–SM scattering from this data. For an anisotropic detector, VSDM can evaluate the orientation-dependent scattering rate in fractions of a microsecond for each point in parameter space.

Even an isotropic analysis can benefit from using VSDM, thanks to the factorization inherent in the VSDM rate calculation. For large sets of input distributions, i.e. $N_{g_\chi} \gg 1$ velocity distributions and $N_{f_S} \gg 1$ response functions, the total number of rate evaluations can become large enough that even low-dimensional numerical integration becomes inconveniently slow. With VSDM, one only has to calculate the $\ell = m = 0$ modes of the velocity and momentum distributions, $\langle g_\chi | n00 \rangle$ and $\langle f_S^2 | n'00 \rangle$, and the $\ell_{\max} = 0$ kinematic scattering matrix $\mathcal{I}_{nn'}^{(0)}$. The angular average rate, proportional to $\mathcal{K}_{0,0}^{(0)}$, is then efficiently obtained from (2.46) for each combination of models.

Acknowledgments

The work of B.L. was supported by the U.S. Department of Energy under Grant Number DE-SC0011640. The work of A.R. was supported in part by NSF CAREER grant PHY-1944826. We thank Pankaj Munbodh for providing helpful comments on the manuscript, and Pierce Giffin and P.K. for their comments on an earlier version of the Python implementation vsdm.

A Associated Legendre Polynomials and Spherical Harmonics

The associated Legendre polynomials $P_\ell^m(x)$ appear frequently in physics. They are solutions to the angular components of the 3d Laplace equation, i.e.:

$$\frac{d}{dx}\left((1-x^2)\frac{d}{dx}P_\ell^m(x)\right) + \left(\ell(\ell+1) - \frac{m^2}{1-x^2}\right)P_\ell^m(x) = 0, \qquad \text{(A.1)}$$

and they are a crucial component in defining the spherical harmonics.

Standard computational libraries (e.g. SciPy for Python) include routines for evaluating spherical harmonics and the associated Legendre polynomials P_ℓ^m, but these tend to fail for $\ell, m > 10^2$. In this Appendix, we outline the recursive method that VSDM uses to evaluate $Y_{\ell m}$. It retains its accuracy up to $\ell, m \leq 1800$, which is more than sufficient for our needs.

A.1 Definitions

We define the associated Legendre polynomial of order ℓ and degree m via its Rodriguez formula,

$$P_\ell^m(x) \equiv \frac{(-1)^{\ell+m}}{2^\ell \ell!} (1-x^2)^{m/2} \frac{d^{\ell+m}}{dx^{\ell+m}} (1-x^2)^\ell, \tag{A.2}$$

including the Condon–Shortley phase $(-1)^m$. At $m = 0$, $P_\ell^0 = P_\ell$ is the regular (i.e. not associated) Legendre polynomial,

$$P_\ell(x) \equiv \frac{(-1)^\ell}{2^\ell \ell!} \frac{d^\ell}{dx^\ell} (1-x^2)^\ell. \tag{A.3}$$

(A.2) is valid for negative m, but these negative-order polynomials can also be derived from positive-order $P_\ell^{|m|}$ as follows:

$$P_\ell^{-|m|} = (-1)^m \frac{(\ell - |m|)!}{(\ell + |m|)!} P_\ell^{|m|}(x). \tag{A.4}$$

We define the complex spherical harmonics (Y_ℓ^m) as:

$$Y_\ell^m(\theta, \phi) \equiv \sqrt{\frac{2\ell+1}{4\pi} \frac{(\ell-m)!}{(\ell+m)!}} P_\ell^m(\cos\theta) e^{im\phi}, \tag{A.5}$$

while for the real spherical harmonics $(Y_{\ell m})$, we take:

$$Y_{\ell m}(\theta, \phi) \equiv \begin{cases} \sqrt{2}(-1)^m \sqrt{\frac{2\ell+1}{4\pi} \frac{(\ell-|m|)!}{(\ell+|m|)!}} P_\ell^{|m|}(\cos\theta)\sin(|m|\varphi) & \text{for } m < 0, \\ \sqrt{\frac{2\ell+1}{4\pi}} P_\ell(\cos\theta) & \text{for } m = 0, \\ \sqrt{2}(-1)^m \sqrt{\frac{2\ell+1}{4\pi} \frac{(\ell-m)!}{(\ell+m)!}} P_\ell^m(\cos\theta)\cos(m\varphi) & \text{for } m > 0, \end{cases} \tag{A.6}$$

so that the negative-m real harmonics are odd under $\varphi \to -\varphi$, and the positive-m harmonics are even.

A.2 Difficulty At Large ℓ

The associated Legendre polynomial P_ℓ^m is a polynomial of degree ℓ, with coefficients that can be determined analytically. For example,

$$P_\ell^m(x) = (-1)^m 2^\ell (1-x^2)^{m/2} \sum_{k=m}^{\ell} \frac{k!}{(k-m)!} x^{k-m} \binom{\ell}{k} \binom{\frac{1}{2}(\ell+k-1)}{\ell}, \tag{A.7}$$

with the "n choose k" symbols defined for non-integer n as

$$\begin{pmatrix} n \\ k \end{pmatrix} \equiv \frac{\Gamma(n+1)}{\Gamma(k+1)\,\Gamma(n-k+1)}. \tag{A.8}$$

At $m = \ell$ this yields a particularly simple expression,

$$P_\ell^\ell(x) = (-1)^\ell (2\ell - 1)!!\,(1 - x^2)^{\ell/2}, \tag{A.9}$$

where !! is the double factorial, $n!! = (n)(n-2)(n-4)\ldots(3)(1)$ for odd n.

Attempting to evaluate P_ℓ^m directly from (A.7) quickly leads to underflow and overflow errors when ℓ becomes large. For example, Stirling's approximation gives the asymptotic form for the double factorial as

$$(2\ell - 1)!! \approx \sqrt{\frac{4\ell - 4}{2\ell - 1}} \left[\frac{(2\ell - 1)^2}{2e(\ell - 1)} \right]^\ell, \tag{A.10}$$

which exceeds 10^{100} for $\ell > 60$. This large factor must cancel against similarly exponentially small factors from $\sqrt{(\ell - m)!/(\ell + m)!}$, x^{k-m}, and $(1 - x^2)^{m/2}$ to yield $\mathcal{O}(1)$ values for $Y_\ell^m(\theta, \phi)$, and for $\ell \sim 10^2$ this cannot be done accurately from the polynomial form of (A.7).

A great many recursion relations for P_ℓ^m and $P_{\ell\pm1}^{m\pm1}$ can be derived from (A.1). Two relatively simple recursion relations keep either ℓ or m fixed: we refer to these as fixed-degree or fixed-order recursion relations, respectively. SciPy (v1.15.0) appears to use the fixed-degree recursion relations: to find $P_\ell^m(x)$, it first evaluates $P_\ell^\ell(x)$ via (A.9), then it employs the appropriate decreasing-order recursion relation to evaluate $P_\ell^{\ell-1}$, $P_\ell^{\ell-2}$, ..., P_ℓ^m. As a result, SciPy is only able to calculate Y_ℓ^m for ℓ and m up to about $\ell \approx m \approx 85$. This may be insufficient for our purposes.

Instead, we employ a fixed-order recursion relation, beginning with P_m^m and evaluating P_{m+1}^m, P_{m+2}^m, ..., P_ℓ^m by increasing the degree. This is much more numerically stable, permitting the calculation of Y_ℓ^m up to $\ell \sim 1800$ with an absolute accuracy of 10^{-10} or better, using standard 64-bit floating precision.

A.3 Recursion Relations

The associated Legendre polynomial satisfies the modified Bonnet recurrence formula:

$$(\ell - m)P_\ell^m = x(2\ell - 1)P_{\ell-1}^m - (\ell + m - 1)P_{\ell-2}^m, \tag{A.11}$$

suppressing the x dependence of $P_\ell^m(x)$. Given P_m^m and P_{m+1}^m, where

$$P_{m+1}^m = x(2m + 1)P_m^m, \tag{A.12}$$

(A.11) provides the means to evaluate P_ℓ^m for $\ell \geq m + 2$. This is the *Numerical Recipes* [64] recommendation for evaluating P_ℓ^m.

For VSDM, the reason to calculate P_ℓ^m is to subsequently calculate $Y_{\ell m}$ or Y_ℓ^m, so we define the "reduced" (or partially normalized) associated Legendre polynomials $\mathcal{P}_\ell^m$,

$$\mathcal{P}_\ell^m(x) \equiv (-1)^m \sqrt{\frac{(\ell - m)!}{(\ell + m)!}} P_\ell^m(x), \tag{A.13}$$

to absorb the exponentially small prefactor. Including the factor of $(-1)^m$ removes the Condon–Shortley phase, so as to simplify the evaluation of the real spherical harmonics:

$$Y_{\ell m}(\theta, \phi) = \sqrt{\frac{2\ell + 1}{4\pi}} \mathcal{P}_\ell^{|m|}(\cos\theta) \times \begin{cases} \sqrt{2}\sin(|m|\phi) & m < 0, \\ 1 & m = 0, \\ \sqrt{2}\cos(m\phi) & m > 0. \end{cases} \tag{A.14}$$

Adapting (A.11) for $\mathcal{P}_\ell^m$, we find:

$$\sqrt{\ell^2 - m^2}\,\mathcal{P}_\ell^m = (2\ell - 1)x\mathcal{P}_{\ell-1}^m - \sqrt{(\ell-1)^2 - m^2}\,\mathcal{P}_{\ell-2}^m, \tag{A.15}$$

with the initial values calculated via

$$\mathcal{P}_m^m = (1 - x^2)^{m/2}\prod_{j=0}^{m-1}\sqrt{1 - \frac{1/2}{1+j}}, \qquad\qquad \mathcal{P}_{m+1}^m = \sqrt{2m+1}\,x\,\mathcal{P}_m^m. \tag{A.16}$$

The form of the product above is the reason why we use $\mathcal{P}_\ell^m$ rather than P_ℓ^m: each term in this product is approximately 1, even when m is absurdly large.

The analogous fixed-degree recursion (apparently used by SciPy) is less stable. While $\mathcal{P}_\ell^\ell$ can be calculated precisely using (A.16), the decreasing-order recursion relation,

$$\mathcal{P}_\ell^m(x) = \frac{(m+1)\alpha_x\mathcal{P}_\ell^{m+1}(x) - \sqrt{(\ell-m-1)(\ell+m+2)}\,\mathcal{P}_\ell^{m+2}(x)}{\sqrt{(\ell-m)(\ell+m+1)}}, \tag{A.17}$$

depends on a coefficient

$$\alpha_x \equiv \frac{x}{\sqrt{1-x^2}} \tag{A.18}$$

that becomes much larger than $\mathcal{O}(1)$ as $x \to \pm 1$. Consequently, this routine for calculating $\mathcal{P}_\ell^m$ and Y_ℓ^m becomes unreliable at significantly smaller values of $\ell_{\max}$, compared to the fixed-order recursion.

A.4 Accuracy At Large ℓ

At large m, the term $(1 - x^2)^{m/2}$ in (A.16) becomes exponentially small everywhere except about $x \approx 0$. With the fixed-order recursion relation, this is one limiting factor in our ability to accurately resolve $Y_{\ell m}(\theta, \phi)$: if $(1 - x^2)^{m/2} = |\sin^m\theta|$ is too small, then our algorithm sets $Y_\ell^m \to 0$ for those values of θ. This is related to a physical effect, however: the large m spherical harmonics vanish as $P_\ell^m(\cos\theta) \propto \sin^m\theta$ near the $\theta \approx 0$ poles. The $m \approx \ell$ harmonics have support only near the equator, $\theta \approx \pi/2$, where $x = \cos\theta \approx 0$. Our fixed-order recursive method is stable to extremely large values of ℓ and m in this regime: for example, taking $\ell = 3 \times 10^9$ and $m = 1.5 \times 10^9$, VSDM finds

$$Y_{3\,000\,000\,000}^{1\,500\,000\,000}(\theta = 1.570, \phi = 0) = -0.335\,001\,108, \tag{A.19}$$

which matches the result from Mathematica 12 to better than one part in 10^9.

Away from the equator, however, the $m \ll \ell$ harmonics become inaccurate at $\ell > 1800$. For example,

$$Y_{2000}^{500}(2.91, 0) \approx 3.90715 \cdot 10^{-6}, \tag{A.20}$$

but at $\theta = 2.92$ the result drops abruptly (and unphysically) to zero. So, in applications that require a precision of better than 10^{-6}, our $\ell \gtrsim 2000$ spherical harmonics are not guaranteed to be accurate.

At $\ell = 1800$, discontinuities of this type still occur, but only once Y_ℓ^m itself is quite small. For example, we find

$$Y_{1800}^{700}(2.78, 0) \simeq 8.039\,439 \cdot 10^{-10}, \qquad\qquad Y_{1800}^{700}(2.79, 0) \to 0, \tag{A.21}$$

losing precision above above $\theta > 2.785$ (where $Y_\ell^m(\theta, 0) \lesssim 1.1 \cdot 10^{-11}$). This value of $m \sim 700$ roughly maximizes the error for the $\ell = 1800$ harmonics. Because this error remains smaller than 10^{-10} at all values of θ, we say that our spherical harmonic evaluation remains precise up to $\ell_{\max} = 1800$.

For $\ell \leq 1800$, we find smooth results for the full domain of $|x| \leq 1$, for all $|m| \leq \ell$, even in the polar regions where Y_ℓ^m is exponentially small. Taking $\ell = 10^3$ for a final demonstration,

$$Y_{1000}^{100}(3.1, 0) = 9.603\,018\,299\,15 \times 10^{-28}, \tag{A.22}$$

$$Y_{1000}^{500}(2.8, 0) = 8.399\,092\,757\,04 \times 10^{-52} \tag{A.23}$$

$$Y_{1000}^{997}(1.9, 0) = 2.583\,063\,136\,80 \times 10^{-21}. \tag{A.24}$$

These values of ℓ are an order of magnitude larger than the largest values of $\ell_{\max}$ that we expect to need for VSDM.

In the Python package `vsdm`, the spherical harmonics and partially normalized associated Legendre polynomials are given by:

$$\texttt{plm\_norm}(\ell, m, x) = \mathcal{P}_\ell^m(x), \tag{A.25}$$

$$\texttt{ylm\_cx}(\ell, m, \theta, \phi) = Y_\ell^m(\theta, \phi), \tag{A.26}$$

$$\texttt{ylm\_real}(\ell, m, \theta, \phi) = Y_{\ell m}(\theta, \phi), \tag{A.27}$$

with θ and ϕ in radians.

B Spherical Wavelet and Tophat Basis Functions

The d-dimensional spherical Haar wavelets and spherical tophat functions are both normalized according to

$$\int_0^1 x^{d-1} dx \, r_n(x) r_{n'}(x) = \delta_{nn'} \tag{B.1}$$

for $x \in [0, 1]$. The $d = 3$ wavelet functions $r_n(x) = h_n(x)$ are already provided in Section 2.2. For arbitrary values of d, the wavelet base of support remains the same as in (2.26), but the values of A_n and B_n change:

$$h_n(x) = \begin{cases} +A_n & x_1(n) \leq x < x_2(n), \\ -B_n & x_2(n) < x \leq x_3(n), \\ 0 & \text{otherwise}, \end{cases} \tag{B.2}$$

where

$$x_1(n) = 2^{-\lambda}\mu, \qquad x_2(n) = 2^{-\lambda}(\mu + \tfrac{1}{2}), \qquad x_3(n) = 2^{-\lambda}(\mu + 1), \tag{B.3}$$

and

$$A_n = \sqrt{\frac{d}{x_3^d - x_1^d} \frac{x_3^d - x_2^d}{x_2^d - x_1^d}} \qquad\qquad B_n = \sqrt{\frac{d}{x_3^d - x_1^d} \frac{x_2^d - x_1^d}{x_3^d - x_2^d}}. \tag{B.4}$$

For the $n = 0$ wavelet,

$$A_0 \equiv \sqrt{d}, \qquad\qquad B_0 \equiv 0. \tag{B.5}$$

The $d = 1$ case returns the standard Haar wavelets. In the Python implementation, vsdm, the value of d can be altered from its default ($d = 3$) by adding a dim entry to the basis dictionary: e.g.,

```
basisV = dict(uMax=820*km_s, type='wavelet', dim=1)
```

Tophat Functions: Some of the intermediate steps of the calculation are faster when evaluated using piecewise-constant tophat basis functions, and then applying a discrete wavelet transformation to obtain the wavelet coefficients. (See Section 2.7.) Unlike the wavelets, the tophat basis functions are allowed to have varying widths. A list of boundary points $x_0, x_1, x_2, x_3, \ldots, x_N, x_{N+1}$ defines a basis with N many tophat functions. The nth basis function ($n = 0, 1, \ldots, N - 1$) is defined as

$$t_n(x) = \begin{cases} +T_n & x_n \leq x < x_{n+1}, \\ 0 & \text{otherwise}, \end{cases} \qquad T_n = \sqrt{\frac{d}{x_{n+1}^d - x_n^d}}. \qquad \text{(B.6)}$$

The implementation of Tophat in Julia is discussed in Section 3.1. In Python, the list of boundary points for a dimensionful parameter u is included as an item uiList in the basis dictionary:

```
basisV = dict(uiList=[j*km_s for j in range(0,830,20)], type='tophat')
```

The list uiList must be ordered, but it is not mandatory for the smallest value to be zero. The dimensionful quantity u is converted to dimensionless x via

$$x = \frac{u}{u_{\text{max}}}. \qquad \text{(B.7)}$$

C   Python Example Code

Section 4 provides multiple examples for each intermediate stage of the VSDM calculation, in more detail than is typically needed for a single analysis. In this section we provide a simpler summary, showing all the steps needed for a complete analysis. The result is simpler than it might appear in Section 4.

This example calculates $\langle g_\chi | n\ell m \rangle$ from an analytic function $g_\chi(\vec{v})$ defined by the user, while importing the $\langle f_s^2 | n\ell m \rangle$ coefficients from a CSV file in the vsdm format. Such a file could have been generated by running vsdm.Fnlm; or, it could have been made outside of vsdm, e.g. by the user's implementation of the discrete wavelet-harmonic transform of Section 2.7.

```python
from vsdm import *
import numpy as np
import quaternionic
# Step 1: project gX(v) onto wavelet-harmonic basis:
# 1a. define a gX(uSph) function:
def gX_v(vSph):
    v, theta, phi = vSph
    gX = ... # a function of v, theta, and phi
    return gX # in units of (inverse velocity)**3
gX_v.phi_symmetric = False # True, if gX is azimuthally symmetric
```

```
# 1b. define velocity basis functions
basisV = dict(uMax=820*km_s, type='wavelet')
# 1c. get <gX|nlm> coefficients
nlmlist = [(n,l,m) for n in range(2**9) for l in range(13)
          for m in range(-l, l+1)]
integ_params = dict(method='gquad', rtol_f=1e-5, atol_f=1e-8)
gX = EvaluateFnlm(basisV, gX_v, integ_params, nlmlist=nlmlist,
                  csvsave_name='gXnlm.csv', f_type='gX',
                  use_gvar=False)
gX._makeFarray() # populate the f_lm_n arrays for later
# Step 2: import <fs2|nlm> coefficients from a CSV file
# 2a. define momentum basis functions (must match CSV file!)
basisQ = dict(uMax=10*qBohr, type='wavelet')
# 2b. initialize an empty Fnlm object and import CSV coefficients:
fs2 = Fnlm(basisQ, f_type='fs2', use_gvar=False)
fs2.center_Z2 = True # add symmetry properties of fs2 as needed
fs2.importFnlm_csv('fs2_6eV.csv')
fs2.DeltaE = 6.0*eV # optional: keep track of excitation energy here
fs2._makeFarray() # populate the f_lm_n arrays for later
# Step 3: make an McalI matrix for specific DM model
modelDMSM = dict(mX=20*MeV, fdm=(-4,2), DeltaE=fs2.DeltaE, mSM=mElec)
mI = McalI(basisV, basisQ, modelDMSM, mI_shape=(13,512,fs2.nMax+1),
           do_mcalI=True, center_Z2=True)
# Step 4: make the McalK partial rate matrix
mK = RateCalc(gX, fs2, mI, lmod=2) # center-symmetric: lmod=2
# Step 5: calculate Wigner G matrix for a list of rotations
# rotating the detector about an axis 'nHat' by angle 'beta'
nHat = quaternionic.array([0, 3.2, -1.2, 2.7])
nHat = nHat/nHat.abs # nHat: a 3d unit vector
qR = quaternionic.array([1, 0, 0, 0])
rotations = [np.cos(beta*np.pi/360)*qR + np.sin(beta*np.pi/360)*nHat
            for beta in range(0, 360, 3)]
wG = WignerG(mK.ellMax, rotations=rotations, lmod=2)
# Step 6: calculate scattering rate for each rotation
mass_kg = 1.0 # 1 kg detector
time_yr = 1.0 / SECONDS_PER_YEAR # 1 second exposure time
N_events = mK.Nevents(wG, exp_kgyr=mass_kg*time_yr, mCell_g=125.,
                      sigma0_cm2=1e-37, rhoX_GeVcm3=0.4)
```

The final line produces a list, N_events, for the scattering rate (in events per second) for each orientation of the detector, in the same order as the rotations list. The hypothetical detector has a mass of $1.0\,\text{kg}$, and the molar mass for the f_S^2 unit cell is $m_{\text{cell}}^{\text{(mol)}} = 125\,\text{g}$. The Julia version of the ExposureFactor would have taken N_T, the number of unit cells, as an input; in this case, $N_T = N_A M_T / m_{\text{cell}}^{\text{(mol)}} = 8.0\,N_A$, where N_A is the Avogadro constant.

This example code works for either discrete or continuous final states. For discrete final states, fs2_6eV.csv provides the function $f_{g \to s}^2(\vec{q})$, and

$$f_S^2(\vec{q}, E) = \text{fs2}(\vec{q})\,\delta(E - \Delta E). \tag{C.1}$$

Here mK.Nevents is dimensionless, counting the total number of scattering events that produce this $6\,\text{eV}$ excited state.

Alternatively, `fs2_6eV.csv` could just as well provide the $\langle f_S^2(E)|n\ell m\rangle$ coefficients at fixed $E = 6\,\text{eV}$, in which case

$$\texttt{fs2}(\vec{q}) = f_S^2(\vec{q}, 6\,\text{eV}) \tag{C.2}$$

has units of inverse energy. Likewise, `mK.Nevents` counts the number of events expected within the 6 eV energy bin, and

$$\texttt{N\_events}(\mathcal{R}) = T_{\text{exp}} \frac{dR(\mathcal{R})}{dE}\bigg|_{E=6\,\text{eV}}. \tag{C.3}$$

References

[1] B. Lillard, *Vector Space Integration for Dark Matter Scattering* (2023), 2310.01480.

[2] B. Lillard, *Wavelet-Harmonic Integration Methods* (2023), 2310.01483.

[3] Y. Hochberg, Y. Kahn, M. Lisanti, C. G. Tully and K. M. Zurek, *Directional detection of dark matter with two-dimensional targets*, Phys. Lett. B **772**, 239 (2017), doi:10.1016/j.physletb.2017.06.051, 1606.08849.

[4] R. Budnik, O. Chesnovsky, O. Slone and T. Volansky, *Direct Detection of Light Dark Matter and Solar Neutrinos via Color Center Production in Crystals*, Phys. Lett. B **782**, 242 (2018), doi:10.1016/j.physletb.2018.04.063, 1705.03016.

[5] Y. Hochberg, Y. Kahn, M. Lisanti, K. M. Zurek, A. G. Grushin, R. Ilan, S. M. Griffin, Z.-F. Liu, S. F. Weber and J. B. Neaton, *Detection of sub-MeV Dark Matter with Three-Dimensional Dirac Materials*, Phys. Rev. D **97**(1), 015004 (2018), doi:10.1103/PhysRevD.97.015004, 1708.08929.

[6] S. Griffin, S. Knapen, T. Lin and K. M. Zurek, *Directional Detection of Light Dark Matter with Polar Materials*, Phys. Rev. D **98**(11), 115034 (2018), doi:10.1103/PhysRevD.98.115034, 1807.10291.

[7] A. Coskuner, A. Mitridate, A. Olivares and K. M. Zurek, *Directional Dark Matter Detection in Anisotropic Dirac Materials*, Phys. Rev. D **103**(1), 016006 (2021), doi:10.1103/PhysRevD.103.016006, 1909.09170.

[8] R. M. Geilhufe, F. Kahlhoefer and M. W. Winkler, *Dirac Materials for Sub-MeV Dark Matter Detection: New Targets and Improved Formalism*, Phys. Rev. D **101**(5), 055005 (2020), doi:10.1103/PhysRevD.101.055005, 1910.02091.

[9] S. M. Griffin, Y. Hochberg, K. Inzani, N. Kurinsky, T. Lin and T. Chin, *Silicon carbide detectors for sub-GeV dark matter*, Phys. Rev. D **103**(7), 075002 (2021), doi:10.1103/PhysRevD.103.075002, 2008.08560.

[10] C. Blanco, Y. Kahn, B. Lillard and S. D. McDermott, *Dark Matter Daily Modulation With Anisotropic Organic Crystals*, Phys. Rev. D **104**, 036011 (2021), doi:10.1103/PhysRevD.104.036011, 2103.08601.

[11] A. Coskuner, T. Trickle, Z. Zhang and K. M. Zurek, *Directional detectability of dark matter with single phonon excitations: Target comparison*, Phys. Rev. D **105**(1), 015010 (2022), doi:10.1103/PhysRevD.105.015010, 2102.09567.

[12] C. Blanco, I. Harris, Y. Kahn, B. Lillard and J. Pérez-Ríos, *Molecular Migdal effect*, Phys. Rev. D **106**(11), 115015 (2022), doi:10.1103/PhysRevD.106.115015, 2208.09002.

[13] C. Boyd, Y. Hochberg, Y. Kahn, E. D. Kramer, N. Kurinsky, B. V. Lehmann and T. C. Yu, *Directional detection of dark matter with anisotropic response functions* (2022), 2212.04505.

[14] R. Catena, T. Emken, M. Matas, N. A. Spaldin and E. Urdshals, *Direct searches for general dark matter-electron interactions with graphene detectors: Part I. Electronic structure calculations* (2023), 2303.15497.

[15] R. Catena, T. Emken, M. Matas, N. A. Spaldin and E. Urdshals, *Direct searches for general dark matter-electron interactions with graphene detectors: Part II. Sensitivity studies* (2023), 2303.15509.

[16] R. Agnese *et al.*, *First Dark Matter Constraints from a SuperCDMS Single-Charge Sensitive Detector*, Phys. Rev. Lett. **121**(5), 051301 (2018), doi:10.1103/PhysRevLett.121.051301, [Erratum: Phys.Rev.Lett. 122, 069901 (2019)], 1804.10697.

[17] A. Aguilar-Arevalo *et al.*, *Constraints on Light Dark Matter Particles Interacting with Electrons from DAMIC at SNOLAB*, Phys. Rev. Lett. **123**(18), 181802 (2019), doi:10.1103/PhysRevLett.123.181802, 1907.12628.

[18] Q. Arnaud *et al.*, *First germanium-based constraints on sub-MeV Dark Matter with the EDELWEISS experiment*, Phys. Rev. Lett. **125**(14), 141301 (2020), doi:10.1103/PhysRevLett.125.141301, 2003.01046.

[19] L. Barak *et al.*, *SENSEI: Direct-Detection Results on sub-GeV Dark Matter from a New Skipper-CCD*, Phys. Rev. Lett. **125**(17), 171802 (2020), doi:10.1103/PhysRevLett.125.171802, 2004.11378.

[20] I. Arnquist *et al.*, *First Constraints from DAMIC-M on Sub-GeV Dark-Matter Particles Interacting with Electrons*, Phys. Rev. Lett. **130**(17), 171003 (2023), doi:10.1103/PhysRevLett.130.171003, 2302.02372.

[21] A. Arvanitaki, S. Dimopoulos and K. Van Tilburg, *Resonant absorption of bosonic dark matter in molecules*, Phys. Rev. X **8**(4), 041001 (2018), doi:10.1103/PhysRevX.8.041001, 1709.05354.

[22] R. Essig, J. Pérez-Ríos, H. Ramani and O. Slone, *Direct Detection of Spin-(In)dependent Nuclear Scattering of Sub-GeV Dark Matter Using Molecular Excitations*, Phys. Rev. Research. **1**, 033105 (2019), doi:10.1103/PhysRevResearch.1.033105, 1907.07682.

[23] C. Blanco, J. I. Collar, Y. Kahn and B. Lillard, *Dark Matter-Electron Scattering from Aromatic Organic Targets*, Phys. Rev. D **101**(5), 056001 (2020), doi:10.1103/PhysRevD.101.056001, 1912.02822.

[24] Y. Wu, K. Freese, C. Kelso, P. Stengel and M. Valluri, *Uncertainties in Direct Dark Matter Detection in Light of Gaia's Escape Velocity Measurements*, JCAP **10**, 034 (2019), doi:10.1088/1475-7516/2019/10/034, 1904.04781.

[25] A. Radick, A.-M. Taki and T.-T. Yu, *Dependence of Dark Matter - Electron Scattering on the Galactic Dark Matter Velocity Distribution*, JCAP **02**, 004 (2021), doi:10.1088/1475-7516/2021/02/004, 2011.02493.

[26] T. N. Maity, T. S. Ray and S. Sarkar, *Halo uncertainties in electron recoil events at direct detection experiments*, Eur. Phys. J. C **81**(11), 1005 (2021), doi:10.1140/epjc/s10052-021-09805-2, 2011.12896.

[27] C. P. Liu, C.-P. Wu, J.-W. Chen, H.-C. Chi, M. K. Pandey, L. Singh and H. T. Wong, *Spin-dependent dark matter-electron interactions*, Phys. Rev. D **106**(6), 063003 (2022), doi:10.1103/PhysRevD.106.063003, 2106.16214.

[28] M. R. Buckley, S. H. Lim, E. Putney and D. Shih, *Measuring Galactic Dark Matter through Unsupervised Machine Learning* (2022), 2205.01129.

[29] T. N. Maity and R. Laha, *Dark matter substructures affect dark matter-electron scattering in xenon-based direct detection experiments*, JHEP **02**, 200 (2023), doi:10.1007/JHEP02(2023)200, 2208.14471.

[30] J. D. Lewin and P. F. Smith, *Review of mathematics, numerical factors, and corrections for dark matter experiments based on elastic nuclear recoil*, Astropart. Phys. **6**, 87 (1996), doi:10.1016/S0927-6505(96)00047-3.

[31] A. L. Fitzpatrick, W. Haxton, E. Katz, N. Lubbers and Y. Xu, *The Effective Field Theory of Dark Matter Direct Detection*, JCAP **02**, 004 (2013), doi:10.1088/1475-7516/2013/02/004, 1203.3542.

[32] R. Catena, D. Cole, T. Emken, M. Matas, N. Spaldin, W. Tarantino and E. Urdshals, *Dark matter-electron interactions in materials beyond the dark photon model*, JCAP **03**, 052 (2023), doi:10.1088/1475-7516/2023/03/052, 2210.07305.

[33] R. Essig, J. Mardon and T. Volansky, *Direct Detection of Sub-GeV Dark Matter*, Phys. Rev. D **85**, 076007 (2012), doi:10.1103/PhysRevD.85.076007, 1108.5383.

[34] E. Aprile *et al.*, *Dark Matter Search Results from a One Ton-Year Exposure of XENON1T*, Phys. Rev. Lett. **121**(11), 111302 (2018), doi:10.1103/PhysRevLett.121.111302, 1805.12562.

[35] C. Cheng *et al.*, *Search for Light Dark Matter-Electron Scatterings in the PandaX-II Experiment*, Phys. Rev. Lett. **126**(21), 211803 (2021), doi:10.1103/PhysRevLett.126.211803, 2101.07479.

[36] J. Aalbers *et al.*, *First Dark Matter Search Results from the LUX-ZEPLIN (LZ) Experiment* (2022), 2207.03764.

[37] E. Aprile *et al.*, *First Dark Matter Search with Nuclear Recoils from the XENONnT Experiment* (2023), 2303.14729.

[38] P. Agnes *et al.*, *Constraints on Sub-GeV Dark-Matter–Electron Scattering from the DarkSide-50 Experiment*, Phys. Rev. Lett. **121**(11), 111303 (2018), doi:10.1103/PhysRevLett.121.111303, 1802.06998.

[39] E. Aprile *et al.*, *Search for New Physics in Electronic Recoil Data from XENONnT*, Phys. Rev. Lett. **129**(16), 161805 (2022), doi:10.1103/PhysRevLett.129.161805, 2207.11330.

[40] T. Trickle, Z. Zhang, K. M. Zurek, K. Inzani and S. M. Griffin, *Multi-Channel Direct Detection of Light Dark Matter: Theoretical Framework*, JHEP **03**, 036 (2020), doi:10.1007/JHEP03(2020)036, 1910.08092.

[41] Y. Hochberg, Y. Kahn, N. Kurinsky, B. V. Lehmann, T. C. Yu and K. K. Berggren, *Determining Dark-Matter–Electron Scattering Rates from the Dielectric Function*, Phys. Rev. Lett. **127**(15), 151802 (2021), doi:10.1103/PhysRevLett.127.151802, 2101.08263.

[42] S. Knapen, J. Kozaczuk and T. Lin, *Dark matter-electron scattering in dielectrics*, Phys. Rev. D **104**(1), 015031 (2021), doi:10.1103/PhysRevD.104.015031, 2101.08275.

[43] R. Catena, T. Emken, N. A. Spaldin and W. Tarantino, *Atomic responses to general dark matter-electron interactions*, Phys. Rev. Res. **2**(3), 033195 (2020), doi:10.1103/PhysRevResearch.2.033195, 1912.08204.

[44] W. Yin, *Highly-boosted dark matter and cutoff for cosmic-ray neutrinos through neutrino portal*, EPJ Web Conf. **208**, 04003 (2019), doi:10.1051/epjconf/201920804003, 1809.08610.

[45] T. Bringmann and M. Pospelov, *Novel direct detection constraints on light dark matter*, Phys. Rev. Lett. **122**(17), 171801 (2019), doi:10.1103/PhysRevLett.122.171801, 1810.10543.

[46] H. An, M. Pospelov, J. Pradler and A. Ritz, *Directly Detecting MeV-scale Dark Matter via Solar Reflection*, Phys. Rev. Lett. **120**(14), 141801 (2018), doi:10.1103/PhysRevLett.120.141801, [Erratum: Phys.Rev.Lett. 121, 259903 (2018)], 1708.03642.

[47] T. Emken, C. Kouvaris and N. G. Nielsen, *The Sun as a sub-GeV Dark Matter Accelerator*, Phys. Rev. D **97**(6), 063007 (2018), doi:10.1103/PhysRevD.97.063007, 1709.06573.

[48] D. Baxter *et al.*, *Recommended conventions for reporting results from direct dark matter searches*, Eur. Phys. J. C **81**(10), 907 (2021), doi:10.1140/epjc/s10052-021-09655-y, 2105.00599.

[49] S. K. Lee, M. Lisanti and B. R. Safdi, *Dark-Matter Harmonics Beyond Annual Modulation*, JCAP **11**, 033 (2013), doi:10.1088/1475-7516/2013/11/033, 1307.5323.

[50] C. McCabe, *The Earth's velocity for direct detection experiments*, JCAP **02**, 027 (2014), doi:10.1088/1475-7516/2014/02/027, 1312.1355.

[51] B. Lillard, *earthspeed: For finding the instantaneous velocity of the Earth with respect to the galactic dark matter halo* (2024).

[52] D. G. York *et al.*, *The Sloan Digital Sky Survey: Technical Summary*, Astron. J. **120**, 1579 (2000), doi:10.1086/301513, astro-ph/0006396.

[53] A. G. A. Brown *et al.*, *Gaia Data Release 2: Summary of the contents and survey properties*, Astron. Astrophys. **616**, A1 (2018), doi:10.1051/0004-6361/201833051, 1804.09365.

[54] L. Necib, M. Lisanti and V. Belokurov, *Inferred Evidence For Dark Matter Kinematic Substructure with SDSS-Gaia* (2018), doi:10.3847/1538-4357/ab095b, 1807.02519.

[55] N. W. Evans, C. A. J. O'Hare and C. McCabe, *Refinement of the standard halo model for dark matter searches in light of the Gaia Sausage*, Phys. Rev. D **99**(2), 023012 (2019), doi:10.1103/PhysRevD.99.023012, 1810.11468.

[56] N. Bozorgnia, A. Fattahi, C. S. Frenk, A. Cheek, D. G. Cerdeno, F. A. Gómez, R. J. J. Grand and F. Marinacci, *The dark matter component of the Gaia radially anisotropic substructure*, JCAP **07**, 036 (2020), doi:10.1088/1475-7516/2020/07/036, 1910.07536.

[57] S. K. Lee, M. Lisanti, A. H. G. Peter and B. R. Safdi, *Effect of Gravitational Focusing on Annual Modulation in Dark-Matter Direct-Detection Experiments*, Phys. Rev. Lett. **112**(1), 011301 (2014), doi:10.1103/PhysRevLett.112.011301, 1308.1953.

[58] J. Bezanson, A. Edelman, S. Karpinski and V. B. Shah, *Julia: A fresh approach to numerical computing*, SIAM Review **59**(1), 65 (2017), doi:10.1137/141000671, https://doi.org/10.1137/141000671.

[59] M. Giordano, *Uncertainty propagation with functionally correlated quantities*, ArXiv e-prints (2016), 1610.08716.

[60] A. Radick, *SphericalHaarTransform.jl*, https://github.com/ariaradick/SphericalHaarTransform.jl (2025).

[61] S. G. Johnson, *The HCubature.jl package for multi-dimensional adaptive integration in Julia*, https://github.com/JuliaMath/HCubature.jl (2021).

[62] K. Chen, *MCIntegration.jl: Monte carlo integration in Julia*, https://github.com/numericalEFT/MCIntegration.jl (2017).

[63] M. Boyle, *Quaternionic.jl*, doi:10.5281/zenodo.5514302 (2023).

[64] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, *Numerical recipes in C (2nd ed.): the art of scientific computing*, Cambridge University Press, USA, ISBN 0521431085 (1992).