AES509 Final Project

Daniel Fernandez

December 2025

1 Module

For this project I created a module named NeutrinoFlux.py that creates an energy flux model for neutrino emission from Gamma Ray Bursts (GRBs). Determining the limits on neutrino energies for GRB spectral data is essential in the pursuit of identifying a multi-messenger signal in astrophysics. The field of multi-messenger astronomy seeks to collate the different methods of observing our universe to get multiple detections for the same event(s), which can teach us so much more about the physics of what we are seeing. This module focuses on what I study (Gamma Rays) and one other emission type (Neutrinos).

The module's input data are the parameters that we use in my research field to try and fit the spectra of GRBs to what is known as the BAND function. This BAND function is shown below, and contains the variables ϵ_γ, α_γ, E_γ, and β_γ.

$$F_\gamma(E_\gamma) = \frac{dN(E_\gamma)}{dE_\gamma} = f_\gamma \begin{cases} \left(\frac{\epsilon_\gamma}{\mathrm{MeV}}\right)^{\alpha_\gamma} \left(\frac{E_\gamma}{\mathrm{MeV}}\right)^{-\alpha_\gamma}, & E_\gamma < \epsilon_\gamma \\ \left(\frac{\epsilon_\gamma}{\mathrm{MeV}}\right)^{\beta_\gamma} \left(\frac{E_\gamma}{\mathrm{MeV}}\right)^{-\beta_\gamma}, & E_\gamma \geq \epsilon_\gamma \end{cases},$$

Figure 1: Band Function as written by Zhang (2013)[1]

This work is based on the theory done by Bing Zhang and Pawan Kumar in their paper on modeling neutrino fluxes [1]. I expand on their work by changing their flux equations to account for bulk GRB data and normalization of fluences. This helps to compare the modeled neutrino fluxes to what is expected based on our current knowledge of neutrino production in high energy events. The comparison is shown by the plotting functions within my module that create a plot with both my model data and the estimated neutrino energy data from the IceCube collaboration.

This module has 5 functions within; get_input_parameters, flux_integrand, Nu_flux, plot_individual_fluxes, and plot_combined. The first two are simple, extracting the needed inputs from a given csv file and defining the formula used

in integrating these function for fluence. Nu_flux is the main beefy function that takes the input data and converts a given BAND function GRB spectrum into an estimate for a neutrino energy spectrum. The last two functions are self-explanatory as two different plot outputs.

2 Example Use Case

The example I used for running/testing the module takes a csv file of GRB parameter data and creates the two plots using the functions discussed. The plots, shown in figures 2 and 3, are common energy flux plots used in GRB spectra research which compare fluence to the energy range of the spectrum. The code I wrote to create these from 300 data points uses a for loop in the testing section under the module that iterates over each burst to create the first plot. For all the necessary inputs with a multi-point csv file just follow the loop format shown in the module transcribed in Appendix A, specifically the test section under if==__name__==__main__. It then saves the Flux outputs to a new array that gets summed up to create the second plot.

There are also a few assert statements in the testing to confirm if there are errors/issues with NaN or negative values which shouldn't be allowed.

3 Notes

Collecting the information needed for GRB analysis can be difficult at the best of times, I probably could have made a project about collating these parameters more easily all on its own if I wanted to.

The csv file I use for the data and analysis is from realistic GRB parameters but it does serve the purpose of collecting realistic data to analyze.

The sample size of 300 GRBs does clutter the first plot quite a bit but this one is mostly just to see how the BAND function changes based on the parameters that are inserted.

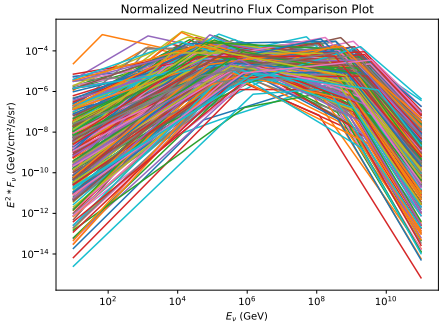

Figure 2: Comparing all 300 fluxes, messy sample size.

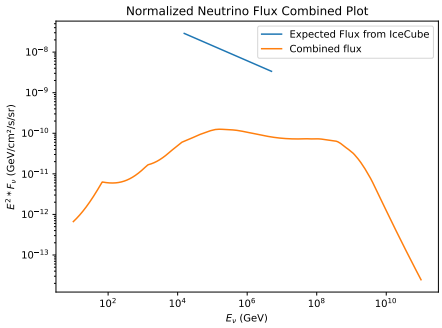

Figure 3: Comparison of combined total neutrino flux for one year with expected energy flux.

References

[1] Bing Zhang and Pawan Kumar. Model-dependent high-energy neutrino flux from gamma-ray bursts. *Physical Review Letters*, 110(12), March 2013.

Appendix A

```
"""
neutrino_flux.py

Combined module for:
- Computing GRB neutrino flux (Nu_flux)
- Plotting individual and combined neutrino fluxes

Includes a self-test and demo under:
    if __name__ == "__main__":
"""

import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as integrate
import astropy.units as u
import pandas as pd

#-------------------------------
#Retrieving Data
#-------------------------------
def get_input_parameters(infile):
    """Function to retrieve the desired input parameters from
        a CSV file.
    Parameters defined more clearly under Nu_flux function"""
    df = pd.read_csv(infile)
    L_52 = df["L_52"].values
    z=df["z"].values
    alp_y = df["alp_y"].values
    beta_y = df["beta_y"].values
    p = df["p"].values
    Gamma = df["Gamma"].values
    R_14 = df["R_14"].values
    Nnu = df["Nnu"].values
    eps_gamma = df["eps_gamma"].values
    eps_B = df["eps_B"].values

    return L_52, z, alp_y, beta_y, p, Gamma, R_14, Nnu,
        eps_gamma, eps_B
# -------------------------------
# Core physics / computation
# -------------------------------
def flux_integrand(E_nu, Flux_nu):
    """Helper integrand: E_nu * Flux_nu. Necessary fluence
        output in GRB research"""
    return Flux_nu * E_nu


def Nu_flux(
```

```
    L_52, z, alp_y, beta_y, p, Gamma,
    R_14, Nnu, eps_gamma, eps_B):
"""
Compute normalized neutrino flux for given GRB parameters.

Parameters
----------
L_52 : float
    Gamma-ray luminosity in units of 1e52 erg/s.
z : float
    Redshift, depicted as z in literature.
alp_y, beta_y : float
    Photon spectral indices (Band function).
p : float
    Proton spectral index.
Gamma : float
    Bulk Lorentz factor.
R_14 : float
    Radius in units of 1e14 cm (i.e., R = R_14 * 1e14 cm).
Nnu : int
    Number of neutrino energy grid points.
eps_gamma : float
    Photon spectral break energy (MeV).
eps_B : float
    Magnetic energy fraction.

Returns
-------
E_nu : ndarray
    Neutrino energies (GeV), log-spaced.
Flux_nu : ndarray
    Normalized neutrino flux (neutrinos / cm^2 / s / GeV).
"""
#Neutrino spectral indices derived from photon spectrum
    and p,
#Defined in Zhang, Kumar (2013)
alp_nu = p + 1 - beta_y
beta_nu = p + 1 - alp_y
gamma_nu = beta_nu + 2

# Neutrino energy grid (GeV)
E_nu = np.logspace(1, 11, Nnu)

#Optical depth and spectral breaks (following Zhang-like
    model)
Tau = (0.8 * L_52) / (((Gamma / 10**2.5) ** 2) * R_14 *
    eps_gamma)
eps_0nu1 = 7.3e5 * ((1 + z) ** -2) * ((Gamma / 10**2.5) **
     2) * (eps_gamma ** -1)
eps_nu2 = (
```

```
    3.4e8
    * ((1 + z) ** -1)
    * (eps_B ** -0.5)
    * (L_52 ** -0.5)
    * ((Gamma / 10**2.5) ** 2)
    * R_14
)
#Neutrino Spectrum Break Defined based on optical depth
    with Tau being smaller than 3
eps_nu1 = eps_0nu1 * min(1.0, (Tau / 3.0) ** (1.0 - beta_y
    ))

# Piecewise power-law neutrino spectrum before
    normalization
Flux_nu = np.zeros_like(E_nu)

#Spectral break locations
e1 = E_nu < eps_nu1
e_mid = (E_nu >= eps_nu1) & (E_nu <= eps_nu2)
e2 = E_nu > eps_nu2

#Each section of the piecewise Flux spectrum
Flux_nu[e1] = (eps_nu1 / E_nu[e1]) ** alp_nu
Flux_nu[e_mid] = (eps_nu1 / E_nu[e_mid]) ** beta_nu
Flux_nu[e2] = (
    (eps_nu1 ** beta_nu)
    * (eps_nu2 ** (gamma_nu - beta_nu))
    * (E_nu[e2] ** (-gamma_nu))
)

# Integrate non-normalized neutrino flux to create fluence
Integral = integrate.trapezoid(flux_integrand(E_nu,
    Flux_nu), E_nu)

# Photon flux setup for normalization
N2 = 1000
E_gamma = np.logspace(-3, 1, N2)  # MeV
F_gamma = np.zeros_like(E_gamma)

#New spectrum break points for second piecewise function
g1 = E_gamma < eps_gamma
g2 = E_gamma >= eps_gamma

F_gamma[g1] = (eps_gamma ** alp_y) * (E_gamma[g1] ** (-
    alp_y))
F_gamma[g2] = (eps_gamma ** beta_y) * (E_gamma[g2] ** (-
    beta_y))

Photon_I = integrate.trapezoid(F_gamma * E_gamma, E_gamma)
     # MeV/cm^2/s
```

```
    # Photon luminosity normalization
    DLcm = 2.067743e28  # cm, for z ~ 1 (example)
    MeV2erg = 1.6022e-6 #Converting energy from MeV to ergs
    L_gamma = 1e52 * L_52  # erg/s

    #Isotropic shell to get total Luminosity
    Luminosity_nonorm = Photon_I * MeV2erg * 4.0 * np.pi *
        DLcm**2
    f_gamma_norm = L_gamma / Luminosity_nonorm

    F_gamma *= f_gamma_norm
    Photon_I = integrate.trapezoid(F_gamma * E_gamma, E_gamma)
         # renormalized

    # Neutrino normalization (eq. 11-style)
    MeV2GeV = 1e-3 #Conversion factor
    L_y = 1.0 #Photon Luminosity (1e52 erg/s)
    L_p = 10.0 #Proton Luminosity
    f_ratio = L_y / L_p #Necessary conversion factor

    E_pmin = 20.0 * np.min(E_nu)
    E_pmax = 20.0 * np.max(E_nu)
    f_p = np.log(eps_nu2 / eps_nu1) / np.log(E_pmax / E_pmin)

    # Chi for average pion production rate in a given Photon
        Shell event
    X_p = 0.2
    # Number of neutrinos lost in production process
    Nloss = (1.0 / 8.0) * (f_p / f_ratio) * (1.0 - (1.0 - X_p)
         ** Tau)


    #Normalizing spectrum
    Norm_neutrino = Nloss * Photon_I * MeV2GeV / Integral
    Flux_nu *= Norm_neutrino

    return E_nu, Flux_nu


# ------------------------------
# Plotting helpers
# ------------------------------
def plot_individual_fluxes(E_arrays, F_arrays, L_values,
    outfile):
    """
    Plot E^2 * F_nu vs E_nu for many bursts and save to file.
    """
    plt.figure()
    ax = plt.subplot(1, 1, 1)
```

```
    ax.set_xscale("log")
    ax.set_yscale("log")
    E = E_arrays
    for F, L in zip(F_arrays, L_values):
        plt.plot(E, E**2 * F, label=f"L = {L*1e52:.2e} erg/s")

    plt.xlabel(r"$E_\nu$ (GeV)")
    plt.ylabel(r"$E^{2}*F_\nu$ (GeV/cm /s/sr)")
    plt.title("Normalized Neutrino Flux Comparison Plot")
    # Uncomment if you want the legend (can get crowded)
    #plt.legend(fontsize=2, loc="upper left")
    plt.tight_layout()
    plt.savefig(outfile)
    plt.close()


def plot_combined(E, F_tot, outfile):
    """
    Plot combined E^2 F_nu vs E_nu together with IceCube
        reference points.
    """
    #Sample Expected IceCube reference points
    #Represent neutrino energies that could be seen from a GRB
    x_ic = [5001198.290704517, 15481.563370926802]
    y_ic = [3.3598182862838015e-9, 2.8875287417994505e-8]

    F_combined = F_tot * 7.83e-9
    #Factor to average over a full year and full 4pi
        steradians for estimated
    #667 GRBs a year for total flux

    plt.figure()
    ax = plt.subplot(1, 1, 1)
    ax.set_xscale("log")
    ax.set_yscale("log")

    plt.plot(x_ic, y_ic, "-", label="Expected Flux from
        IceCube")
    plt.plot(E, E**2 * F_combined, label="Combined flux")

    plt.xlabel(r"$E_\nu$ (GeV)")
    plt.ylabel(r"$E^{2}*F_\nu$ (GeV/cm /s/sr)")
    plt.title("Normalized Neutrino Flux Combined Plot")
    plt.legend()
    plt.tight_layout()
    plt.savefig(outfile)
    plt.close()


# ----------------------------
```

```
# Self-test & demo block
# ------------------------------
if __name__ == "__main__":

    print("===Test and demo for neutrino_flux.py ===")
    # ---------------------------
    # 1) Dummy test: very simple parameters
    # ---------------------------
    print("\n[Dummy test] Single call with simple parameters
        ...")
    E_dummy, F_dummy = Nu_flux(
        L_52=1.0,
        z=1.0,
        alp_y=1.0,
        beta_y=2.0,
        p=2.0,
        Gamma=300.0,
        R_14=1.0,
        Nnu=50,
        eps_gamma=0.3,
        eps_B=0.01,
    )

    print(
        f"  E_dummy: length={len(E_dummy)}, "
        f"min={E_dummy[0]:.2e}, max={E_dummy[-1]:.2e}")
    print(
        f"  F_dummy: length={len(F_dummy)}, "
        f"min={F_dummy.min():.2e}, max={F_dummy.max():.2e}"
    )

    # Basic structural checks
    assert len(E_dummy) == 50
    assert len(F_dummy) == 50
    assert np.all(np.diff(E_dummy) > 0), "E_dummy must be
        strictly increasing (logspace)."
    assert np.all(F_dummy >= 0), "Flux should be non-negative
        ."

    print("  Dummy test passed.")

    # ---------------------------
    # 2) "Realistic" GRB-like test
    # (CSV Data File with 300 samples of realistic GRB band
       parameters)
    # ---------------------------
    print("\n[Realistic test] GRB-like parameter set...")
    (L_52, z, alp_y, beta_y, p, Gamma, R_14,
     Nnu, eps_gamma, eps_B) = get_input_parameters("./
         grb_neutrino_inputs_300.csv")
```

```
F_array = []
for i in range(len(L_52)):
    E_real, F_real = Nu_flux(
        L_52[i],
        z[i],
        alp_y[i],
        beta_y[i],
        p[i],
        Gamma[i],
        R_14[i],
        Nnu[i],
        eps_gamma[i],
        eps_B[i])
    F_array.append(F_real)

F_tot = sum(F_array)


plot_individual_fluxes(E_real, F_array, L_52, "
    Flux_comparison_Plot.pdf")
plot_combined(E_real, F_tot, "Combined_Fluxes_Plot.pdf")

# Check some sanity properties (no NaNs, positive, has
    structure)
assert not np.any(np.isnan(F_real)), "Realistic flux
    contains NaNs."
assert np.all(F_real >= 0), "Realistic flux should be non-
    negative."
print("  Realistic test passed.")
print("  Plots saved as:")
print("    - Flux_Comparison_Plot.pdf")
print("    - Combined_Fluxes_Plot.pdf")
print("\n===Self-tests and demo completed successfully.
    ===")
```