Hazma Documentation

Release 1.2

Logan A. Morrison et. al.

Feb 19, 2018

INTRODUCTION AND DETAILS

CHAPTER ONE

INTRODUCTION

1.1 Breif Description

This package is used to compute the gamma ray spectra $\frac{dN}{dE}$ for light particles, such as, pions, kaon, electrons and muons, in an energy regime where the mass effects are important, i.e. is the MeV energy range. The code has been written in python/cython.

1.2 Installation

To install Hazma, clone the package from [https://github.com/LoganAMorrison/Hazma.git{]}(https://github.com/LoganAMorrison/Hazma.git):

```
git clone https://github.com/LoganAMorrison/Hazma.git
```

Once cloned, move into the `Hazma` directory and install using `pip`:

```
cd Hazma
sudo pip install .
```

CHAPTER

TWO

GAMMA RAY SPECTRA GENERATOR (HAZMA.GAMMA_RAY)

2.1 Description

Sub-package for generating gamma ray spectra given a multi-particle final state.

Hazma includes two different methods for computing gamma ray spectra. The first is done by specifying the final state of a process. Doing so, the particle decay spectra are then computed. The second method `gamma_ray_rambo` takes in the tree-level and radiative squared matrix elements and runs a Monte-Carlo to generate the gamma ray spectra.

2.2 Functions

`hazma.gamma_ray` has the following functions

Semi-Analytic Spectrum Generator	*hazma.gamma_ray.gamma_ray*
Monte Carlo Spectrum Generator	*hazma.gamma_ray.gamma_ray_rambo*

CHAPTER

THREE

RAMBO (HAZMA.RAMBO)

- Author - Logan A. Morrison and Adam Coogan
- Date - December 2017

3.1 Functions

Computing annihilation cross sections	*hazma.rambo.compute_annihilation_cross_section*
Computing decay widths	*hazma.rambo.compute_decay_width*
Computing energy histograms	*hazma.rambo.generate_energy_histogram*
Computing a single relativistic phase space point	*hazma.rambo.generate_phase_space_point*
Computing many relativistic phase space points	*hazma.rambo.generate_phase_space*

3.2 Description

Sub-package for generating phases space points and computing phase space integrals using a Monte Carlo algorithm called RAMBO. This algorithm was originally implemented by [1].

This algorithm starts by generating random, massless four-momenta q_i with energies distributed according to $q_0 \exp(-q_0)$. These q_i's are then transformed into p_i's by boosting the q_i's into the center-of-mass frame so that $\sum_i p_i^\mu = (E_{\text{CM}}, 0, 0, 0)$. Lastly, the p_i's are rescaled into k_i's so that the k_i's give the correct masses.

3.3 Details

The RAMBO algorithm can be broken up into four steps:

- Randomly generate n massless four-momenta: $\{q_i\}$.
- Boost the set $\{q_i\}$ into $\{p_i\}$.
- Transform the set $\{p_i\}$ into $\{k_i\}$.
- Compute the weight of the phase space point.

Repeating these steps many times will generate the phase space.

3.3.1 Generating the $\{q_i\}$

The first step is to generate the set $\{q_i\}$, where $i \in \{1, \dots, n\}$ and n is the number of final state particles. The q_i's will have energies distributed according to $q_0 \exp(-q_0)$ and will be massless. Additionally, the q_i's will be unphysical, i.e. will not have the correct center-of-mass energy. To generate the $\{q_i\}$, we first compute $4n$ uniform random numbers, $\rho_1^i, \rho_2^i, \rho_3^i, \rho_4^i$, on the interval $(0, 1)$. Then, we compute the following:

$$c^i = 2\rho_1^i - 1.0 \qquad \phi^i = 2\pi\rho_2^i$$

Then, the components of q_i will be:

$$\begin{aligned} q_i^0 &= -\log(\rho_3\rho_4) \\ q_i^x &= q_i^0\sqrt{1-(c^i)^2}\cos\phi^i \\ q_i^y &= q_i^0\sqrt{1-(c^i)^2}\sin\phi^i \\ q_i^z &= q_i^0 c^i \end{aligned}$$

3.3.2 Generating the $\{p_i\}$

Next, the $\{q_i\}$'s need to be boosted to generate the $\{p_i\}$'s. This is done using the following:

$$\begin{aligned} p_i^0 &= x(\gamma q_i^0 + \vec{b}\cdot\vec{q}_i) \\ \vec{p}_i &= x\left(\vec{q}_i^0 + q_i^0\vec{b} + a\left(\vec{b}\cdot\vec{q}_i\right)\vec{b}\right) \end{aligned}$$

where

$$\vec{b} = -\vec{Q}/M, \quad x = E_{\text{CM}}/M, \quad a = \frac{1}{1+\gamma}$$

Here, Q is a four-vector equal to the sum of the q_i's, $M = \sqrt{Q^2}$ and $\gamma = Q^0/M$ is the relativistic boost factor.

3.3.3 Generating the $\{k_i\}$

At this point, we have the $\{p_i\}$'s, which are four-momenta with the correct center of mass energy, but which are still massless. Our next, step is to transform the $\{p_i\}$'s into $\{k_i\}$'s, which have the correct masses.

In order to get the correct masses, we need to rescale the $\{p_i\}$'s. This is done as follows:

$$\begin{aligned} \vec{k}_i &= \xi\vec{p}_i \\ k_i^0 &= \sqrt{m_i^2 + \xi^2\left(p_i^0\right)^2} \end{aligned}$$

where ξ is the scaling factor. To compute the scaling factor, we require that the sum of the k_i^0's gives the correct center-of-mass energy. That is, that ξ satisfies:

$$E_{\text{CM}} = \sum_{i=1}^{n}\sqrt{m_i^2 + \xi^2\left(p_i^0\right)^2}$$

Thus, to compute the $\{k_i\}$'s, we solve for ξ an then rescale the $\{p_i\}$'s according to the above transformations.

3.3.4 Compute the phase space weight

The last step is to compute the phase space weight. The weights are such that, when summing over all phase space points, weighted by the weights, we get the correct phase space volume. The phase space weight is given by

$$W = E_{\text{CM}}^{2n-3}\left(\sum_{i=1}^{n}|\vec{k}_i|\right)^{2n-3}\left(\prod_{i=1}^{n}\frac{|\vec{k}_i|}{k_i^0}\right)\left(\sum_{i=1}^{n}\frac{|\vec{k}_i|^2}{k_i^0}\right)^{-1}\frac{(\pi/2)^{n-1}(2\pi)^{4-3n}}{\Gamma(n)\Gamma(n-1)}$$

CHAPTER

FOUR

DECAY (HAZMA.DECAY)

In this section, we describe how the radiative decay spectra are computed for the muon, charged pion and neutral pion.

4.1 Functions

The submodule `hazma.decay` has the following functions:

Muon	*hazma.decay.muon*
Neutral Pion	*hazma.decay.neutral_pion*
Charged Pion	*hazma.decay.charged_pion*
Short Kaon	*hazma.decay.short_kaon*
Long Kaon	*hazma.decay.long_kaon*
Charged Kaon	decay_charged_kaon

4.2 Details

4.2.1 Muon

The dominant contribution to the radiative decay of the muon comes from $\mu^{\pm} \to e^{\pm}\nu\bar{\nu}\gamma$. The unpolarized differential branching fraction of this decay mode in the *muon rest frame* can be written as [1]

$$\frac{dB}{dy\, d\cos\theta_{\gamma}^{R}} = \frac{1}{y}\frac{\alpha}{72\pi}(1-y)\left[12\left(3-2y(1-y)^2\right)\log\left(\frac{1-y}{r}\right) + y(1-y)(46-55y) - 102\right]$$

where $r = (m_e/m_\mu)^2$, $0 \leq y = 2E_{\gamma}^{R\mu}/m_\mu \leq 1-r$, ($E_{\gamma}^{R\mu}$ is the energy of the photon in the muon rest frame) and θ_{γ}^{R} is the angle the photon makes with respect to some axis in the muon rest frame. In order to obtain the decay spectrum in the laboratory frame, we need to boost the above spectrum. In other words, we need to change variables from the gamma ray energy and angle in the muon rest frame to those in the lab frame. We then integrate over the angle to compute dN/dE_γ. The Jacobian for this change of variables is

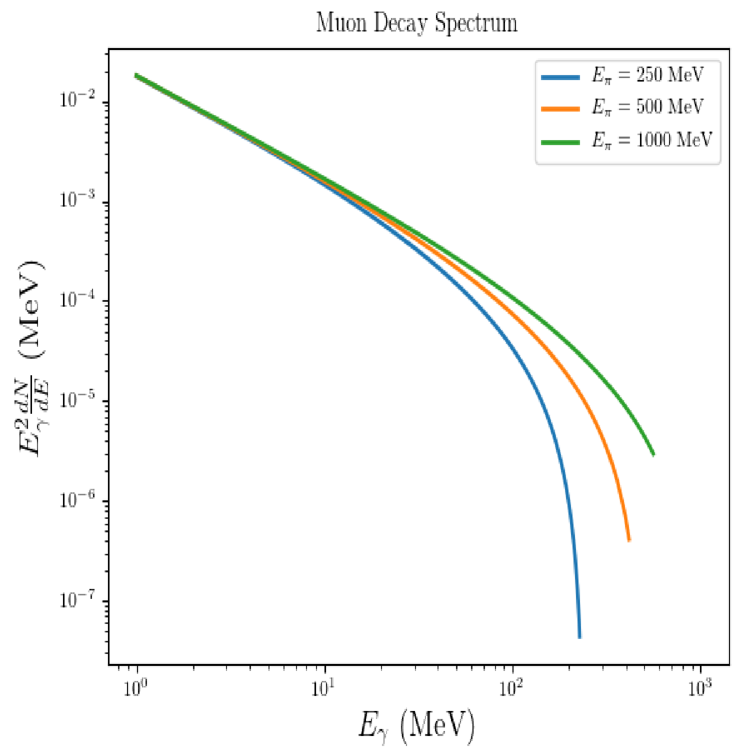
$$J = \frac{1}{2\gamma(1-\beta\cos\theta_{\gamma}^{L})}$$

where the boost parameters are

$$\gamma = E_\mu/m_\mu, \qquad \beta = \sqrt{1-\left(\frac{m_\mu}{E_\mu}\right)^2}$$

Integrating over angles yields the gamma ray spectrum in the lab frame:

$$\frac{dN}{dE_\gamma^L} = \int_{-1}^{1} d\cos\theta_\gamma^L \frac{1}{2\gamma(1-\beta\cos\theta_\gamma^L)} \frac{dB}{dE_\gamma^{R\mu}}$$



4.2.2 Charged Pion

To compute the gamma ray spectrum from a charged pion, one considers to possible decay modes. These decay modes are $\pi^\pm \to \mu^\pm\nu_\mu\gamma$ and $\pi^\pm \to \mu^\pm\nu_\mu \to e^\pm\nu_\mu\nu_\mu\nu_e\gamma$. To compute the gamma ray spectrum from the first decay mode, one uses results from [2]. It turns out that the spectrum from this decay mode is roughly a factor of 100 times smaller than the spectrum from the second decay mode. We thus ignore the contributions from $\pi^\pm \to \mu^\pm\nu_\mu\gamma$.

To compute the γ-ray spectrum from $\pi^\pm \to \mu^\pm\nu_\mu \to e^\pm\nu_\mu\nu_\mu\nu_e\gamma$, we first take the muon decay spectra (see section on muon decay spectra) and boost the muon into the pion rest frame use the following:

$$\gamma_1 = E_{R\mu}/m_\mu \qquad \beta_1 = \sqrt{1-\left(\frac{m_\mu}{E_{R\mu}}\right)^2} \qquad E_{R\mu} = \frac{m_\pi^2 - m_\mu^2}{m_\pi^2 + m_\mu^2}$$

where $E_{R\mu}$ is the energy of the muon in the pion rest frame. The photon spectrum in the charged pion rest frame, $dN/dE_\gamma^{R\pi}$, is obtain by integrating the differential branching ratio times a Jacobian factor $1/2\gamma_1(1-\beta_1\cos\theta_\gamma^{R\pi})$ over the angle the photon makes with the muon. Once this integration is completed, one then boosts into the laboratory frame of reference. The steps are nearly identical to boosting from the muon rest frame to the pion rest frame. The only thing that changes in the boost factor and the Jacobian. In going from the charged pion rest frame to the laboratory frame, the Jacobian and boost factor are

$$J = \frac{1}{2\gamma_2(1-\beta_2\cos\theta_\gamma^L)} \qquad \gamma_2 = E_\pi/m_\pi \qquad \beta_2 = \sqrt{1-\left(\frac{m_\mu}{E_\pi}\right)^2}$$

The gamma-ray spectrum in the laboratory frame will thus be

$$\frac{dN}{dE_{\gamma}^{L}} = \int_{-1}^{1} d\cos\theta_{\gamma}^{L} \frac{1}{2\gamma_2(1-\beta_2\cos\theta_{\gamma}^{L})} \times \left(\int_{-1}^{1} d\cos\theta_{\gamma}^{R\pi} \frac{1}{2\gamma_1(1-\beta_1\cos\theta_{\gamma}^{L})} \frac{dB}{dE_{\gamma}^{R\mu}} \right)$$

where

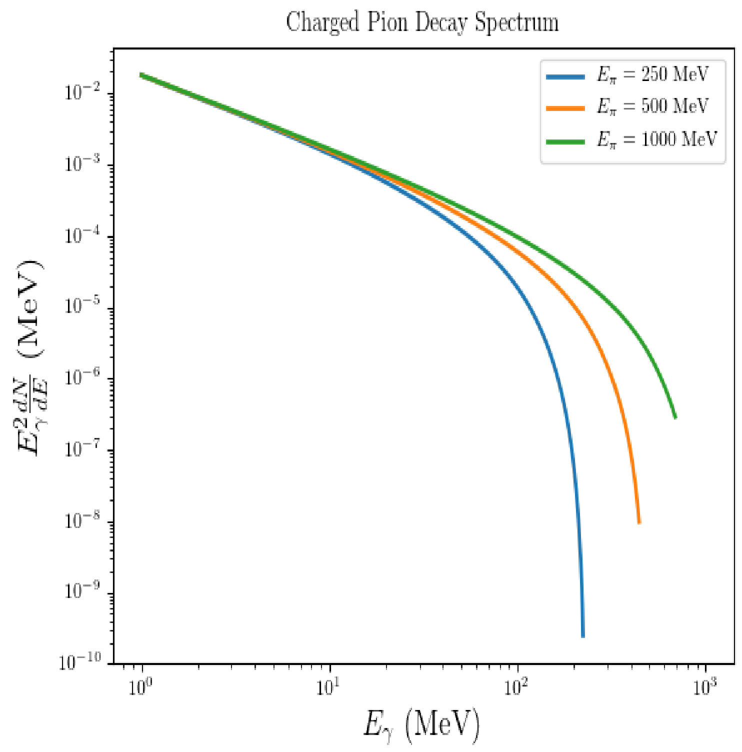
$$E_{\gamma}^{R\mu} = \gamma_1 E_{\gamma}^{R\pi} \left(1 - \beta_1 \cos\theta_{\gamma}^{R\pi}\right)$$

and

$$E_{\gamma}^{R\pi} = \gamma_2 E_{\gamma}^{L} \left(1 - \beta_2 \cos\theta_{\gamma}^{L}\right)$$

The limits on the photon energy are given by

$$0 \leq E_{\gamma}^{L} \leq \frac{m_{\mu}^2 - m_e^2}{2m_{\mu}} \gamma_1\gamma_2(1+\beta_1)(1+\beta_2)$$



4.2.3 Neutral Pion

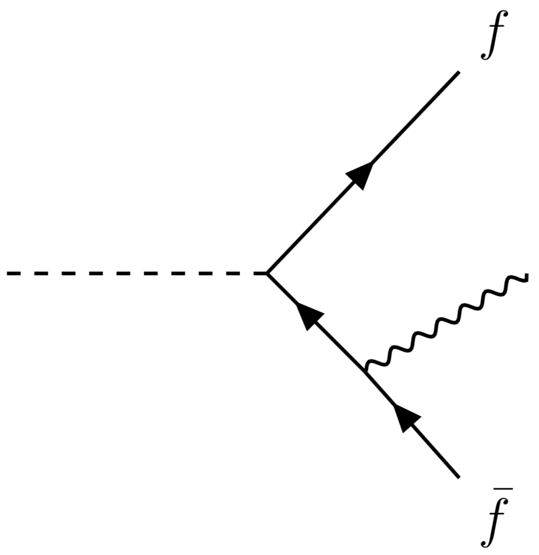
The dominant decay mode of the neutral pion is $\pi^0 \rightarrow \gamma\gamma$. In the laboratory frame, the spectrum is

$$\frac{dN}{dE_{\gamma}} = \frac{2}{m_{\pi}\gamma\beta}$$

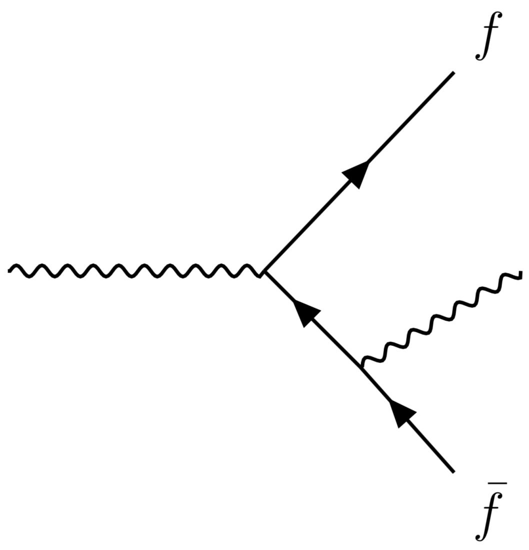
4.2.4 References

FINAL STATE RADIATION

Along with computing decay spectra, hazma is able to compute final state radiation spectra from decays of off-shell mediators (scalar, psuedo-scalar, vector or axial-vector.) The relavent diagrams for such processes are

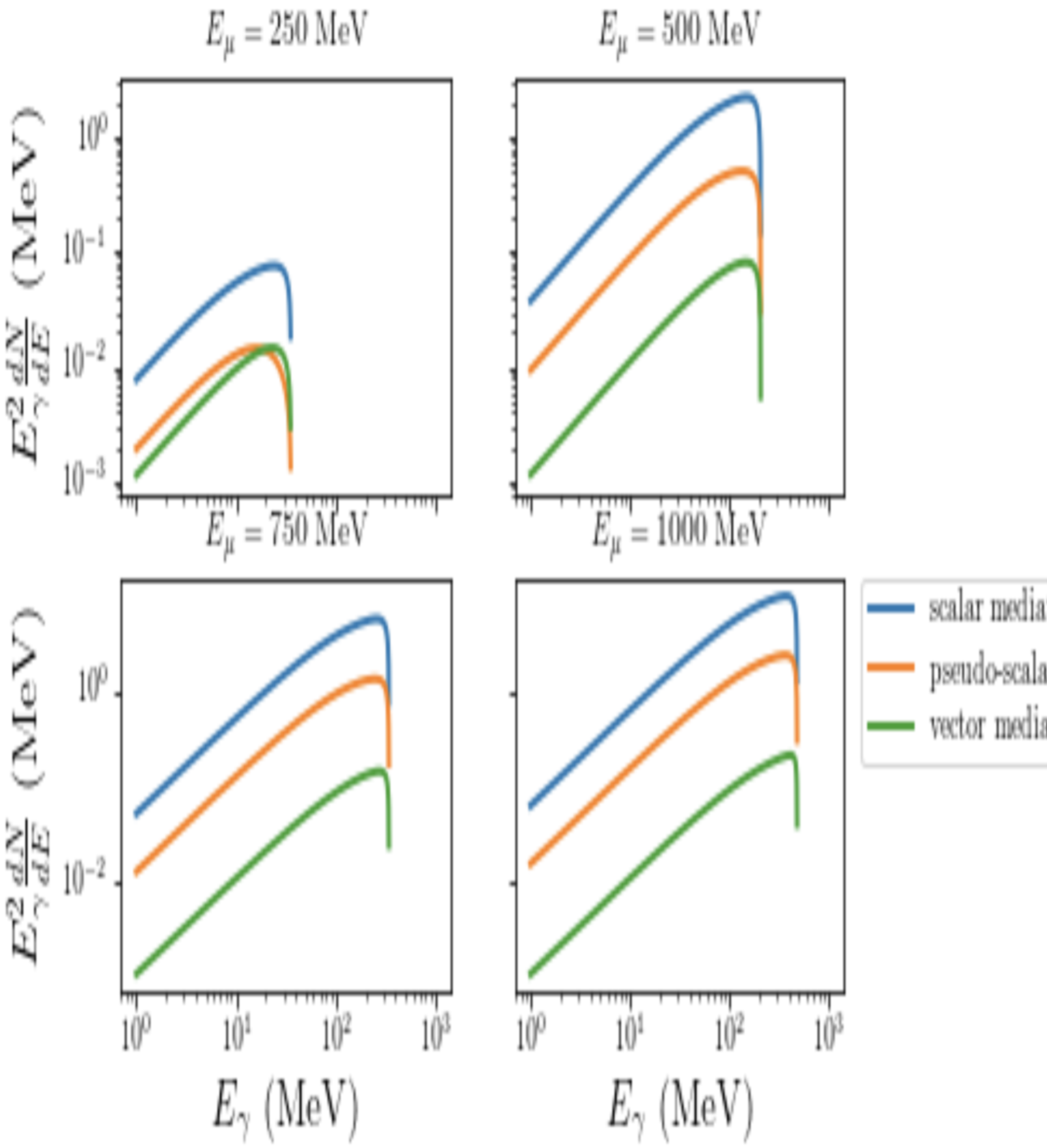


1. Scalar mediator



2. Vector mediator

Computing the matrix elements squared of these diagrams (including diagrams with the photon attached to the other fermion leg) and integrating over all variables except the photon energy yields $d\sigma(M^* \to \mu^+\mu^-\gamma)/dE$. To compute dN/dE, we divide $d\sigma(M^* \to \mu^+\mu^-\gamma)/dE$ by $\sigma(M^* \to \mu^+\mu^-)$.


$E_\mu = 250$ MeV
$E_\mu = 500$ MeV
$E_\mu = 750$ MeV
$E_\mu = 1000$ MeV
$E_\gamma^2 \frac{dN}{dE}$ (MeV)
E_γ (MeV)
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{1}
10^{2}
10^{3}
scalar media
pseudo-scala
vector media

CHAPTER SIX

TUTORIAL

Will update soon!

CHAPTER

SEVEN

HAZMA.GAMMA_RAY.GAMMA_RAY

hazma.gamma_ray.**gamma_ray**(*particles*, *cme*, *eng_gams*, *mat_elem_sqrd=<function <lambda>>*, *num_ps_pts=1000*, *num_bins=25*)

Returns total gamma ray spectrum from a set of particles.

Blah and blah.

Parameters

- **particles** (*array_like*) – List of particle names. Availible particles are 'muon', 'electron' 'charged_pion', 'neutral pion', 'charged_kaon', 'long_kaon', 'short_kaon'.
- **cme** (*double*) – Center of mass energy of the final state in MeV.
- **eng_gams** (*np.ndarray[double, ndim=1]*) – List of gamma ray energies in MeV to evaluate spectra at.
- **mat_elem_sqrd** (double(*func)(np.ndarray,)) – Function for the matrix element squared of the proccess. Must be a function taking in a list of four momenta of size (num_fsp, 4). Default value is a flat matrix element.
- **num_ps_pts** (*int {1000}, optional*) – Number of phase space points to use.
- **num_bins** (*int {25}, optional*) – Number of bins to use.

Returns **spec** (*np.ndarray*) – Total gamma ray spectrum from all final state particles.

Notes

The total spectrum is computed using

$$\frac{dN}{dE}(E_\gamma) = \sum_{i,j} P_i(E_j) \frac{dN_i}{dE}(E_\gamma, E_j)$$

where i runs over the final state particles, j runs over energies sampled from probability distributions. $P_i(E_j)$ is the probability that particle i has energy E_j. The probabilities are computed using *hazma.phase_space_generator.rambo*. The total number of energies used is *num_bins*.

Examples

Example of generating a spectrum from a muon, charged kaon and long kaon with total energy of 5000 MeV.

```
>>> from hazma.gamma_ray import gamma_ray
>>> import numpy as np
>>>
```

```
>>> particles = np.array(['muon', 'charged_kaon', 'long_kaon'])
>>> cme = 5000.
>>> eng_gams = np.logspace(0., np.log10(cme), num=200, dtype=np.float64)
>>>
>>> spec = gamma_ray(particles, cme, eng_gams)
```

CHAPTER

EIGHT

HAZMA.GAMMA_RAY.GAMMA_RAY_RAMBO

`hazma.gamma_ray.gamma_ray_rambo`(*isp_masses*, *fsp_masses*, *cme*, *mat_elem_sqrd_tree=<function <lambda>>*, *mat_elem_sqrd_rad=<function <lambda>>*, *num_ps_pts=1000*, *num_bins=25*)

Returns total gamma ray spectrum from a set of particles using a Monte Carlo.

Returns total gamma ray spectrum from a set of particles. This is done by first running a Monte Carlos to compute the non-radiative cross cross_section, $\sigma(I \to F)$ from the non-radiative squared matrix element: $|M(I \to F)|^2$. The differential radiative cross section, $\frac{d\sigma(I\to F+\gamma)}{dE_\gamma}$, is then computed using a Monte Carlo using the radiative squared matrix element $|M(I \to F + \gamma)|^2$. The spectrum is then

$$\frac{dN}{dE_\gamma} = \frac{1}{\sigma(I \to F)} \frac{d\sigma(I \to F + \gamma)}{dE_\gamma}$$

Parameters

- **isp_masses** (*np.ndarray[double, ndim=1]*) – Array of masses of the initial state particles.
- **fsp_masses** (*np.ndarray[double, ndim=1]*) – Array of masses of the final state particles.
- **cme** (*double*) – Center of mass energy of the process.
- **mat_elem_sqrd_tree** (*double(*)(np.ndarray[double, ndim=1])*) – Tree level squared matrix element.
- **mat_elem_sqrd_rad** (*double(*)(np.ndarray[double, ndim=1])*) – Radiative squared matrix element.
- **num_ps_pts** (*int*) – Number of Monte Carlo events to generate.
- **num_bins** (*int*) – Number of gamma ray energies to use.

Returns

- **eng_gams** (*np.ndarray[double, ndim=1]*) – Array of the gamma ray energies.
- **dndes** (*np.ndarray[double, ndim=1]*) – Array of the spectrum values evaluated at the gamma ray energies.

Examples

Compute spectrum from two fermions annihilating into a pair of charged fermions through a scalar mediator.

```
>>> from hazma import rambo
>>> from hazma.matrix_elements.simplified_models import xx_to_s_to_ff
>>> from hazma.matrix_elements.simplified_models import xx_to_s_to_ffg
>>>
>>> mx, ms = 120., 0.
>>> qf = 1.
>>> gsxx, gsff = 1.0, 1.0
>>>
>>> num_ps_pts = 10**6
>>> fsp_masses = np.array([mmu, mmu, 0.0])
>>> isp_masses = np.array([mx, mx])
>>> cme = 1000.
>>> num_bins = 150
>>>
>>> tree = lambda moms : xx_to_s_to_ff(moms, mx, mmu, ms, gsxx, gsff)
>>> radiative = lambda moms : xx_to_s_to_ffg(moms, mx, mmu,
..                                           ms, qf, gsxx, gsff)
>>>
>>> engs, dnde = gamma_ray.gamma_ray_rambo(isp_masses, fsp_masses, cme,
..                                         tree, radiative, num_ps_pts,
..                                         num_bins)
```

CHAPTER

NINE

HAZMA.RAMBO.COMPUTE_ANNIHILATION_CROSS_SECTION

`hazma.rambo.`**`compute_annihilation_cross_section`**(*num_ps_pts*, *isp_masses*, *fsp_masses*, *cme*, *mat_elem_sqrd=<function <lambda>>*, *num_cpus=None*)

Computes the cross section for a given process.

Parameters

- **`num_ps_pts`** (*`int`*) – Total number of phase space points to generate.
- **`masses`** (*`numpy.ndarray`*) – List of masses of the initial state and final state particles.
- **`cme`** (*`double`*) – Center-of-mass-energy of the process.
- **`mat_elem_sqrd`** (*`(double)(numpy.ndarray) {lambda klist: 1}`*) – Function for the matrix element squared.
- **`num_cpus`** (*`int {None}`*) – Number of cpus to use in parallel with rambo. If not specified, 75% of the cpus will be used.

Returns

- **cross_section** (*double*) – Cross section for X -> final state particles(fsp), where the fsp have masses *masses* and the process X -> fsp has a squared matrix element of *mat_elem_sqrd*.
- **std** (*double*) – Estimated error in cross section.

CHAPTER

TEN

HAZMA.RAMBO.COMPUTE_DECAY_WIDTH

hazma.rambo.**compute_decay_width**(*num_ps_pts*, *fsp_masses*, *cme*, *mat_elem_sqrd=<function <lambda>>*, *num_cpus=None*)

Computes the cross section for a given process.

Parameters

- **num_ps_pts** (*int*) – Total number of phase space points to generate.
- **masses** (*numpy.ndarray*) – List of masses of the initial state and final state particles.
- **cme** (*double*) – Center-of-mass-energy of the process.
- **mat_elem_sqrd** (*(double)(numpy.ndarray) {lambda klist: 1}*) – Function for the matrix element squared.
- **num_cpus** (*int {None}*) – Number of cpus to use in parallel with rambo. If not specified, 75% of the cpus will be used.

Returns

- **cross_section** (*double*) – Cross section for X -> final state particles(fsp), where the fsp have masses *masses* and the process X -> fsp has a squared matrix element of *mat_elem_sqrd*.
- **std** (*double*) – Estimated error in cross section.

HAZMA.RAMBO.GENERATE_ENERGY_HISTOGRAM

hazma.rambo.**generate_energy_histogram**(*num_ps_pts*, *masses*, *cme*, *mat_elem_sqrd=<function <lambda>>*, *num_bins=25*, *num_cpus=None*)

Generate energy histograms for each of the final state particles.

Parameters

- **num_ps_pts** (*int*) – Total number of phase space points to generate.
- **masses** (*numpy.ndarray*) – List of masses of the final state particles.
- **cme** (*double*) – Center-of-mass-energy of the process.
- **mat_elem_sqrd** (*(double)(numpy.ndarray) {lambda klist: 1}*) – Function for the matrix element squared.
- **num_bins** (*int*) – Number of energy bins to use for each of the final state particles.
- **num_cpus** (*int {None}*) – Number of cpus to use in parallel with rambo. If not specified, 75% of the cpus will be used.

Returns

energy_histograms (*numpy.ndarray*) – List of energies and dsigma/dE's. The resulting array has the shape (num_fsp, 2, num_bins). The array is formatted such that energy_histograms = {{{E11, E12, ...}, {hist11, hist12, ...}},

., ., .,

{{EN1, EN2, ...}, {histM1, histN2, ...}}}.

Examples

Making energy histograms for 4 final state particles and plotting their energy spectra.

```
>>> from hazma import rambo
>>> import numpy as np
>>> num_ps_pts = 100000
>>> masses = np.array([100., 100., 0.0, 0.0])
>>> cme = 1000.
>>> num_bins = 100
>>>
>>> eng_hist = rambo.generate_energy_histogram(num_ps_pts, masses, cme,
...                                            num_bins=num_bins)
>>> import matplotlib as plt
>>> for i in range(len(masses)):
...     plt.loglog(pts[i, 0], pts[i, 1])
```

CHAPTER

TWELVE

HAZMA.RAMBO.GENERATE_PHASE_SPACE

hazma.rambo.**generate_phase_space**(*num_ps_pts*, *masses*, *cme*, *mat_elem_sqrd=<function <lambda>>*, *num_cpus=None*)

Generate a specified number of phase space points given a set of final state particles and a given center of mass energy. NOTE: weights are not normalized.

Parameters

- **num_ps_pts** (*int*) – Total number of phase space points to generate.
- **masses** (*numpy.ndarray*) – List of masses of the final state particles.
- **cme** (*double*) – Center-of-mass-energy of the process.
- **mat_elem_sqrd** (*(double)(numpy.ndarray) {lambda klist: 1}*) – Function for the matrix element squared.
- **num_cpus** (*int {None}*) – Number of cpus to use in parallel with rambo. If not specified, 75% of the cpus will be used.

Returns

phase_space_points (*numpy.ndarray*) – List of phase space points. The phase space points are in the form {{ke11, kx11, ky11, kz11, . . . , keN1, kxN1, kyN1, kzN1, weight1},

.

{ke1N, kx1N, ky1N, kz1N, . . . , keNN, kxNN, kyNN, kzNN, weightN}}

Examples

Generate 100000 phase space points for a 3 body final state.

```
>>> from hazma import rambo
>>> import numpy as np
>>> masses = np.array([100., 200., 0.0])
>>> cme = 1000.
>>> num_ps_pts = 100000
>>> num_fsp = len(masses)
>>>
>>> pts = rambo.generate_phase_space(num_ps_pts, masses, cme)
```

CHAPTER

THIRTEEN

HAZMA.RAMBO.GENERATE_PHASE_SPACE_POINT

`hazma.rambo.`**`generate_phase_space_point`**(*masses*, *cme*, *num_fsp*)

Generate a phase space point given a set of final state particles and a given center of mass energy.

Parameters

- **`masses`** (*`numpy.ndarray`*) – List of masses of the final state particles.
- **`cme`** (*`double`*) – Center-of-mass-energy of the process.
- **`mat_elem_sqrd`** (*`(double)(numpy.ndarray) {lambda klist: 1}`*) – Function for the matrix element squared.

Returns **phase_space_points** (*numpy.ndarray*) – List of four momenta and a event weight. The returned numpy array is of the form {ke1, kx1, ky1, kz1, . . . , keN, kxN, kyN, kzN, weight}.

CHAPTER

FOURTEEN

HAZMA.DECAY.MUON

`hazma.decay.muon(eng_gam, eng_mu)`

Compute dNdE from muon decay.

Compute dNdE from decay $\mu^{\pm} \to e^{\pm} + \nu_e + \nu_\mu + \gamma$ in the laborartory frame given a gamma ray engergy of `eng_gam` and muon energy of `eng_mu`.

Parameters

- **eng_gam** (*numpy.ndarray*) – Gamma ray energy(ies) in laboratory frame.
- **eng_mu** (*double*) – Muon energy in laboratory frame.

Returns **spec** (*numpy.ndarray*) – List of gamma ray spectrum values, dNdE, evaluated at `eng_gam` given muon energy `eng_mu`.

Examples

Calculate spectrum for single gamma ray energy

```
>>> from hazma import decay
>>> eng_gam, eng_mu = 200., 1000.
>>> spec = decay.muon(eng_gam, eng_mu)
```

Calculate spectrum for array of gamma ray energies

```
>>> from hazma import decay
>>> import numpy as np
>>> eng_gams = np.logspace(0.0, 3.0, num=200, dtype=float)
>>> eng_mu = 1000.
>>> spec = decay.muon(eng_gams, eng_mu)
```

CHAPTER

FIFTEEN

HAZMA.DECAY.NEUTRAL_PION

`hazma.decay.`**`neutral_pion`**(*eng_gam*, *eng_pi*)

Compute dNdE from neutral pion decay.

Compute dNdE from decay $\pi^0 \to \gamma + \gamma$ in the laborartory frame given a gamma ray engergy of `eng_gam` and neutral pion energy of `eng_pi`.

Parameters

- **eng_gam** (*double or numpy.ndarray*) – Gamma ray energy(ies) in laboratory frame.
- **eng_pi** (*float*) – Neutral pion energy in laboratory frame.

Returns spec (*np.ndarray*) – List of gamma ray spectrum values, dNdE, evaluated at *eng_gams* given neutral pion energy *eng_pi*.

Examples

Calculate spectrum for single gamma ray energy

```
>>> from hazma import decay
>>> eng_gam, eng_pi = 200., 1000.
>>> spec = decay.neutral_pion(eng_gam, eng_pi)
```

Calculate spectrum for array of gamma ray energies

```
>>> from hazma import decay
>>> import numpy as np
>>> eng_gams = np.logspace(0.0, 3.0, num=200, dtype=float)
>>> eng_pi = 1000.
>>> spec = decay.neutral_pion(eng_gams, eng_pi)
```

CHAPTER

SIXTEEN

HAZMA.DECAY.CHARGED_PION

`hazma.decay.`**`charged_pion`**`(`*eng_gam*, *eng_pi*`)`

Compute dNdE from charged pion decay.

Compute dNdE from decay $\pi^{\pm} \to \mu^{\pm} + \nu_{\mu} \to e^{\pm} + \nu_{e} + \nu_{\mu} + \gamma$ in the laborartory frame given a gamma ray engergy of `eng_gam` and muon energy of `eng_pi`.

Parameters

- **`eng_gam`** (*`double or numpy.ndarray`*) – Gamma ray energy(ies) in laboratory frame.
- **`eng_pi`** (*`double`*) – Charged pion energy in laboratory frame.

Returns spec (*double np.ndarray*) – List of gamma ray spectrum values, dNdE, evaluated at *eng_gams* given charged pion energy *eng_pi*.

Examples

Calculate spectrum for single gamma ray energy

```
>>> from hazma import decay
>>> eng_gam, eng_pi = 200., 1000.
>>> spec = decay.charged_pion(eng_gam, eng_pi)
```

Calculate spectrum for array of gamma ray energies

```
>>> from hazma import decay
>>> import numpy as np
>>> eng_gams = np.logspace(0.0, 3.0, num=200, dtype=float)
>>> eng_pi = 1000.
>>> spec = decay.charged_pion(eng_gams, eng_pi)
```

CHAPTER

SEVENTEEN

HAZMA.DECAY.LONG_KAON

hazma.decay.**long_kaon**(*eng_gam*, *eng_k*)

Compute dNdE from long kaon decay.

Compute dNdE from decay of charged kaon through $K \to X$ in the laboratory frame given a gamma ray engergy of `eng_gam` and long kaon energy of `eng_k`.

Parameters

- **eng_gam** (*float or numpy.ndarray*) – Gamma ray energy(ies) in laboratory frame.
- **eng_k** (*float*) – Charged kaon energy in laboratory frame.

Returns spec (*np.ndarray*) – List of gamma ray spectrum values, dNdE, evaluated at *eng_gams* given muon energy *eng_mu*.

Examples

Calculate spectrum for single gamma ray energy

```
>>> from hazma import decay
>>> eng_gam, eng_kl = 200., 1000.
>>> spec = decay.long_kaon(eng_gam, eng_kl)
```

Calculate spectrum for array of gamma ray energies

```
>>> from hazma import decay
>>> import numpy as np
>>> eng_gams = np.logspace(0.0, 3.0, num=200, dtype=float)
>>> eng_kl = 1000.
>>> spec = decay.long_kaon(eng_gams, eng_kl)
```

Notes

The decay modes impemendted are

$$K_L \to \pi^{\pm} + e^{\pm} + \nu_e$$

$$K_L \to \pi^{\pm} + \mu^{\mp} + \nu_{\mu}$$

$$K_L \to \pi^0 + \pi^0 + \pi^0$$

$$K_L \to \pi^{\pm} + \pi^{\mp} + \pi^0$$

CHAPTER

EIGHTEEN

HAZMA.DECAY.SHORT_KAON

`hazma.decay.`**`short_kaon`**`(`*eng_gam*, *eng_k*`)`

Compute dNdE from short kaon decay.

Compute dNdE from decay of short kaon through $K \to X$ in the laboratory frame given a gamma ray engergy of `eng_gam` and short kaon energy of `eng_k`.

Parameters

- **`eng_gam`** (*`double or numpy.ndarray`*) – Gamma ray energy(ies) in laboratory frame.
- **`eng_k`** (*`float`*) – Charged kaon energy in laboratory frame.

Returns **spec** (*np.ndarray*) – List of gamma ray spectrum values, dNdE, evaluated at `eng_gams` given muon energy `eng_mu`.

Notes

The decay modes impemendted are

$$K_S \to \pi^+ + \pi^-$$

$$K_S \to \pi^0 + \pi^0$$

Examples

Calculate spectrum for single gamma ray energy

```
>>> from hazma import decay
>>> eng_gam, eng_ks = 200., 1000.
>>> spec = decay.short_kaon(eng_gam, eng_ks)
```

Calculate spectrum for array of gamma ray energies

```
>>> from hazma import decay
>>> import numpy as np
>>> eng_gams = np.logspace(0.0, 3.0, num=200, dtype=float)
>>> eng_ks = 1000.
>>> spec = decay.short_kaon(eng_gams, eng_ks)
```

CHAPTER

NINETEEN

INDICES AND TABLES

- genindex
- modindex
- search

INDEX