**READ ME: PhotonBeam.py**

The main method is used to call on all methods and objects used to create kernels, figures, and total dose distributions.

**Description of Classes and Functions (top to bottom in the program):**

**Methods:**

**Main Method:** Defines the necessary variables, and sequentially calls on methods to get the Terma, the primary kernel, and then convolve to get the total dose, and any figures desired. Uncomment anything that is not needed.

**Energy\_transferred:** this method is used to sum over all the deposited energy in a kernel, and output this energy in Joules.

**Dose\_curve:** this method is used to create the PDD for question 6, and the transverse curve plot.

**Compton\_plot:** this method is used to generate the analytical Compton angle distribution, and overlay it with the monte carlo generated one. Before calling this, you need to adjust comptonAngle() so that it returns the photon angle, R1, instead of the electron angle, theta.

**TotalContourPlot:** this method is used to generate the transverse and sagittal total dose profiles as seen in question 6 of the report.

**Convolution:** this method is used to convolve the terma with the kernel, using a fast fourier transform.

**GetKernel:** this method is used to generate the primary dose kernel for a given photon energy. Input “True” as the final argument if you are loading a kernel.

**PhotonBeamSim:** this method is used to generate the interactions per voxel needed for the terma. It creates a phantom object, and centres the photon field in this phantom.

**VoxelInteractions:** this method is called on by PhotonBeamSim() to actually simulate all the interaction sites using the transformation method with the photon attenuation probability distribution.

**KleinNishina:** This method inputs an energy, and a photon scattering angle, and returns a sampled Compton electron scattering angle and energy.

**ComptonAngle:** This method uses the rejection method, and calls on KleinNishina to sample a Compton scattering angle and energy.

**ComptonScatter:** this method is called when a Compton interaction takes place. It will determine the electron scatter angle and energy from ComptonAngle and propagate the electron until its stopped.

**PPEnergy:** this method uses the transformation method with the pair production probability distribution for scattering angle, to determine the scatter angle of an electron and positron after a pair production interaction.

**PairProduction:** This method is called when a pair production interaction occurs. This will determine the electron and positron scatter angles and energies, and propagate them.

**CSV2Array:** this method is used to take a CSV containing electron stopping power data, and convert it into an interpolation function, which inputs an energy and outputs a stopping power.

**closestIndex:** this method takes in an array and a value, and find which array index corresponds to a number closest to value. This is used for determining which voxel in the phantom an electron is currently in.

**phiScatterAngle:** randomly samples the azimuthal scatter angle

**thetaScatterAngle:** Uses the multiple scattering theory to generate an electron scatter angle based on energy and step size.

**KernelContours:** This method takes a kernel, and makes the contour plot down the centre of the central axis, as seen in my report.

**Classes:**

**Photon:** a photon object can be created, with a certain position and initial direction, and its interact() method can be used to force it to interact at its position, and either a Compton, pair, or photoelectric event will be simulated, and the electrons (positrons) will be created and dose will be deposited.

**Electron:** create an instance of an electron with a starting position, direction, and call its transport() method to propagate it until its out of energy.

**Positron:** create an instance of a positron with a starting position, direction, and call its transport() method to propagate it until its out of energy.

**Phantom:** create a phantom object with a certain size, which will be used to hold the terma. Its addDose() method will add a given amount of energy to a certain position within.

**Kernel:** creates an object which will hold the kernel. Its addDose() method will add a given amount of energy to a certain position within.