Function Manual : Blazar Halo from a surrounding Magnetic Field

I. UNPHYSICAL ASSUMPTIONS

The code generates event based on the constraints discussed in Long & Vachaspati (2015). Each simulated photons eventually produces a leptonic pair that upscatter a photon back to Earth, however their initial momentum has no constraint (i.e. we cannot guarantee that a simulated photon originated from a jet). The code works the following way. It first draws the energy E_γ of the photon that hit earth, the distribution of which we draw this energy can be found in globalvars.py (named *globalvars.drawE()*) and is currently based on Blazar emissions from the Fermi sample which suggest a spectrum fitted by a power law satisfying[1],

$$\frac{dN_{\gamma_0}}{dE_{\gamma_0}} \sim \left(\frac{E_{\gamma_0}}{\text{TeV}}\right)^{-2.5}, \tag{1}$$

which translates into the following spectrum for the observed photons when considering that $E_{\gamma_0} \sim E_\gamma^{1/2}$,

$$\frac{dN_{\gamma}}{dE_{\gamma}} \sim \left(\frac{E_{\gamma}}{\text{TeV}}\right)^{-1.75}. \tag{2}$$

One the energy is specified, one can work backward to determine the electron's energy E_e and initial photon's energy E_{γ_0}. With these we draw a propagation distance for the initial photon $D_\gamma(E_{\gamma_0})$ and the electron $D_e(E_e)$. We assume the mean free path (MFP) of the photon to be the mean travelling distance before the photon pair produces electrons, however the mean free path of the electron is currently set to be its cooling distance. In a realistic scenario, the electrons would produce a cascade of photons over D_e instead of emitting a single one at the end. See the discussion in the paper Duplessis & Vachaspati for some additional comments on this assumption.

II. REQUIREMENTS

On top of Numpy, Scipy and matplotlib, the following modules are required to run functions that utilizes allsky maps. As of this writing, the functions that uses these libraries are old and not ever used, hence not included in the manual. New ones should be written once all sky maps are simulated.
astropy,
mpl_toolkits.basemap

III. SETTING UP THE PARAMETERS

First open UserInput.py and select the settings. The variables available to set are all commented. These initializes the paremeters and, unless they are changed explicitly when calling functions, it is these values that will be used by the code. Some of these settings cannot be changed after initialization, it all depends on the extra arguments available for the coded function and wheter you want to change them yourself by redefining UserInput.(variable)=(new). Make sure to reload the modules if you decide to change some variables midway through. Once the UserInput.py variables are saved, one simply loads the modules which contains the functions of interest. An example code is given in example.py.

Note: All the angles must be supplied in radians, the function degrad converts degrees to radians.

---

[1] See arXiv:1609.00387 and reference therein.

IV. MAGNETIC FIELD CASES

The magnetic field is initialized from a set of 6 pre-determined background magnetic field cases in which we can embed a blazar (located at the origin, while Earth is at $\boldsymbol{z} = -D_s\hat{\boldsymbol{z}}$). To use random magnetic fields, one must do a bit more work beyond the UserInput.py folder, the work required is explained in sec IV A.

Case 1 to 5 are the same as the ones studied in Long & Vachaspati (2015).

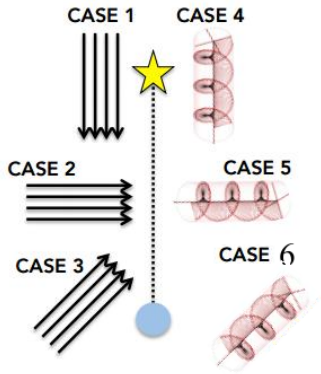

FIG. 1: Figure reproduced from Long & Vachaspati (2015)

- Case 1: $\vec{B}_1 = -B_0\hat{z}$
- Case 2: $\vec{B}_2 = B_0\hat{y}$
- Case 3: $\vec{B}_3 = B_0\big(\cos(\beta)\hat{y} - \sin(\beta)\hat{z}\big)$
- Case 4: $\vec{B}_4 = B_0\big(\cos(2\pi z/\lambda + \alpha)\hat{y} + \sigma\sin(\cos(2\pi z/\lambda + \alpha)\hat{x}\big)$
- Case 5: $\vec{B}_5 = B_0\big(\cos(2\pi x/\lambda + \alpha)\hat{y} - \sigma\sin(\cos(2\pi x/\lambda + \alpha)\hat{z}\big)$
- Case 6: $\vec{B}_6 = B_0\big(\cos(2\pi s/\lambda + \alpha)\hat{s}_{\perp,1} + \sigma\sin(\cos(2\pi s/\lambda + \alpha)\hat{s}_{\perp,2}\big)$

Here β is the orientation angle of the magnetic field in case 3, λ, α and σ are the coherence length, the phase shift and helicity of the magnetic fields in cases $\{4, 5, 6\}$. For $\vec{B}_6(\vec{x})$, we introduced $s, \hat{s}_{\perp,1}, \hat{s}_{\perp,2}$ where $s = \vec{x}\cdot\hat{s}$ for some arbitrary chosen $\hat{s}$ while

$$\hat{s}_{\perp,1}\cdot\hat{s}_{\perp,2} = \hat{s}_{\perp,1}\cdot\hat{s} = \hat{s}\cdot\hat{s}_{\perp,2} = 0. \tag{3}$$

To fix $\hat{s}$, the user supplies angles for $theta6, phi6$ and we construct,

$$\hat{s} = \cos(theta6)\hat{z} + \sin(theta6)\cos(phi6)\hat{y} + \sin(theta6)\sin(phi6)\hat{x} \tag{4}$$

$$\hat{s}_{\perp,1} = -\hat{s}\times\hat{x}/|\hat{s}\times\hat{x}| \text{ or if ill defined, } \hat{s}_{\perp,1} = \hat{s}\times\hat{y}/|\hat{s}\times\hat{y}| \tag{5}$$

$$\hat{s}_{\perp,2} = \hat{s}_{\perp,1}\times\hat{s} \tag{6}$$

Setting $theta6 = 0,\ phi6 = 0$ reduces to case 4.

A. Random Magnetic Field

It is also possible to use a random magnetic field comprised of modes all having the same magnitude k_{mag}. I.e. we can generate a field

$$B(\vec{x}) = \frac{1}{2M}\sum_{\vec{k}\in K}\vec{B}(\vec{k})e^{i\vec{k}\cdot\vec{x}} \tag{7}$$

with the set K being comprised of $2M$ vectors, half of which are isotropically chosen at random with magnitude k_{mag} while the other half insures a real B. For more information on how the Fourier modes are obtained, see the paper. Two options are used to select the vectors. The first is called the "uniform" option, here we select $M = 2N+1$ vectors approximately uniformely distributed on the sphere. The second option is called the "MC" (for Monte Carlo) option where $M = N$ selects the vector from a uniform random sampling of the sphere. Note that this allows more flexibility than specifying a fix case as listed above. The magnetic field used in the code is set by the function named B found in the constraint.py folder. Hence changing the function constraint.B will change the magnetic field throughout the code. The function quoted below will do just that, it will set the magnetic field to be computed by Eq. (7). As this process can be done relatively fast for small M, this field can be generate "on the go" and is therefore used when generating multiple realization of halo maps. The function we use is,

- stochMorp.setnewB(kmode,Nmodes,helicity,option=1,B0=UserInput.B0) —— Sets the magnetic field to be that of Eq (7)
  Input
  -kmode: float, specifies the norm of the mode k. It also sets the coherence length through $\lambda_c = 2\pi/\text{kmode}$.
  -Nmodes: Integer, sets the number of modes to use in the set K
  -helicity: Number between -1 and 1. Setting this to 0 yields a nonhelical magnetic field while 1 or -1 is fully helical.
  Optional Arguments
  -option: Integer denoting the options 1 or 2 as defined above. -option: B0 sets the root mean square value (in Gauss) of the random magnetic field.
  Output
  Sets B to be computed from Eq. (7) in subsequent simulations.

V. USEFUL FUNCTIONS

The code contains a large number of functions, most of which have an intermediate use or are simply out of date. In here we tabulate a few functions that should allow one to generalize the results of cite our paper. The more recent functions usually saves the data in the folder "sim_data" instead of returning it.

The main idea was to solve the constraint to find the PP locations and hence the observed photons, then run another function that selected the events in a specified jet. From this subset of events simulated we applied the functions that would compute the Q-statistics.

A. Event Generation

- EvSim.SimEvent(ds,NumPhotons,cn=1): —— Used to generate events.
  Input
  -ds : Distance of the source from Earth in Mpc.
  -NumPhotons: Number of photons to simulate.
  Optional Arguments
  -cn: Denotes the file location in which the event will be saved.
  Output
  This will create two documents in the folder "sim_data/3devents/case(cn)/"
  -3devents : Contains NumPhotons entries with the cartesian coordinates of their PP location, [..., $[x_i, y_i, z_i]$, ...]
  -3deventsangle : Contains NumPhotons entries with the angular coordinates and energies of the observed photons [..., $[\delta_i - \theta_i, \phi_i, E_i, \theta_i]$, ...]

- EvSim.JetEvs(NumJets=10,omega=5*gv.degrad,cn=1,doplots=False,jetfix=True): —— Creates a jet and then selects the events in said jet.
  The jets are randomly generated and have as requirement that at least 3 PP location lie within its openning angle. The code makes 20 attempts (I choose this number randomly to insure the code doesn't get stuck) to find a suitable jet.

  Input
  -NumJets: Number of jets to find.

-omega: Half-openning angle of the jet.
-cn: Integer of the folder "sim_data/3devents/case(cn)/" that contains the events and where additional files will be generated.
-doplots: Logical variable, if True, creates a 3dplot of the PP locations with the ones in the jets highlighted (the highlightning works poorly). The plots are saved as "3dimg_(jetnumber)".
-jetfix: If True, only generate jets that contains Earth in its LOS. If False, randomly select a PP location to be the center of the jet.
Output
-NumJets images of the observed events under the name "jetskyimg_(jetnumber)".
-Text files named "jetopenning_(jetnumber)" containing the value of omega.
-Text files named "jetdir_(jetnumber)" containing the cartesian coordinates of the unit vector denoting the jet's direction.
-Text files named "jetcart_(jetnumber)" containing the cartesian coordinates of the PP locations intersecting the selected Jet, [..., $[x_i, y_i, z_i]$, ...]
-Text files named "jetang_(jetnumber)" containing the the angular coordinates and energies of the observed photons due to the jet, [..., $[\delta_i - \theta_i, \phi_i, E_i, \theta_i]$, ...]

- EvSim.SimNHalos(N=2,kmode=kmode,Nmodes=Nmodes,hel=helicity,cns=100, njets=njets,Nphotons=Nphotons,MCB=False,B0=UI.B0,UseUIBins=True, case=False,jetfix=True,CalcBcB=False,CQ=0,omega=5*gv.degrad): —— Generates N simulations and saves the data in "sim_data/3devents/case(number)/"

  Input
  -N: Number of simulations to generate.
  -cns: Integer of the first folder "sim_data/3devents/case(cns)/" that contains the events and where additional files will be generated.
  -njets: Number of jets to find.
  -Nphotons: number of photons to simulate in each realization
  -omega: Half-openning angle of the jet.
  -B0: Sets the root mean square of the magnetic field in Gauss.
  -kmode: float, specifies the norm of the mode k. It also sets the coherence length through $\lambda_c = 2\pi/\text{kmode}$.
  -Nmodes: Integer, sets the number of modes to use in the set K
  -hel: Number between -1 and 1. Setting this to 0 yields a nonhelical magnetic field while 1 or -1 is fully helical.
  -jetfix: If True, only generate jets that contains Earth in its LOS. If False, randomly select a PP location to be the center of the jet.
  -CQ: Integer (0 or 1 as of this writing), CQ=0 (1) computes the Q-statistic without (with) the Heaviside function.
  -UseUIBins: if True, the energy bins used to compute Q are the ones supplied in the UserInput.py file. If False, the bins are automatically set by dividing the observed photons in 3 bins. -MCB: Logical, if True, uses the "MC" method to generate the random magnetic fields (i.e. option 2). We use option 1 if False. (see Sec IV A for details about the options)
  -case: Integer, if not False, this selects the magnetic field as defined by case (number) to be used.
  -CalcBcB: if True, computes $\langle \mathbf{B} \cdot \nabla \times B \rangle_{LOS}$ along the jet's LOS and outputs a plot in "/case(number)/"
  Output
  N files in the folder 'sim_data/3devents" (see the description of "cns"). The files contain the output of JetEvs and SimEvent. They also contain the Q statistics of every halo created by the i^{th} jet and j^{th} energy bin in the file named "Q_i_bin_j.txt" . The energy ranges of each bins are stored in the file "UsedEbins.txt" and the angular region R used as the x-axis of the Q values are stored in "region_i.txt".

- stochMorp.stomorp(ds,NumPhotons,phi=0,omega=np.pi,alpha=0,LeptonId=False) —— OLD AND LIMITED. We keep it as it can track the sign of the lepton producing the observed events. The Jet variables are only important when jetactivated=1 in the UserInput.py file.
  Input
  -ds: Distance of the source from Earth in Mpc.
  -NumPhotons : Number of photons to simulate.
  Optional Arguments
  -alpha: Angle of the jet with respect to Earth (alpha=0 $\implies$ jet points at earth).
  -phi: Angle of the jet in the azimuthal angle.
  -Omega: Openning angle of the jet (photons must lie within an angle of Omega of the direction set by alpha

and phi).
-LeptonId : if True, keep track of the charge $q = \pm 1$ of the lepton that upscattered the observed photon. If False, we return $q = 1$ for all events.

Output
tcospsol,tsinpsol,events
-events: Array of events with $[..., [\theta_i, \phi_i, E_i, q_i], ...]$
-tcospsol: Array of simulated event with $[..., \frac{\theta_i}{degrad} \cos\phi_i, ...]$
-tsinpsol: Array of simulated event with $[..., \frac{\theta_i}{degrad} \sin\phi_i, ...]$

B. Event Plot

- stochMorp.plotStoMorp(figname,phi=0,omega=np.pi,alpha=0,NumEvents=1000,Morp=False,LeptonId=False) —— Plots Blazar emission.
If no angles are supplied, the code assumes an isotropic emission from the Blazar. The code also assumes that the distance of the Blazar to earth is given by the global variable UserInput.dS .
Input
-figname: Name of the figure, example figname='test'
Optional Arguments
-alpha: Angle of the jet with respect to Earth (alpha=0 $\implies$ jet points at earth).
-phi: Angle of the jet in the azimuthal angle.
-Omega: Openning angle of the jet (photons must lie within an angle of Omega of the direction set by alpha and phi).
-NumEvents: Number of Photons to simulate.
-Morp : If True, also draw the expected morphology if no randomness in the MFP exists. This option can work poorly for certain magnetic field choices (such as a stochastic one).
-LeptonId : If True, color code the events by positron (red) and electron (black) instead of energy bins.
Output
Saves a figure in the *imgs/* directory as figname

- stochMorp.plotevs(figname,ev) —— Plot some event array ev
Input
-figname: Name of the figure, example figname='test'
-ev : Numpy array of events arranged by energy bins. Format as outputed by qstatistics.genevents: $ev[i, j] = [\theta_{i,j}, \phi_{i,j}]$, i:Number of Ebins, j:Number of Event.
Output
Saves a figure in the *imgs/* directory as figname

C. Q-statistics

- qstatistics.analyzeq(evtot,evBl=False,evMc=False) —— Computed Q out of events.
Input
evtotC: Total event file as output in genevents.
Optional Arguments
evBl: Blazar event file as output in genevents. If not passed, the output for Qbla is an array of 0.
evMc: Background event file as output in genevents. If not passed, the output for Qmc is an array of 0.

Output
Qtot,Qbla,Qmc
Qtot: Q-stats for total events.
Qbla: Q-stats for blazar events.
Qmc: Q-stats for background events.
The output have the same format: Q[i,j,k] with i being the energy pair number, $j = \{0, 1\}$ being the mean and standard deviation to Q's contribution from each HE photons, k being the index of the radius surveyed.

- qstatistics.plotq(figname,Qtot,Qbla=False,Qmc=False): —— Plots the Q's
  Input
  figname: Name of the figure, example figname='test' Qtot: Same format as outputted by qstats or qanalyze. The error bars are created using the value stored in the $j = 1$ index.
  Optional Arguments
  Qbla,Qmc: Same format as outputted by qstats or qanalyze. The error bars are created using the value stored in the $j = 1$ index. If these arrays are not passed to the function, we do not plot them.
  Output
  Saves a figure in the *imgs/* directory as figname
  Creates figures such as in fig 2.

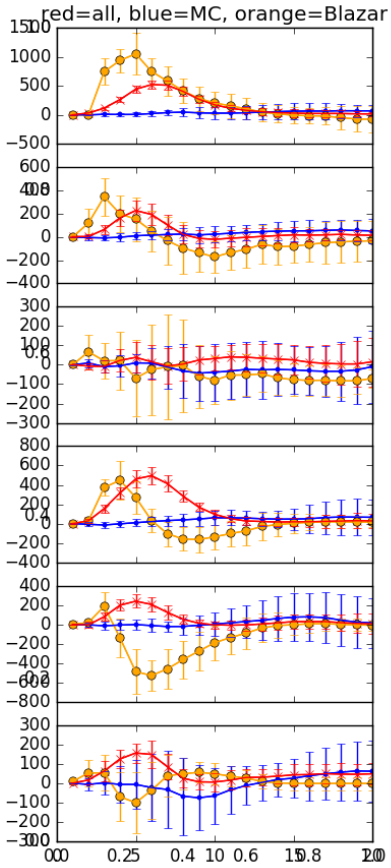

FIG. 2: q-stats figures. Labels, legends and titles might change in the future (for instance the title on the right figure is wrong but has then been fixed in the code).