Computer Physics Communications 278 (2022) 108422

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

TauRunner: A public Python program to propagate neutral and charged leptons ☆,☆☆

Ibrahim Safa [a,b,*], Jeffrey Lazar [a,b,*], Alex Pizzuto [b], Oswaldo Vasquez [a], Carlos A. Argüelles [a], Justin Vandenbroucke [b]

[a] Department of Physics & Laboratory for Particle Physics and Cosmology, Harvard University, Cambridge, MA 02138, USA
[b] Department of Physics and Wisconsin IceCube Particle Astrophysics Center, University of Wisconsin-Madison, Madison, WI 53706, USA

ARTICLE INFO

Article history:
Received 3 January 2022
Received in revised form 4 May 2022
Accepted 17 May 2022
Available online 24 May 2022

Keywords:
Ultra-high energy
Neutrinos
Neutrino telescope
Simulation
Tau regeneration
Open source

ABSTRACT

In the past decade IceCube's observations have revealed a flux of astrophysical neutrinos extending to 10^7 GeV. The forthcoming generation of neutrino observatories promises to grant further insight into the high-energy neutrino sky, with sensitivity reaching energies up to 10^{12} GeV. At such high energies, a new set of effects becomes relevant, which was not accounted for in the last generation of neutrino propagation software. Thus, it is important to develop new simulations which efficiently and accurately model lepton behavior at this scale. We present `TauRunner`, a `Python`-based package that propagates neutral and charged leptons. `TauRunner` supports propagation between 10 GeV and 10^{12} GeV. The package accounts for all relevant secondary neutrinos produced in charged-current tau neutrino interactions. Additionally, tau energy losses of taus produced in neutrino interactions are taken into account, and treated stochastically. Finally, `TauRunner` is broadly adaptable to divers experimental setups, allowing for user-specified trajectories and propagation media, neutrino cross sections, and initial spectra.

Program summary
Program title: `TauRunner`
CPC Library link to program files: https://doi.org/10.17632/82nyd9skhj.1
Developer's repository link: https://github.com/icecube/TauRunner
Licensing provisions: GNU General Public License 3
Programming language: `Python`
Nature of problem: Propagation of ultra-high energy neutrinos in dense media accounting for various effects associated with ν_τ and $\tau^\pm$ energy losses.
Solution method: Monte Carlo methods.

1. Introduction

Most natural and anthropogenic neutrino sources produce neutrinos with energies below 1 TeV [1], where the smallness of the neutrino-nucleon cross section [2] allows them to freely stream through large amounts of column density—the density integrated along the neutrino trajectory. Famously, low-energy solar neutrinos produced in nuclear processes in the Sun are not only able to escape the dense solar core but also can diametrically traverse hundreds of Earths unimpeded. In this energy range, the negligible scattering rates imply that the problem of neutrino transport requires only considering the changing of flavors between neutrinos. This problem prompted the neutrino community to develop analytical methods and numerical schemes to compute the neutrino oscillation probabilities efficiently [3], *e.g.* `nuSQuIDS` [4] among others [5–8]. These solutions, currently available through a variety of software packages and libraries [9,10], are currently used by neutrino experiments to extract the neutrino oscillation parameters.

Recently, the construction of gigaton-scale neutrino detectors, such as the IceCube Neutrino Observatory [11] in the Antarctic

☆ The review of this paper was arranged by Prof. Z. Was.
☆☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).
* Corresponding authors at: Department of Physics and Wisconsin IceCube Particle Astrophysics Center, University of Wisconsin-Madison, Madison, WI 53706, USA.
E-mail addresses: isafa@fas.harvard.edu (I. Safa), jeffreylazar@fas.harvard.edu (J. Lazar).

https://doi.org/10.1016/j.cpc.2022.108422
0010-4655/

continent, has enabled the observation of neutrinos with energies as large as 10 PeV. In this high-energy frontier, neutrino oscillations can be safely neglected for Earth-traversing neutrinos; however, in this regime, the neutrino interaction length becomes comparable to or much smaller than Earth's diameter [12], requiring new solutions to the neutrino transport problem. While the first generation of software packages that aimed to address this problem [13–15,4,16,17] included the effects of neutrino-nucleon neutral- and charged-current interactions, they neglected secondary neutrinos from lepton charged-current interactions, except in the case of tau neutrinos. Tau neutrinos were handled as a special case because, as recognized in [18,19], due to the short lifetime of the taus, it still carries most of its energy at the time of decay yielding high-energy secondary neutrinos. This effect, often known as tau regeneration, implies that Earth is less opaque to tau neutrinos relative to other flavors.

In these first-generation packages tau regeneration was implemented by using the so-called on-spot tau decay approximation, which neglects tau energy losses. Though this approximation satisfies the needs of most current scenarios and experimental settings, next-generation neutrino telescopes aim to reach EeV energies [20,21]. At these extremely high energies, the taus produced in neutrino interactions are sufficiently long-lived that their energy losses cannot be neglected. Recently, dedicated software packages have been made available to solve this problem in this energy regime. However, the bulk of the available solutions neglects the stochasticity of tau losses considering only their mean effect. This limits their ability to function as event generators in neutrino telescopes and produces mismodeling of the yield of tau-induced events for a small number of scatterings, where the stochastic nature of the losses is more relevant. A notable exception is the `NuPropEarth` [22] package developed for the KM3NeT experiment [23], which is presently being built in the Mediterranean Sea. Though `NuPropEarth` offers a complete solution, this package requires a large number of dependencies to function, making its distribution and installation difficult. In this article, we describe a new package, `TauRunner`, that aims to provide a complete and versatile solution to the neutrino transport problem at high energies. Our `Python`-based package is designed to have minimal dependencies, to allow the user to construct arbitrary neutrino trajectories and propagation media, and to provide interfaces to modify physics inputs such as neutrino cross sections easily. This package was first introduced in [24,25], where it was used to study the ANITA anomalous events [26,27], and is currently used in studies relating to extremely high-energy neutrinos in IceCube [28]. With respect to the preliminary version, the version presented in this paper contains significant improvements in terms of performance and available features to the user. In this article, we describe the software and provide examples, benchmarks and comparisons to other packages that have similar aims. We expect that our software will be useful for next-generation neutrino detectors operating in liquid water (P-ONE [29]), solid water (IceCube-Gen2 [20]), mountains (Ashra NTA [30], TAMBO [31]), and outer space (POEMMA [21]). Our hope is that the success of neutrino oscillation measurements enabled by the previous generation of software will be mirrored in the study of high-energy neutrino properties with efficient propagation software such as the one presented in this paper.

The rest of this article is organized as follows. In Sec. 2 we outline the transport equation, the algorithm used to solve it, and the interaction; in Sec. 3 we explain the code structure; in Sec. 4 we present studies of the software performance; in Sec. 6 we lay out the examples included with the code. Finally in Sec. 7 we conclude.

2. Algorithm overview

The aim of this software is to solve the transport equation for high-energy neutrino fluxes passing through matter. The transport equation can be written as follows [34],

$$\frac{\mathrm{d}\vec{\varphi}(E,X)}{\mathrm{d}X} = -\sigma(E)\vec{\varphi}(E,X) + \int_E^\infty \mathrm{d}\tilde{E}\; f(\tilde{E},E)\vec{\varphi}(\tilde{E},X), \tag{1}$$

where E is the neutrino energy, X is the target column density, $\sigma(E) = \mathrm{diag}(\sigma_\nu, \sigma_{\bar{\nu}})$ holds the total ν and $\bar{\nu}$ cross section per target nucleon, $f(\tilde{E},E)$ is a function that encodes the migration from higher to lower neutrino energies and between ν and $\bar{\nu}$, and $\vec{\varphi}(E,x) = \{\phi_\nu, \phi_{\bar{\nu}}\}$ contains the neutrino and anti-neutrino spectrum. At energies supported by this package, 10 GeV–10^{12} GeV, neutrino-nucleon deep inelastic scattering (DIS) is the dominant neutrino interaction process. The first term on the right hand side accounts for the loss of flux at energy E due to charged-current (CC) and neutral-current (NC) interactions, whereas the second term is the added contribution from neutrinos at higher energy, $\tilde{E}$, to E through NC interactions of $\nu_{e,\mu,\tau}$ and CC interactions in the ν_τ channel.

This latter channel is unique in that the short τ lifetime causes the decay of the charged lepton before losing a large fraction of the parent energy. The τ then decays into a daughter ν_τ, meaning that the primary ν_τ flux is not lost, but only cascades down in energy. Moreover, if the τ decays leptonically, $\bar{\nu}_\mu$ and $\bar{\nu}_e$ are created, contributing significantly to the outgoing flux, as was first pointed out in [35]. By default, `TauRunner` takes all those contributions into account. The story is simpler for the electron channel. There, CC interactions result in electrons which lose their energy quickly and are subsequently absorbed in the medium. As a result, electron losses are not modeled in `TauRunner` by default, though the capability exists if needed. For the muon flavor, muons resulting from CC interactions can travel $\mathcal{O}(1)$ kmwe. Therefore, it is important to model the propagation and losses of muons near the point of exit, and that is accounted for in `TauRunner` as well.

2.1. Algorithm description

In `TauRunner`, Eq. (1) is solved using a Monte-Carlo approach. A flowchart of the `TauRunner` Monte-Carlo algorithm is shown in Fig. 1. Given an initial neutrino type, energy, and incident angle, it begins by calculating the mean interaction column depth, λ_{int}, which depends on the medium properties and neutrino cross section. A column depth is then randomly sampled from an exponential distribution with parameter λ_{int}, and the neutrino advances the corresponding free-streaming distance. If the neutrino does not escape the medium, either an NC or CC interaction is chosen via the accept/reject method. In the case of an NC interaction, the neutrino energy loss is sampled from the differential cross section, and the process repeats. In the case of a CC interaction, a charged lepton is created with energy sampled from the neutrino differential cross section.

The treatment of the charged lepton then varies according to the initial neutrino flavor. Electrons are assumed to be absorbed and the propagation stops there. μ and τ, however, are recorded and passed to `PROPOSAL` [36] to be propagated through the same medium. μ that do not escape will either decay at rest resulting in neutrinos that are below the energies supported by `TauRunner`, or get absorbed. Therefore a μ that does not escape is not tracked further. Finally, τs can either escape or decay. In the latter case, a secondary ν_τ is created whose energy is sampled from tau decay distributions provided in [37]. Additionally, if the τ decays leptonically, ν_e or ν_μ will be created. When this happens, the

Table 1
Software comparison table. Each row of this table represents a given package. Input and output particles include their not explicitly mentioned antiparticles. Custom medium refers to a user-defined `Body` in `TauRunner`. The Energy losses column compares the treatment of charged particle losses.

Software	Language	Input	Output	Medium	Energy losses ($l^{\pm}$)
`TauRunner`	`Python`	$\nu_{\tau,\mu,e}, \tau, \mu$	$\nu_{\tau,\mu,e}, \tau, \mu$	Earth/Sun/Moon/Custom	`PROPOSAL`
`NuPropEarth` [22]	`C++`	$\nu_{\tau,\mu,e}$	$\nu_{\tau,\mu,e}, \tau$	Earth/Custom	`TAUSIC`
`nuPyProp` [32]	`Python/FORTRAN`	ν_τ	τ	Earth	Internal
`NuTauSim` [33]	`C++`	ν_τ	τ	Earth	Continuous

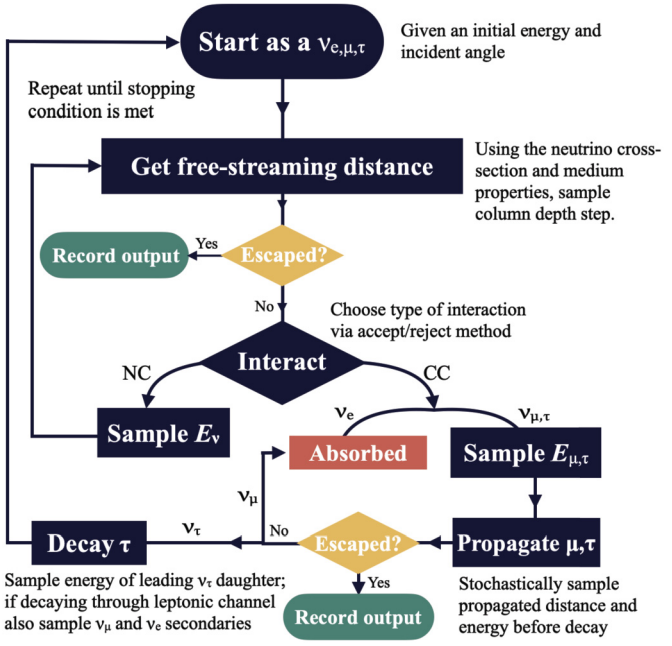

Fig. 1. *Flowchart of the `TauRunner` propagation algorithm.* Square boxes indicate actions performed by the software. Diamond boxes indicate decision-making stopping points. Rounded-corner squared boxes indicate beginning and end of the algorithm. Note that users can select also charged leptons as the initial state, in which case the algorithm skips straight to the charged particle propagation step.

properties of the resulting secondaries are recorded and added to a basket which stores all secondary particles to be propagated together after the primary particle propagation is complete.

2.2. Lepton interactions and decays

Measurements of neutrino cross sections with matter have been performed up to a few PeV in energy [38]. This includes a multitude of accelerator [39,40] and reactor [41,42] experiments as well as solar [43], atmospheric [44], and astrophysical neutrinos [45,46]. However, the energy range supported by `TauRunner` goes far beyond the measurements, where the fractional momenta, x_{Bjorken}, of the quarks probed by the neutrino can reach $x_{\text{Bjorken}} \ll 10^{-8}$. The nucleon structure function is not measured at such low x_{Bjorken} and is extrapolated in cross section calculations [47,22]. Such extrapolations neglect gluon color screening making perturbative QCD calculations of the neutrino cross section grow faster than allowed by unitarity at extremely high energies [48]. Phenomenological approaches to include gluon screening parameterize the extremely small x_{Bjorken} behavior using a dipole model [49] of the nucleon so as to result in a $\ln^2(s)$ dependence of the cross section at extremely high energies [50]. This ultimately results in a difference of a factor ~ 2 at 10^{12} GeV. `TauRunner` provides, by default, neutrino and anti-neutrino DIS cross section tables for two PDF models: a perturbative QCD calculation [47], and a dipole model [49]. The user also has the option to provide their own cross sections, see Sec. 3.4 for more details.

In the Standard Model, when neutrinos undergo CC interactions, they convert to their charged partners through the exchange of a W boson. Charged particles lose energy in dense media through many processes, and the relative importance of each process depends on the lepton's mass and its energy [51]. At lower energies, a charged lepton can ionize atoms as it traverses the medium. This process is described by the Bethe-Bloche equation, and at higher energies scales logarithmically and becomes sub-dominant for all flavors. A charged lepton can also interact with the electric field of a nucleus, losing energy in the process through the emission of a photon. This process, called bremsstrahlung, scales like the inverse-squared mass of the lepton, and is therefore the dominant energy loss mechanism for electrons. Another possible interaction with the field of a nucleus leads to the production of electron-positron pairs. This process scales like the inverse of the lepton mass, and is one of the leading energy-loss mechanisms for μ and τ. Finally, the leptons can also lose energy by exchanging a photon with a nucleon, in what is referred to as a photonuclear interaction. This process dominates tau energy losses at the highest energies ($\geq 10^9$ GeV). The aforementioned processes are implemented in `PROPOSAL`, which we use to model them in `TauRunner`. Apart from interacting, μ and taus can also undergo weak decays. This process scales like the mass of the lepton to the fifth power, and is therefore the most likely outcome for taus propagating in Earth up to 10^9 GeV. Above this energy, the total interaction length for other processes becomes shorter than the decay length. μ, on the other hand, are much more likely to lose all of their energy before decaying at rest, or getting absorbed by a nucleus. Therefore, we only model decays of τ leptons using parametrizations in [37].

3. Structure of the code

`TauRunner` may be run either from the command line by running `main.py` or may be imported to run within another script or `Jupyter` notebook. To run from the command line, the user must minimally specify the initial energy, the incident nadir angle, and the number of events to be simulated. These can be specified with the `-e`, `-t`, and `-n` command line flags respectively. This will run the `TauRunner` algorithm in Earth with a chord geometry. The `TauRunner` output will be printed in the terminal unless an output file is specified with the `--save` flag. If this option is specified, `TauRunner` will save both a `numpy` array and a `json` file with the configuration parameters at the specified location. In order to ensure reproducibility, the user may specify a seed for the random number generator with the `-s` flag. By default, `main.py` propagates an initial ν_τ flux, but a user may specify other initial particle types by using the `--flavor` flag. Additional options that may be specified by the user can be found in the `initialize_args` function of `main.py` or by running `main.py` with the `-h` flag.

To run within another script or `Jupyter` notebook the user must import the `run_MC` function from `main.py`. In this latter case one must also create a `TauRunner` `Particle`, `Track`, `Body`, `CrossSection` objects and a `PROPOSAL` propagator. The `Particle` class, described in Sec. 3.1, contains the particle properties as well as methods for particle propagation. The `Track` class, described in Sec. 3.2, parametrizes the geometry of the par-

ticle trajectories. The `Body` class, described in Sec. 3.3, defines the medium in which the propagation is to occur. The `CrossSection` class, described in Sec. 3.4, defines neutrino cross section model. Additionally, `TauRunner` provides a convenience function for constructing `PROPOSAL` propagators, `make_propagator`, which can be imported from the `utils` module. Explicit examples of how to run `TauRunner` can be found in Sec. 6. `Casino.py` combines these classes according to the logic outlined in Fig. 1.

After discussing the package broadly, we will discuss conventions in Sec. 3.6 and describe `TauRunner`'s output in Sec. 3.7

3.1. `Particle`

A `Particle` instance contains the structure of a `TauRunner` event. This includes, among other quantities, the particle's initial and current energies, particle type, and position. Additionally, it has a number of methods for particle decay and interaction as well as charged lepton propagation. Finally, the τ decay parametrization is contained in `particle/utils.py`.

The user may propagate ν_e, ν_μ, ν_τ, μ^-, τ^-, or any of the corresponding anti-particles in `TauRunner`. To do this, the user should initialize the `Particle` object with the corresponding Particle Data Group Monte Carlo number [51]. It should be noted that the user may create an $e^\pm$, but the internal logic of `TauRunner` assumes all $e^\pm$ are immediately absorbed and thus no propagation occurs; see Fig. 1.

3.2. `Track`

The `Track` class contains the geometrical information about the particle's trajectory. A track is parametrized by an affine parameter which defines the position along the trajectory: 0 is the beginning of the trajectory, and 1 is the end. Almost all of the methods of the `Track` class are mappings between the affine parameter and physically relevant quantities, *e.g.* radius, distance traveled, and column depth. The only argument which is generic to the `Track` class is `depth` which specifies the distance below the surface of the body at which to stop propagation. This may intuitively be thought of as the depth of the detector to which the particles are propagated. An illustration of the `TauRunner` geometry and a diagram of the functional relation of physical quantities to the affine parameter is shown in Fig. 2

The `Track` class allows the user to make custom trajectories. The user need only specify mappings between the affine parameter and these variables. Different trajectories may require additional arguments from the user, depending on the nature of the trajectory. To illustrate this point, we can look at the two tracks which are implemented by default, the `Chord` and `Radial` trajectories. The former is used for paths which originate outside the `Body` and cross a section of `Body`. The latter is used for paths which originate at the center of the `Body`. The former `Track` describes neutrinos coming from space and passing through Earth on the way to a detector, as in the case of Earth-skimming τ searches, while the latter gives the trajectory of a neutrino originating in the center of the planet, relevant for searches for neutrinos from gravitationally trapped dark matter. Clearly, an incoming angle needs to be specified for the `Chord` trajectory. Thus, we can see that the necessary arguments for specifying a `Track` may vary from one geometry to another.

3.3. `Body`

The `Body` class specifies the medium in which the `Particle` is to be propagated. In `TauRunner`, we require that all bodies be

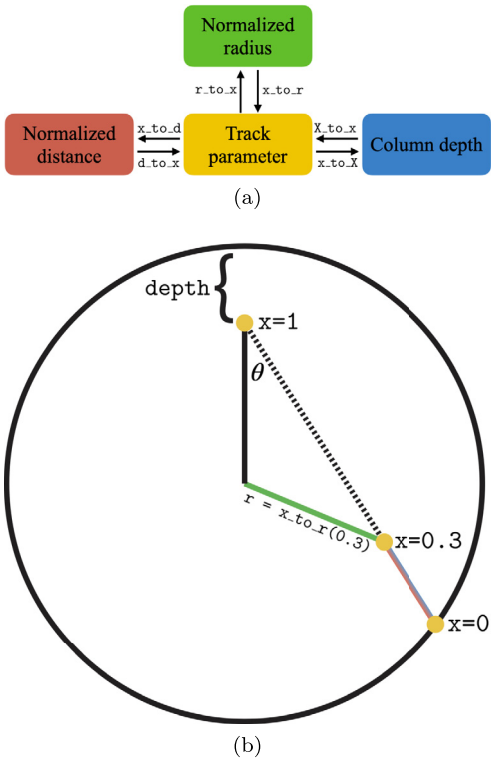

Fig. 2. *Schematic of `TauRunner` geometry as contained within the `Track` class.* (a) shows the relation between the physical quantities relevant to propagation and the affine parameter that parametrizes the `Track`. The arrows connecting these quantities are labeled with the functions used to convert between them in `TauRunner`. Specifically, these are the functions a user must define in order to specify a custom `Track` geometry. All distances are normalized with respect to the radius of the body in which the track sits. (b) shows a diagram of these parameters within a spherical `TauRunner` body. Colors correspond to the boxes in (a). Additionally, it illustrates the `depth` parameter which intuitively gives the depth of the detector. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

spherically symmetric, and so a `Body` may be minimally specified by a physical radius, and a density profile. The density profile may be a positive scalar, a unary function which returns a positive scalar, or a potentially-mixed list of positive scalars and such functions. The sole argument of the functions used to specify the density should be the radius at which the density is to be given, in units of the radius of the body, *i.e.* the domains should be $[0, 1]$. In this system $r = 0$ is the center of the body and $r = 1$ the surface. If the user wishes to make a layered body, *i.e.* one where a list specifies the density profile, they must pass a list of tuple with the length of this list equal to the number of layers. The first element of each tuple should be the scalar or function which gives the density, and the second element should be the right hand boundary of the layer in units of the radius. The last right hand boundary should always be 1 since $r = 1$ is the outer edge of the body. Lastly, all densities should be specified in g/cm^3.

In addition to a radius and a density profile, the user may also provide the `proton_fraction` argument to specify the fraction of protons to total nucleons in the body. By default, we assume that the propagation medium is isoscalar, *i.e.* we set the proton fraction to 0.5 throughout the entire body. As in the case of the density profile, this argument may be a scalar, a function, or a list of function-boundary tuples. The domains of any functions provided must be $[0, 1]$, and the ranges must be in this same interval.

While the user can construct bodies themselves, there are five bodies implemented by default in `TauRunner`: the Earth, a high-metallicity Sun, and low-metallicity Sun, the moon, a constant density slab. We use the PREM parametrization to model the densities

of Earth [52]. For the Sun, we use fits provided by [53]. To instantiate the `Earth` object, one calls the `construct_earth` function, which returns an `Earth` object. Additionally, this function allows one to pass in a list of additional layers which will be placed radially outward from the edge of the PREM Earth. This functionality may be useful for *e.g.* adding a layer of water or ice or adding the atmosphere for simulating atmospheric air showers. Examples on using this functionality may be found in Sec. 3.3. To initialize the Sun, one can use the `construct_sun` function. With this function, the user may specify 'HZ_Sun' or 'LZ_Sun' to use the high- and low-metallicity `TauRunner` suns respectively, or a path to a user defined solar model. An example of how to input solar models is given in Appendix C

3.4. `CrossSection`

The `TauRunner` cross sections module defines the neutrino interactions. Internally, `TauRunner` assumes that cross sections are equal for all neutrino flavors. Additionally, `TauRunner` uses the isoscalar approximation by default, *i.e.* it assumes a medium is made of equal parts p^+ and n; however, this assumption may be changed by altering the `proton_fraction` of the `Body` object. See Sec. 3.3 for more information. The software includes both CSMS [47] and dipole [54] cross sections implemented by default; however, it is straightforward for the user to implement other cross section models by providing `scipy` splines in the appropriate format. For the total neutrino cross section these splines are `scipy.interpolate.UnivariateSpline` objects whose x-axis is the $\log_{10}$ of the neutrino energy in eV and whose y-axis is the $\log_{10}$ of cross section in cm^2. The differential cross section splines are `scipy.interpolate.RectBivariateSpline` objects whose x-axis is the $\log_{10}$ of the neutrino energy in eV, whose y-axis is a convenience variable which combines the incoming and outgoing neutrino energies, E_{in} and E_{out}, given by

$$\eta = \frac{E_{\text{out}} - 10^9\ \text{eV}}{E_{\text{in}} - 10^9\ \text{eV}},$$

and whose z-axis is the $\log_{10}$ of incoming energy times the differential cross section in cm^2. An example of how to construct these splines is given in Appendix B.

3.5. `PROPOSAL`

To propagate charged leptons, `TauRunner` relies on `PROPOSAL`, an open source C++ program with `Python` bindings. A utility module to interface with `PROPOSAL`, `utils/make_propagator.py`, is provided with `TauRunner`. This function instantiates `PROPOSAL` particle and geometry objects, which are then used to create a propagator instance. Since `PROPOSAL` does not support variable density geometries, the `segment_body` function is used to segment the `TauRunner` body into a number of constant density layers. The number of layers is determined by solving for points in the body where fractional change in the density is equal to a constant factor, called `granularity`. This argument may be specified by the user, and by default is set to 0.5. A single propagator object is created for all $\tau^\pm$ and, if needed, for all $\mu^\pm$. Since `TauRunner` assumes $e^\pm$ are always absorbed, a propagator will never be made for these. Whenever a new geometry is used, `PROPOSAL` creates energy loss tables which are saved in `resources/proposal_tables`. The tables require a few minutes to generate, resulting in an overhead for new configurations, but subsequent simulations with the same geometry will not suffer any slow down.

3.6. Conventions

`TauRunner` uses a natural unit system in which $\hbar = c = \text{eV} = 1$. As a consequence of this system, any energy passed to `TauRunner` must be in eV. `TauRunner` includes a `units` package to easily convert common units to the units `TauRunner` expects. This may be imported from the `utils` module, and its usage is demonstrated in several examples. Additionally, since `TauRunner` assumes that propagation occurs in a spherical body, the radius of this body establishes a natural length scale. Thus all distances are expressed as a fraction of this radius.

3.7. Output

The `run_MC` function, which carries out the logic of `TauRunner`, returns a `numpy.recarray`. This array may be set to a variable if running `TauRunner` from a script of notebook, or printed or saved if running `TauRunner` from the command line.

In this paragraph, we will describe the fields of this output. The `"Eini"` field reports the initial energy of the lepton in eV. The `"Eout"` field reports the energy of the particle when propagation has stopped in eV. In the case that the particle was absorbed, this field will always read `0.0`. The `"theta"` field reports the incident angle of the lepton in degrees. The `"nCC"` and `"nNC"` fields report the number of charged and neutral current interactions the particle underwent in its propagation. The `"PDG_Encoding"` field reports the particle type, using the Particle Data Group MC numbering scheme. The `"event_ID"` is a number carried byfield reports which initial lepton the particle comes from. The `"final_position"` field reports the track parameter when the propagation was ended. This may be used to physical quantities of a particle when it was absorbed, or when a user-defined stopping condition was met

4. Performance

For a given primary spectrum and medium through which to propagate, there are a variety of related factors that determine the runtime of the program, including, but not limited to: (1) the initial energy of the neutrinos, (2) the total column depth of the path, (3) the settings for computing energy losses, and (4) which particles are being tracked.

We show example runtimes for a few different use cases in Fig. 3. For a fixed `Track` propagating through Earth, neutrinos with higher initial energy take longer to propagate as they undergo more interactions and as a result experience more stochastic energy losses. Additionally, those particles that are only being propagated through Earth-skimming trajectories ($\cos(\theta) \approx 0$) can be simulated much quicker than those with large column depths. This is especially advantageous for proposed Earth-skimming next generation neutrino observatories, *e.g.* [21,55–57,31].

By default, all secondary particles that are created as a result of interactions are recorded, meaning that every ν_τ CC interaction has a chance to increase the number of particles that need to be simulated. If the user is only interested in outgoing ν_τ and τ lepton distributions, this option can be disabled with by setting `no_secondaries=True`, which can improve the overall runtime by as much as a factor of two.

Runtime can further be reduced depending on the treatment of energy losses of charged leptons. By default, energy losses are handled by `PROPOSAL` [36], which treats them stochastically. The user has the choice to ignore energy losses completely, with the setting `no_losses=True`, which can improve the runtime by as much as 40%, although this approximation can only be used in certain scenarios, such as when the initial tau lepton energy is small enough that the interaction length becomes much smaller

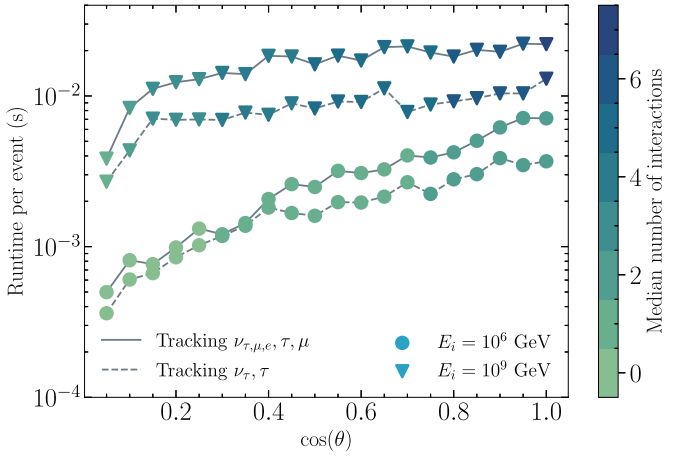

Fig. 3. *Runtime per ν_τ event.* Average runtime per event for various monochromatic fluxes of neutrinos through the Earth, as a function of nadir angle, θ for incident ν_τ with energies of 1 PeV (circles) and 1 EeV (triangles). In general, runtime scales with the average number of interactions, which is a function of the energy of the particles and the column depth through which they propagate. The colorbar indicates the median number of NC+CC interactions that the initial beam of ν_τ undergoes. Tracking secondary particles (solid lines) created in ν_τ CC interactions increases the runtime as the number of particles to propagate increases. Each point represents the average runtime from a simulation including 10^6 events on a single CPU.

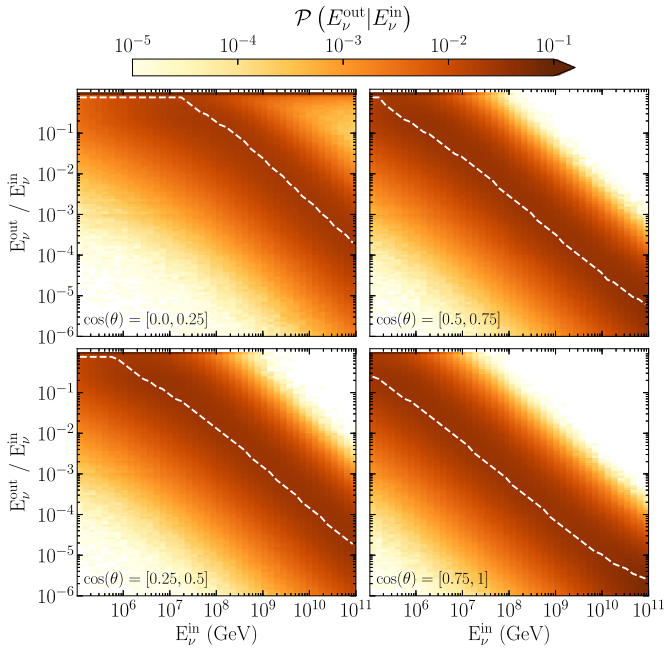

Fig. 4. *Outgoing ν_τ distributions for an E^{-1} power-law flux.* Shown are the outgoing tau neutrino energy fraction as a function of the primary ν_τ flux injected as an E^{-1} power-law from 100 TeV to 10 EeV, shown in slices of equal solid angle in the Northern Sky. The dashed line indicates the median outgoing energy.

than the decay length. This has potential applications for recently proposed indirect searches of ultra-high-energy neutrinos by looking for PeV neutrinos through the Earth [24] using large current and next-generation ice or water Cherenkov detectors, such as IceCube-Gen2 [20]. Within `PROPOSAL`, there is also an option to treat energy losses that are below a certain threshold continuously. We find that setting this parameter to `vcut=1e-3`, meaning all energy losses that represent less than that fraction of the initial particle energy are treated without stochasticity, achieves an optimal runtime while not neglecting any of the important features that are a result of treating energy losses stochastically.

The first time that a user runs the code, there may be additional overhead while `PROPOSAL` calculates energy loss distributions for charged leptons. However, these tables are stored so that future iterations can run more efficiently. Once the user has run the code at least once and the `PROPOSAL` energy loss tables are stored, then current runtimes allow users to propagate approximately one million initial EeV ν_τ through Earth's diameter in approximately eight hours with one CPU. For an initial energy of one PeV, one million ν_τ take approximately one hour, depending on the incident angle. We also found that this runtime varied marginally from machine to machine, and the runtimes in Fig. 3 and the numbers quoted thus far were all found using a heterogeneous distributed cluster of Linux machines. The code was also tested on a machine running MacOS with the Apple M1 chip, where the runtimes were found to extremely comparable to those presented above. For example, 10^4 ν_τ with initial energy of one EeV and $\theta = 0°$ with no secondaries took 0.0127 s per event, on average, and those in the figure above took 0.0124 s per event, on average.

In terms of memory, `TauRunner` can be run on most modern machines, requiring only a few GB of RAM to run. For example, propagating 10^4 ν_τ through the Earth with initial energies of an EeV requires only approximately 1 GB of memory when tracking only ν_τ and τ, and approximately 3 GB when tracking all particles. The vast majority of this memory is allocated for calculating energy losses with PROPOSAL, *e.g.* for various trajectories through the Earth and for various initial energies, we found that $\sim 50-90\%$ of the memory usage was due to `PROPOSAL`. Because most of the memory is due to overhead from the energy losses, there is only a marginal increase in memory usage from propagating many more particles, *e.g.* two sample iterations of the code both took between 2.5 GB and 3.0 GB when propagating 10^4 or 10^6 ν_τ through the Earth with the same initial energies and angles.

5. Outputs and comparisons

The results of several tau neutrino simulation sets are illustrated in this section. Fig. 4 shows column-normalized distributions of outgoing neutrino energy fraction as a function of initial neutrino energy. Interestingly, the dashed line showing the median outgoing tau neutrino energy fraction varies with a constant slope, corresponding to the energy at which Earth becomes transparent. That energy is roughly 10 PeV at the horizon (top left), $\mathcal{O}(1)$ PeV in the mantle (top right and bottom left), and $\mathcal{O}(10)$ TeV through the core (bottom right). This means that for a large fraction of the Northern Sky, tau neutrinos pile-up and escape at energies where the atmospheric neutrino background is relatively low. This idea is also made clear when illustrated for a monochromatic flux. In Fig. 5, EeV tau neutrinos are propagated and the outgoing energies are plotted as a function of nadir angle. A similar feature can be seen, where a majority of neutrinos in this simulation escape with energy above 100 TeV.

`TauRunner` has also been compared to several publicly available packages that perform similar tasks. A summary of the various tested packages and their features is shown in Table 1. Besides `TauRunner`, only `NuPropEarth` offers a full solution in the case of tau neutrinos. To illustrate this, we show in Fig. 6 the output of both packages for an injected monochromatic flux of tau neutrinos at 10^{10} GeV and one degree below the horizon. For secondary taus and tau neutrinos, the two packages show excellent agreement. We note that comparisons with `NuPropEarth` use the trunk version of the code, which has a new treatment for charged particle propagation using `PROPOSAL` instead of `TAUSIC`. Secondary anti-muon and -electron neutrino distributions show slight disagreement in the tails, likely due to different tau polarization treatments. These differences are still being investigated, and will be addressed in an upcoming work.

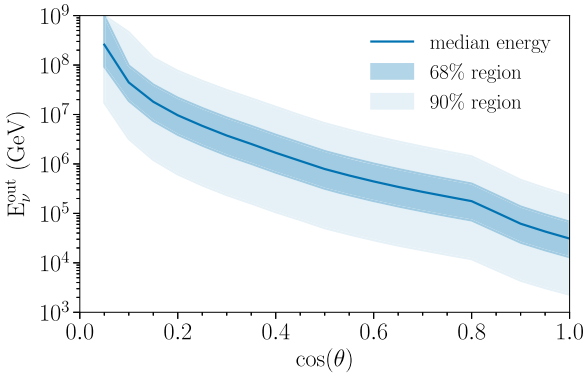

Fig. 5. *EeV tau neutrinos in Earth.* Median outgoing energies of secondary tau neutrinos shown as a function of nadir angle. Also, 68% and 90% probability contours for outgoing energies are included. The feature at approximately $\cos\theta$ of 0.8 is caused by the core.

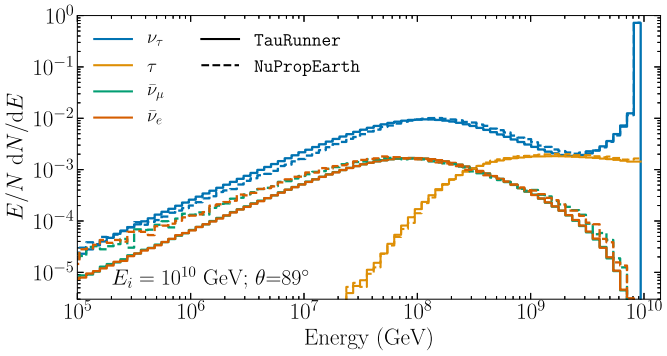

Fig. 6. *A monochromatic flux of tau neutrinos.* Outgoing particle energy distributions for a fixed angle and energy. We include secondary anti-electron and -muon neutrinos, as well as charged taus. `TauRunner` shows good agreement with `NuPropEarth`. This set assumes Earth as a body with a 4 km layer of water.

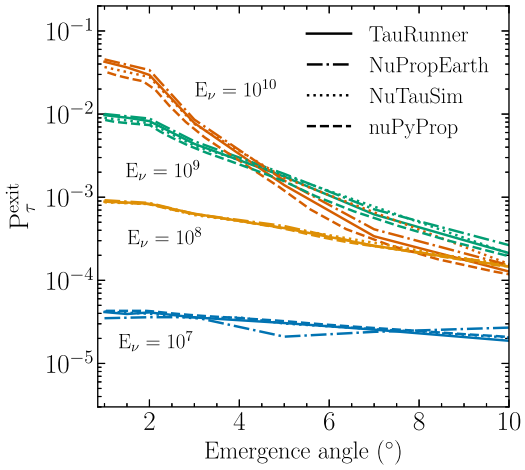

Fig. 7. *Charged tau lepton exit probability.* Different colors correspond to four different monochromatic neutrino energies. The emergence angle is measured with respect to horizon. The `TauRunner` prediction (solid line) is compared to NuTauSim, NuPropEarth, and nuPyProp, which are shown in different linestyles.

Fig. 7 shows a comparison of the charged tau exit probability in Earth as a function of nadir angle. P^{τ}_{exit} is the probability that an incoming neutrino will exit Earth as a charged tau. This quantity is especially relevant for future neutrino observatories hoping to detect Earth-skimming tau neutrinos. In that scenario, exiting taus make up the bulk of the expected signal. `TauRunner` again shows great agreement overall with other packages.

```
CLONE_DIR=/path/to/clone/directory
cd $CLONE_DIR
git clone https://github.com/icecube/TauRunner.git && cd
    TauRunner
pip3 install -e .
```

Listing 1: Installing `TauRunner` using `pip3` with access to source files.

```
CLONE_DIR=/path/to/clone/directory
cd $CLONE_DIR
git clone https://github.com/icecube/TauRunner.git
export PYTHONPATH=$PYTHONPATH:$CLONE_DIR/TauRunner
```

Listing 2: Installing `TauRunner` from source.

6. Examples

In this section, we show examples which illustrate many of the capabilities of `TauRunner`. `TauRunner` can be run from the command line or imported as a package. When a feature can be used via both interfaces, we provide an example for each.

6.1. Installation

`TauRunner` can be installed using `pip3` by running:

```
pip3 install taurunner
```

This will also install any required dependencies, which include `numpy` [58], `scipy` [59], and `PROPOSAL` [36].

Furthermore, certain use cases may require access to the source code, which can be downloaded from the `TauRunner` GitHub. After obtaining the source code, one can install the source code with the package manager `pip3`, while allowing the user to make edits to the source code without having to reinstall the package (see Listing 1).

Alternatively, for those that do not use the `pip3` package manager, one can install all of the dependencies listed in the `requirements.txt` file included on GitHub, and then clone the repository and add the base directory to the `PYTHONPATH` variable, as shown in Listing 2.

6.2. Monochromatic through Earth

Here we give an example of how to use the most fundamental functionality of `TauRunner`: propagating a monochromatic flux of neutrinos at a fixed energy through a body at a fixed angle (see Listing 3).

If you are using the source code installation, you may also achieve this same effect from the command line in the following manner:

```
python main.py -n 1000 -e 1e19 -t 89 --xs CSMS -s 7 --save
    /path/to/outdir/output.npy
```

The `--save` flag tells the program where to save the output. If this is not specified, the output will be printed as a table.

6.3. Isotropic flux through Earth with power law distribution

`TauRunner` also allows the user to sample initial neutrino energies from a power law distribution. For this, the user must provide bounds on the minimum and maximum allowed energies. Furthermore, the user may sample incidence angles to simulate a isotropic flux. We demonstrate bot of these features in the example shown in Listing 4.

This may also be accomplished via the command line interface by running:

```
python main.py -n 1000 -e -2 --e_min 1e15 --e_max 1e21 -t
    range --th_min 0 --th_max 90 -s 7 --xs CSMS
```

```python
import numpy as np

from taurunner.main import run_MC
from taurunner.body.earth import construct_earth
from taurunner.cross_sections import CrossSections
from taurunner.utils import make_propagator, make_initial_e
    , make_initial_thetas

nevents  = 5000 # number of events to simulate
eini     = 1e19 # initial energy in GeV
theta    = 89.0 # incidence angle (nadir)
pid      = 16 # PDG MC Encoding particle ID (nutau)
xs_model = "CSMS" # neutrino cross section model

Earth    = construct_earth(layers=[(4., 1.0)]) # Make Earth
    object with 4km water layer
xs       = CrossSections(xs_model)
energies = make_initial_e(nevents, eini) # Return array of
    initial energies in eV
thetas   = make_initial_thetas(nevents, theta)

tau_prop = make_propagator(pid, Earth)
rand     = np.random.RandomState(seed=7)

output = run_MC(energies,
                thetas,
                Earth,
                xs,
                tau_prop,
                rand,
        )
```

Listing 3: *Propagating a monochromatic flux from a single angle.* Example of using an independent script to propagate a monochromatic flux of neutrinos with initial energy $E_\nu = 10^{10}$ GeV from a nadir angle of 89°, *i.e.* one degree below the horizon.

6.4. Custom flux through Earth

The user may also input custom spectra to sample from. These should be given to `TauRunner` as pickled splines of the flux's cumulative density function. An example on how to construct these splines in the appropriate format is given in Appendix A. The default `TauRunner` distribution includes splines of different GZK models. In this example, we show how to sample energies according to the flux predicted in [60] (see Listing 5).

This may also be accomplished using the command line interface by running:

```
python main.py -n 1000 -e ./resources/ahlers2010_cdf_spline
    .pkl -t range --th_min 0 --th_max 90 -s 7 --xs CSMS
```

6.5. Radial trajectory

Besides the chord trajectory, which simulates neutrinos passing through a body from one side to the other, `TauRunner` provides a radial trajectory, which simulates neutrinos originating from the center of a `Body`. To use this, one need only modify the call to the `make_tracks` function. Note that the `theta` argument which was specified previously has no bearing on this, but must be passed due to implementation issues (see Listing 6).

This can also be accomplished from the command line by running:

```
python main.py -n 1000 -e 1e19 -t 89 --xs CSMS -s 7 --track
    radial
```

6.6. Sun

In addition to the Earth, `TauRunner` allows for propagation in the Sun. `TauRunner` includes high- and low-metallicity Suns, and a user may provide their own solar model. We include an example

```python
import numpy as np

from taurunner.main import run_MC
from taurunner.body.earth import construct_earth
from taurunner.cross_sections import CrossSections
from taurunner.utils import make_propagator, make_initial_e
    , make_initial_thetas

nevents        = 5000 # number of events to simulate
pid            = 16
xs_model       = "CSMS"
no_secondaries = True

Earth = construct_earth(layers=[(4., 1.0)])
xs    = CrossSections(xs_model)
rand  = np.random.RandomState(seed=7)

# Sample power-law with index -2 between 1e6 GeV and 1e12
    GeV
pl_exp   = -2 # power law exponent
e_min    = 1e15 # Minimum energy to sample in eV
e_max    = 1e21 # Maximum energy to sample in eV
energies = make_initial_e(nevents,
                          pl_exp,
                          e_min=e_max,
                          e_max=e_min,
                          rand=rand
                         )

# Sample uniform in solid angle over hemisphere
th_min = 0 # Minimum nadir angle to sample from
th_max = 90 # Maximum nadir angle to sample from
thetas = make_initial_thetas(nevents,
                             (th_min, th_max),
                             rand=rand
                            )

# tracks   = make_tracks(thetas)
tau_prop = make_propagator(pid, Earth)


output = run_MC(energies,
                thetas,
                Earth,
                xs,
                tau_prop,
                rand,
                no_secondaries=no_secondaries
               )
```

Listing 4: *Multiangle injection with energies drawn from a powerlaw distribution.* Example of propagating a flux of neutrinos with initial energies sampled from a power law with incidence angles uniformly sampled over a hemisphere.

of the form that these solar models should take in Appendix C (see Listing 7).

The same result may be achieved from the command line by running;

```
python main.py -n 1000 -e 2.4e17 -t 45 -s 7 --body HZ_Sun
    --xs dipole
```

6.7. Constant slab

The user may use the `radial` track to propagate neutrinos from a 'slab' of material of a constant density. This may be done by making a `Body` object on the fly in the manner shown in Listing 8.

6.8. Layered slab

The constant density slab may be generalized to a slab of multiple layers. As mentioned in Sec. 3.2, the densities in each layer may be positive scalars, unary functions which return positive scalars, or a potentially mixed list of such objects. In this example, we show how to accomplish this latter option (see Listing 9).

```python
import numpy as np

import taurunner as tr
from taurunner.main import run_MC
from taurunner.body.earth import construct_earth
from taurunner.cross_sections import CrossSections
from taurunner.utils import make_propagator, make_initial_e
    , make_initial_thetas

nevents  = 5000
pid      = 16
xs_model = "CSMS"

Earth    = construct_earth(layers=[(4., 1.0)])
xs       = CrossSections(xs_model)

tau_prop = make_propagator(pid, Earth)
rand     = np.random.RandomState(seed=7)

# Sample from pickled CDF
pkl_f    = f'{tr.__path__[0]}/resources/ahlers2010_test.pkl
    ' # Path to pickle file with CDF to sample from
energies = make_initial_e(nevents,
                          pkl_f,
                          rand=rand
)

# Sample uniform in solid angle over hemisphere
th_min = 0 # Minimum nadir angle to sample from
th_max = 90 # Maximum nadir angle to sample from
thetas = make_initial_thetas(nevents,
                             (th_min, th_max),
                             rand=rand
)

output = run_MC(energies,
                thetas,
                Earth,
                xs,
                tau_prop,
                rand
)
```

Listing 5: *Propagating a flux drawn from a provided cumulative distribution function (CDF).* Example of propagating ν_τ with energies drawn from a user-provided flux. TauRunner provides a few CDFs for the user, or custom CDFs may be built.

7. Conclusion

In this article, we have introduced a new package to propagate high-energy neutrinos in a variety of scenarios. Our implementation includes the dominant neutrino-propagation effects and is valid in the energy range of current and proposed neutrino telescopes. Additionally, in our performance section, we have compared our package with other state-of-the-art solutions to this problem and find them in good agreement where they overlap. Finally, the `TauRunner` package is designed to be extendable by the user, by either providing improved or altered physics inputs or constructing new geometries, giving the user the ability to extend the package functionality beyond the examples provided in this article. The authors hope that this work will encourage further development of publicly available physics software.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We acknowledge useful discussions with Joseph Farchione, Alfonso Garcia-Soto, Austin Lee Cummings, Andres Romero-Wolf, and Kareem Ramadan Hassan Aly Muhammad Farrag. We additionally thank Hallsie Reno, Sameer Patel, and Diksha Garg for insightful discussions on tau physics. We further thank Christopher Weaver for providing updated cross section tables and engaging discussions on non-trivial interpolation problems. We would also like to thank Gwenhaël de Wasseige for providing the solar models used in this work. Finally, we would like to give special acknowledgment to Francis Halzen for his support and discovery of tau regeneration, which was pivotal to this work. IS, JL, AP, and JV are supported by

```python
import numpy as np

from taurunner.main import run_MC
from taurunner.body.earth import construct_earth
from taurunner.cross_sections import CrossSections
from taurunner.utils import make_propagator, make_initial_e

nevents  = 5000
eini     = 1e19
pid      = 16
xs_model = "CSMS"

Earth    = construct_earth(layers=[(4., 1.0)]) # Make Earth
     object with 4km water layer
xs       = CrossSections(xs_model)
energies = make_initial_e(nevents, eini)
thetas   = np.zeros(nevents)

tau_prop = make_propagator(pid, Earth)
rand     = np.random.RandomState(seed=7)

output = run_MC(energies,
                thetas,
                Earth,
                xs,
                tau_prop,
                rand,
)
```

Listing 6: *Example of propagating ν_τ along a radial trajectory.* TauRunner allows for arbitrary particle trajectories. This example shows how to use the `radial` trajectory, whereas all previous examples have used the `chord` trajectory.

```python
import numpy as np
from taurunner.main import run_MC
from taurunner.body import construct_sun
from taurunner.cross_sections import CrossSections
from taurunner.utils import make_propagator, make_initial_e
    , make_initial_thetas, units

nevents     = 5000
eini        = 1e13 # the sun is opaque at high energies
theta       = 10.0
pid         = 16
xs_model    = "dipole"
solar_model = "HZ_Sun" # Can also be "LZ_Sun"

xs       = CrossSections(xs_model)
energies = make_initial_e(nevents, eini)
thetas   = make_initial_thetas(nevents, theta)

sun      = construct_sun(solar_model)
tau_prop = make_propagator(pid, sun, granularity=0.5)
rand     = np.random.RandomState(seed=7)

output = run_MC(energies,
                thetas,
                sun,
                xs,
                tau_prop,
                rand
)
```

Listing 7: *Propagating ν_τ through the Sun.* Example of how to propagate ν_τ through a body besides earth.

```python
import numpy as np

from taurunner.body import Body
from taurunner.main import run_MC
from taurunner.cross_sections import CrossSections
from taurunner.utils import make_propagator, make_initial_e
    , make_initial_thetas

nevents  = 5000
eini     = 1e15
theta    = 0
pid      = 14
xs_model = "CSMS"

# Make body with density 3.14 g/cm^3 and radius 1000 km
body     = Body(3.14, 1e3)

xs       = CrossSections(xs_model)
energies = make_initial_e(nevents, eini)
thetas   = make_initial_thetas(nevents, theta)

tau_prop = make_propagator(pid, body)
rand     = np.random.RandomState(seed=7)

output   = run_MC(energies,
                  thetas,
                  body,
                  xs,
                  tau_prop,
                  rand,
                  flavor=pid
)
```

Listing 8: *Propagation of ν_μ through a constant slab.* Although `TauRunner` only supports spherical bodies, we may use a body of constant density along with a `radial` trajectory to propagate a particle through a slab of constant density. One may create the slab from the base *Body* object or use the `body.slab` object. We do the former here for pedagogical purposes, but we recommend using the latter in practice since it has some computational speed ups.

```python
import numpy as np

from taurunner.body import Body
from taurunner.main import run_MC
from taurunner.cross_sections import CrossSections
from taurunner.utils import make_propagator, make_initial_e
    , make_initial_thetas

nevents  = 1000
eini     = 1e15
theta    = 0
pid      = 16
xs_model = "CSMS"

# Make layered body with radius 1,000 km
def density_f(x):
    return x**-2/4
densities  = [4, density_f, 1, 0.4]
boundaries = [0.25, 0.3, 0.5, 1] # Right hand boundaries of
     the layers last boundary should always be 1
body = Body([(d, b) for d, b in zip(densities, boundaries)
    ], 1e3)

xs       = CrossSections(xs_model)
energies = make_initial_e(nevents, eini)
thetas   = make_initial_thetas(nevents, theta)

tau_prop = make_propagator(pid, body)
rand     = np.random.RandomState(seed=7)

output = run_MC(energies,
                thetas,
                body,
                xs,
                tau_prop,
                rand
)
```

Listing 9: *Propagation of ν_τ through a layered slab.* We may employ the same strategy of using a `radial` trajectory to replicate propagation through a slab to propagate through a slab with varying properties.

NSF under grants PLR-1600823 and PHY-1607644 and by the University of Wisconsin Research Council with funds granted by the Wisconsin Alumni Research Foundation. OV acknowledges support by the Harvard College Research Program in the fall of 2020. CAA is supported by the Faculty of Arts and Sciences of Harvard University and the Alfred P. Sloan Foundation.

Appendix A. Constructing CDFs from which to sample

`TauRunner` offers the user the capability to provide custom spectra from which to sample initial energies. In this appendix, we describe the form in which `TauRunner` expects these spectra, and provide an example of constructing one. These should be `scipy.interpolate.UnivariateSpline` objects whose x-axis is the value of the cumulative density function of the spectra to sample and whose y-axis is the true neutrino energy in eV. We now provide an example of constructing these splines. The `.csv` file we use for this contains one column of energies in GeV and a corresponding column of the squared energies times the number density of the flux in units of GeV. It may be found at `resources/ahlers2010.csv` (see Listing 10).

Saving the file in resources is not necessary. The user may now sample from this distribution by passing the path to the file as the energy argument in the command line or as the first argument of the `make_initial_e` function seen in the examples. A more detailed example of constructing these splines in a `Jupyter Notebook` along with some sanity checks may be found on our GitHub in the `examples` folder.

Appendix B. Cross section splines

In this section we give an example of saving cross section splines in the form required by `TauRunner` so that the user may pass their own cross section model if they so choose. The differential splines should be a `scipy.interpolate.RectBivariateSpline` object and the total cross section splines should be a `scipy.interpolate.UnivariateSpline` object. We will now work out an example, assuming that we have two `.csv` files, one each for total and differential cross sections. In the former case, we will assume that it has two columns, the first containing neutrino energies and the second the corresponding total cross section. In the latter case, we will assume that we have three columns, the first containing an incoming neutrino energy, the second containing convenience variable described in Sec. 3.4, and the third containing the corresponding differential cross section. All energy units will be assumed to be GeV and all area units cm^2. In the case of the differential cross section, the values of the convenience variable must be the same for each incoming neutrino energy. As a reminder, `TauRunner` assumes that the cross section is the same for all neutrino flavors and thus the user need make only one set of cross section splines (see Listing 11).

This process would then be repeated for all combinations of interaction type $\in$ [`"CC"`, `"NC"`], neutrino type $\in$ [`"nu"`, `"nubar"`], and nucleon $\in$ [`"p"`, `"n"`] for a total of 8 splines. Now we show a similar example for constructing differential cross section splines. `TauRunner` splines have support down to 1 GeV, and this number is used internally. While it is not strictly necessary to have support down to this energy, it is possible that `TauRunner`

```python
import numpy as np
from scipy.integrate import quad
from scipy.interpolate import UnivariateSpline
import pickle

import taurunner as tr
from taurunner.utils import units

# csv of a benchmark GZK flux
infile = f'{tr.__path__[0]}/resources/ahlers2010.csv'
tab_data = np.genfromtxt(infile, delimiter = ',')
gzk_e    = tab_data[0]*units.GeV # Convert energies to eV
gzk_dnde = tab_data[0]*units.GeV

gzk_mine = gzk_en[0]
gzk_maxe = gzk_en[-1]

# Splining in logspace recommended
gzk_spline = UnivariateSpline(np.log(gzk_en), np.log(
    gzk_flux/gzk_en**2), k = 4, s=1e-2)

integrand = lambda E: np.exp(gzk_spline(np.log(E)))

# integrating in logspace also recommended
norm, _ = quad(lambda x: np.exp(x)*integrand(np.exp(x)), np
    .log(gzk_min), np.log(gzk_max))

pdf = lambda E: integrand(E) / norm

# Make and spline CDF
cdf_energies = np.logspace(np.log10(gzk_min), np.log10(
    gzk_max*1.1), 500) # Maybe more knots than necessary
    but more support is better
cdf = np.array([integrate.quad(lambda x: np.exp(x)*
    probability(np.exp(x)), np.log(gzk_min), np.log(y))[0]
     for y in cdf_energies])
# Make sure this in invertible
mask = np.where(np.logical_and(cdf>0, cdf<=1))[0]
cdf = cdf[mask]
cdf_energies = cdf_energies[mask]
cdf_spl = UnivariateSpline(cdf, cdf_energies)

# Save the spline as a pickle file
out_f = f'{tr.__path__[0]}/resources/ahlers2010.pkl'
with open(out_f, 'wb') as pkl_f:
    pkl.dump(cdf_spl, pkl_f)
```

Listing 10: Constructing custom flux files in the format required by `TauRunner`.

may evaluate the splines in this regime, and thus understanding the behavior of splines in this regime is recommended.

```python
import numpy as np
from scipy.interpolate import RectBivariateSpline
import pickle

import taurunner as tr
from taurunner.utils import units

model_name  = "my_model"
interaction = "NC" # Neutral current
nucleon     = "n" # neutron
nutype      = "nu" # neutrino

# csv containing the neutrino neutron NC dsigma/de
tot_xs_path = f"/path/to/{nutype}_{nucleon}_{interaction}
    _dsde.csv"

e_in = np.genfromtxt(tot_xs_path, delimiter=",")[0]
z    = np.genfromtxt(tot_xs_path, delimiter=",")[1]
dsde = np.genfromtxt(tot_xs_path, delimiter=",")[2]

# Convert to natural units
e_in  = e_in*units.GeV
dsde  = dsde*units.cm**2/units.GeV
dsdx  = dsde*e_in

# Spline in logspace
```

```python
import numpy as np
from scipy.interpolate import UnivariateSpline
import pickle

import taurunner as tr
from taurunner.utils import units

model_name  = "my_model"
interaction = "CC" # Charged current
nucleon     = "p" # proton
nutype      = "nubar" # antineutrino

# csv containing the anti-neutrino proton CC xs
tot_xs_path = f"/path/to/{nutype}_{nucleon}_{interaction}
    _xs.csv"
e = np.genfromtxt(tot_xs_path, delimiter=",")[0]
xs = np.genfromtxt(tot_xs_path, delimiter=",")[1]

# Convert to natural units
e = e*units.GeV
xs = xs*units.cm**2

# Spline in logspace
xs_spl = UnivariateSpline(np.log(e), np.log(xs))

# Save the spline as a pickle file
# Splines must follow this naming convention and be in this
    directory
out_f = f"{tr.__path__[0]}/resources/cross_section_tables/{
    model_name}_{nutype}_{nucleon}_sigma_{interaction}.pkl
    "
with open(out_f, "wb") as pkl_f:
    pkl.dump(xs_spl, pkl_f)
```

Listing 11: Example of constructing differential cross section splines for `TauRunner`.

```python
xs_spl = RectBivariateSpline(np.log(np.unique(e_in)), z, np
    .log(dsdx))

# Save the spline as a pickle file
# Splines must follow this naming convention and be in this
    directory
out_f = f"{tr.__path__[0]}/resources/cross_section_tables/{
    model_name}_{nutype}_{nucleon}_dsde_{interaction}.pkl"
with open(out_f, "wb") as pkl_f:
    pkl.dump(xs_spl, pkl_f)
```

As in the case of the total cross section, this process must be repeated for all combinations of interaction type $\in$ [`"CC"`, `"NC"`], neutrino type $\in$ [`"nu"`, `"nubar"`], and nucleon $\in$ [`"p"`, `"n"`] for a total of 8 splines. This new model may then be used by passing `"my_model"` when initializing the `CrossSection` object.

Appendix C. Solar model format

`TauRunner` expects solar models to have at minimum three columns, one containing the radius in units of the solar radius, one containing the corresponding mass density in g/cm^3, and the last containing the corresponding electron density in N_A^{-1}cm^{-3}. These values should not be comma separated and lines beginning with # will be ignored as comments. Any additional columns will be ignored by `TauRunner`, allowing the user to add additional columns if it is useful, for *e.g.* a column containing the proton fraction to pass to the body.

References

[1] E. Vitagliano, I. Tamborra, G. Raffelt, Rev. Mod. Phys. 92 (2020) 45006, https://doi.org/10.1103/RevModPhys.92.045006, arXiv:1910.11878.

[2] J.A. Formaggio, G.P. Zeller, Rev. Mod. Phys. 84 (2012) 1307–1341, https://doi.org/10.1103/RevModPhys.84.1307, arXiv:1305.7513.

[3] G. Barenboim, P.B. Denton, S.J. Parke, C.A. Ternes, Phys. Lett. B 791 (2019) 351–360, https://doi.org/10.1016/j.physletb.2019.03.002, arXiv:1902.00517.

[4] C.A. Argüelles, J. Salvado, C.N. Weaver, Comput. Phys. Commun. 255 (2020) 107405, https://doi.org/10.1016/j.cpc.2020.107405.

[5] P. Huber, J. Kopp, M. Lindner, M. Rolinec, W. Winter, Comput. Phys. Commun. 177 (2007) 432–438, https://doi.org/10.1016/j.cpc.2007.05.004, arXiv:hep-ph/0701187.
[6] R.G. Calland, A.C. Kaboth, D. Payne, J. Instrum. 9 (2014) P04016, https://doi.org/10.1088/1748-0221/9/04/P04016, arXiv:1311.7579.
[7] M. Wallraff, C. Wiebusch, Calculation of oscillation probabilities of atmospheric neutrinos using nuCraft, arXiv:1409.1387, 2014.
[8] C.A. Argüelles, B.J.P. Jones, Phys. Rev. Res. 1 (2019) 033176, https://doi.org/10.1103/PhysRevResearch.1.033176, arXiv:1904.10559.
[9] Prob3++ software for computing three flavor neutrino oscillation probabilities, http://www.phy.duke.edu/~raw22/public/Prob3++/, 2012.
[10] C.A. Argüelles, J. Salvado, C.N. Weaver, nuSQuIDS, https://github.com/arguelles/nuSQuIDS, 2015.
[11] M.G. Aartsen, et al., J. Instrum. 12 (03) (2017) P03012, https://doi.org/10.1088/1748-0221/12/03/P03012, arXiv:1612.05093.
[12] R. Gandhi, C. Quigg, M.H. Reno, I. Sarcevic, Phys. Rev. D 58 (1998) 093009, https://doi.org/10.1103/PhysRevD.58.093009, arXiv:hep-ph/9807264.
[13] A. Gazizov, M.P. Kowalski, Comput. Phys. Commun. 172 (2005) 203–213, https://doi.org/10.1016/j.cpc.2005.03.113, arXiv:astro-ph/0406439.
[14] T.R. De Young, IceTray: a software framework for IceCube, https://doi.org/10.5170/CERN-2005-002.463, http://cds.cern.ch/record/865626, 2005.
[15] S. Yoshida, R. Ishibashi, H. Miyamoto, Phys. Rev. D 69 (2004) 103004, https://doi.org/10.1103/PhysRevD.69.103004, arXiv:astro-ph/0312078.
[16] A.C. Vincent, C.A. Argüelles, A. Kheirandish, J. Cosmol. Astropart. Phys. 11 (2017) 012, https://doi.org/10.1088/1475-7516/2017/11/012, arXiv:1706.09895.
[17] S. Yoshida, M. Meier, ShigeruYoshida/JULIeT: first official release with a DOI, https://doi.org/10.5281/zenodo.4018117, 2020.
[18] S. Ritz, D. Seckel, Nucl. Phys. B 304 (1988) 877–908, https://doi.org/10.1016/0550-3213(88)90660-8.
[19] F. Halzen, D. Saltzberg, Phys. Rev. Lett. 81 (1998) 4305–4308, https://doi.org/10.1103/PhysRevLett.81.4305, arXiv:hep-ph/9804354.
[20] M.G. Aartsen, et al., J. Phys. G 48 (6) (2021) 060501, https://doi.org/10.1088/1361-6471/abbd48, arXiv:2008.04323.
[21] A.V. Olinto, et al., J. Cosmol. Astropart. Phys. 06 (2021) 007, https://doi.org/10.1088/1475-7516/2021/06/007, arXiv:2012.07945.
[22] A. Garcia, R. Gauld, A. Heijboer, J. Rojo, J. Cosmol. Astropart. Phys. 09 (2020) 025, https://doi.org/10.1088/1475-7516/2020/09/025, arXiv:2004.04756.
[23] S. Adrian-Martinez, et al., J. Phys. G 43 (8) (2016) 084001, https://doi.org/10.1088/0954-3899/43/8/084001, arXiv:1601.07459.
[24] I. Safa, A. Pizzuto, C.A. Argüelles, F. Halzen, R. Hussain, A. Kheirandish, J. Vandenbroucke, J. Cosmol. Astropart. Phys. 01 (2020) 012, https://doi.org/10.1088/1475-7516/2020/01/012, arXiv:1909.10487.
[25] O. Vazquez, I. Safa, J. Lazar, A. Pizzuto, C. Arguelles, A. Kheirandish, J. Vandenbroucke, PoS ICRC2021 (2021) 1030, https://doi.org/10.22323/1.395.1030.
[26] P.W. Gorham, et al., Phys. Rev. Lett. 117 (7) (2016) 071101, https://doi.org/10.1103/PhysRevLett.117.071101, arXiv:1603.05218.
[27] P.W. Gorham, et al., Phys. Rev. Lett. 121 (16) (2018) 161102, https://doi.org/10.1103/PhysRevLett.121.161102, arXiv:1803.05088.
[28] R. Abbasi, et al., PoS ICRC2021 (2021) 1170, https://doi.org/10.22323/1.395.1170.
[29] M. Agostini, et al., Nat. Astron. 4 (10) (2020) 913–915, https://doi.org/10.1038/s41550-020-1182-4, arXiv:2005.09493.
[30] M. Sasaki, T. Kifune, JPS Conf. Proc. 15 (2017) 011013, https://doi.org/10.7566/JPSCP.15.011013.
[31] A. Romero-Wolf, et al., in: Latin American Strategy Forum for Research Infrastructure, 2020, arXiv:2002.06475.
[32] S. Patel, et al., PoS ICRC2021 (2021) 1203, https://doi.org/10.22323/1.395.1203, arXiv:2109.08198.
[33] J. Alvarez-Muñiz, W.R. Carvalho, A.L. Cummings, K. Payet, A. Romero-Wolf, H. Schoorlemmer, E. Zas, Phys. Rev. D 97 (2) (2018) 023021, https://doi.org/10.1103/PhysRevD.97.023021, Erratum: Phys. Rev. D 99 (2019) 069902, arXiv:1707.00334.
[34] M.C. Gonzalez-Garcia, F. Halzen, M. Maltoni, Phys. Rev. D 71 (2005) 093010, https://doi.org/10.1103/PhysRevD.71.093010, arXiv:hep-ph/0502223.
[35] J.F. Beacom, P. Crotty, E.W. Kolb, Phys. Rev. D 66 (2002) 021302, https://doi.org/10.1103/PhysRevD.66.021302, arXiv:astro-ph/0111482.
[36] J.H. Koehne, K. Frantzen, M. Schmitz, T. Fuchs, W. Rhode, D. Chirkin, J. Becker, Comput. Phys. Commun. 184 (2013) 2070–2090, https://doi.org/10.1016/j.cpc.2013.04.001.
[37] S.I. Dutta, M.H. Reno, I. Sarcevic, Phys. Rev. D 66 (2002) 077302, https://doi.org/10.1103/PhysRevD.66.077302, arXiv:hep-ph/0207344.
[38] P. Zyla, et al., Rev. Part. Phys. PTEP 2020 (8) (2020) 083C01, https://doi.org/10.1093/ptep/ptaa104.
[39] A.A. Aguilar-Arevalo, et al., Phys. Rev. D 81 (2010) 092005, https://doi.org/10.1103/PhysRevD.81.092005, arXiv:1002.2680.
[40] M. Tzanov, et al., Phys. Rev. D 74 (2006) 012008, https://doi.org/10.1103/PhysRevD.74.012008, arXiv:hep-ex/0509010.
[41] P. Vogel, J.F. Beacom, Phys. Rev. D 60 (1999) 053003, https://doi.org/10.1103/PhysRevD.60.053003, arXiv:hep-ph/9903554.
[42] A. Kurylov, M.J. Ramsey-Musolf, P. Vogel, Phys. Rev. C 67 (2003) 035502, https://doi.org/10.1103/PhysRevC.67.035502, arXiv:hep-ph/0211306.
[43] M. Agostini, et al., Nature 562 (7728) (2018) 505–510, https://doi.org/10.1038/s41586-018-0624-y.
[44] Z. Li, et al., Phys. Rev. D 98 (5) (2018) 052006, https://doi.org/10.1103/PhysRevD.98.052006, arXiv:1711.09436.
[45] M.G. Aartsen, et al., Nature 551 (2017) 596–600, https://doi.org/10.1038/nature24459, arXiv:1711.08119.
[46] R. Abbasi, et al., arXiv:2011.03560, https://doi.org/10.1103/PhysRevD.104.022001.
[47] Amanda Cooper-Sarkar, Philipp Mertsch, Subir Sarkar, J. High Energy Phys. 08 (2011) 042, https://doi.org/10.1007/JHEP08(2011)042, arXiv:1106.3723.
[48] M. Froissart, Phys. Rev. 123 (1961) 1053–1057, https://doi.org/10.1103/PhysRev.123.1053.
[49] C.A. Argüelles, F. Halzen, L. Wille, M. Kroll, M.H. Reno, Phys. Rev. D 92 (7) (2015) 074040, https://doi.org/10.1103/PhysRevD.92.074040, arXiv:1504.06639.
[50] M.M. Block, F. Halzen, Phys. Rev. Lett. 107 (2011) 212002, https://doi.org/10.1103/PhysRevLett.107.212002, arXiv:1109.2041.
[51] L. Montanet, et al., Phys. Rev. D 50 (1994) 1173–1823, https://doi.org/10.1103/PhysRevD.50.1173.
[52] A.M. Dziewonski, D.L. Anderson, Phys. Earth Planet. Inter. 25 (1981) 297–356, https://doi.org/10.1016/0031-9201(81)90046-7.
[53] G. de Wasseige, personal communication, August 2020.
[54] M.M. Block, L. Durand, P. Ha, Phys. Rev. D 89 (2014) 094027.
[55] A. Neronov, D.V. Semikoz, L.A. Anchordoqui, J. Adams, A.V. Olinto, Phys. Rev. D 95 (2) (2017) 023004, https://doi.org/10.1103/PhysRevD.95.023004, arXiv:1606.03629.
[56] J. Álvarez-Mu niz, et al., Sci. China, Phys. Mech. Astron. 63 (1) (2020) 219501, https://doi.org/10.1007/s11433-018-9385-7, arXiv:1810.09994.
[57] J.A. Aguilar, et al., The next-generation radio neutrino observatory – multi-messenger neutrino astrophysics at extreme energies, arXiv:1907.12526, 2019.
[58] S. van der Walt, S.C. Colbert, G. Varoquaux, Comput. Sci. Eng. 13 (2) (2011) 22–30, https://doi.org/10.1109/MCSE.2011.37, arXiv:1102.1523.
[59] P. Virtanen, et al., Nat. Methods 17 (2020) 261, https://doi.org/10.1038/s41592-019-0686-2, arXiv:1907.10121.
[60] M. Ahlers, L.A. Anchordoqui, M.C. Gonzalez-Garcia, F. Halzen, S. Sarkar, Astropart. Phys. 34 (2010) 106–115, https://doi.org/10.1016/j.astropartphys.2010.06.003, arXiv:1005.2620.