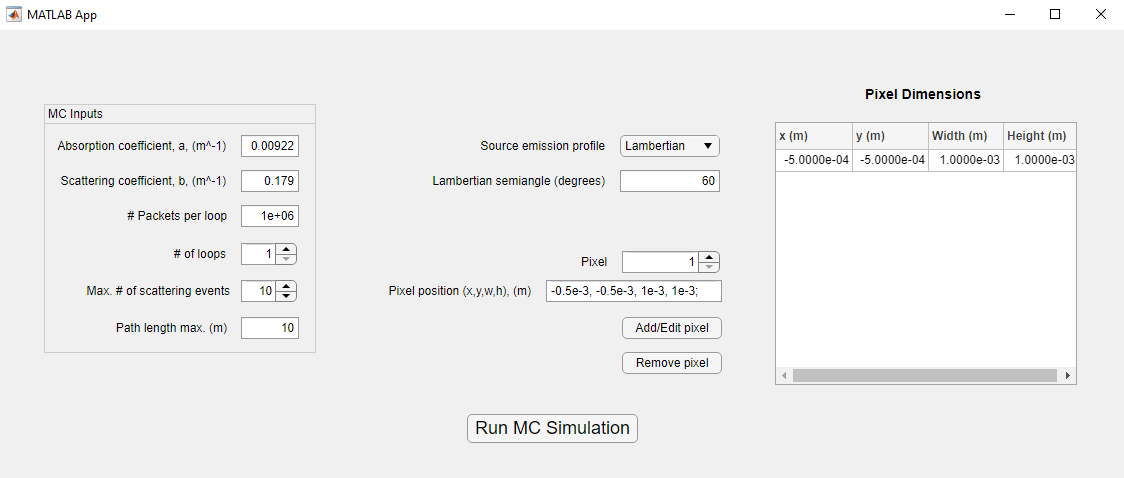
# Monte Carlo light scattering

This app creates and moves the photon packets according to inputs specified by the user. A user-specified number of packets are launched for each loop through the code, for a given number of loops. The final output of each loop includes all the relevant input parameters plus a full history of each photon packet, i.e. its position coordinates and weight at each step in the Monte Carlo modelling process. The outputs of each loop are saved to .mat files and named “LoopX.mat”, where X is the loop number.

## Inputs

The user is prompted to enter these via a GUI.



A summary of each input is given below. Some of these are saved to the “Loop.mat” output files in the variable *inputs*:

* **Absorption coefficient, *a*, (m-1)** – the absorption coefficient used to attenuate the photon packet’s “weight” according to: , where *d* is the distance the packet has travelled.
* **Scattering coefficient, *b*, (m-1)** – scattering coefficient used to calculate the probability of packet scattering.
* **# Packets per loop** – this is number of photon packets launched during each loop, in other words how many packets are being simulated at any given time. It is recommended not to exceed 2e7 packets, more than that and Matlab may not be able to save the large output files and/or a typical desktop PC will struggle with the large number of simultaneous calculations.
* **# of loops** – how many times the code simulates a batch of packets. Each loop is saved to a separate .mat output file.
* **Max. # of scattering events** – places a limit on how many times a packet can scatter before being terminated.
* **Path length max. (m)** – The maximum distance a packet can travel before it is terminated.
* **Source emission profile** – drop down menu allowing you to select the emission profile of the source(s). Choose from Lambertian, Idealised (a “perfect” laser source with no divergence) or a Custom emission profile that is imported from a .csv file.
* **Lambertian Semiangle –** if Lambertian is selected as the source emission profile, this is the half angle at which the intensity decreases to half of its maximum value.
* **Pixel dimensions** – this table lists the transmitter “pixels” that will be used. Their dimensions are specified by the *x* and *y* coordinate of their bottom left corner, their width in the *x* dimension and their height in the *y* dimension (all in m).
  + Pixels can be added, removed and edited using the appropriate buttons and the **Pixel** button used to select which pixel is to be edited/removed.
* **Run MC Simulation** – this begins the simulation. A progress bar will be shown which updates as each loop is completed and the results saved to disk.

## Outputs

The “LoopX.mat” output files save the complete results for the “Xth” loop. Each .mat file saves the key inputs for future reference (*a,b, max. # of scattering events, # packets per loop, pixel dimensions, max. path length, source type* and *total number of loops*).

Also saved is a cell array named *positions*. The first cell contains the positions and weights of every packet at as they were initialised at the transmitter, each subsequent cell contains the positions and weights of every packet at each occasion they were scattered. From this, a the full journey of each packet is preserved for further analysis.

The information in *positions* is formatted such that the columns from 1 to 4 contain (in order) the *x*, *y* and *z* coordinates (in m) and the packets’ weights at that position.

## Script overview

A basic overview of the underlying source code is as follows. The main code is in the file *MC\_scatter.m*, which calls other scripts as detailed below:

1. **Initialise packets:** A certain fraction of photon “packets” are created and launched per iteration. They are initialised with a starting position in the x,y and z axes, a starting scattering (elevation) angle, ψ, a starting azimuth angle, θ, and direction cosine.
2. **Move packets:** each active packet is moved a distance, *d,* which is chosen randomly for each packet. Their new (x,y,z) positions are set according to *d* and their previous direction cosine. Their “weights” are adjusted according to *d* and the absorption coefficient, to account for loss due to absorption.
3. **Terminate out of bound packets:** for computational efficiency, packets that have exceeded the maximum permitted path length are “killed”, and no longer take part in any further calculations.
4. **Terminate low-weight packets:** also for computational efficiency, packets that have a very low weight (<1e-6), and therefore contribute little to the final estimate of received power, are terminated.
5. **Test for loop end conditions**: check if all packets are inactive (i.e. terminated by steps c-d) or if the maximum number of scattering events has been reached. If either condition is met, the loop is terminated.
6. **Boost low-weight packets:** if the loop will continue, we boost the weight of low weight packets to preserve conservation of energy, as per [1].
7. **Scatter packets:** the remaining “active” packets are scattered. This randomly changes their scattering (elevation) angle, ψ.
8. **Update position and weight history:** the most recent packet positions and weights are added to the full history (*positions*).

Step a) occurs only once per loop, steps b) – h) are repeated until the loop terminates. A more detailed description of each step is given below.

## Step descriptions

### Initialise packets

This is performed by the script “create\_photons.m”. Its inputs and outputs are as follows:

|  |  |  |
| --- | --- | --- |
| Parameter | Input/Output | Description |
| *packets* | Input | Total # of packets to be initialsed |
| *init\_packet\_weight* | Output | Initial packet weights (max = 1) |
| *pos* | Output | Initial packet positions (x,y,z) |
| *dir* | Output | Initial packet direction cosines |

Note that *z* is the axis from the transmitter to the receiver, with the transmitter(s) at *z* = 0. *x* and *y* are the axes orthogonal to *z*. ψ is the elevation angle between *z* and the *x*,*y* plane, where ψ = 0° is parallel to the *z* axis. θ is the azimuth angle, the angle in the *x*,*y* plane.

In the case of the “basic” script described here, all packets are initialised at (0,0,0) with a direction cosine (0,0,1), i.e. all packets pointing directly towards the receiver.

### Move packets

This is performed by the function “move\_packets.m”. Its inputs and outputs are as follows:

|  |  |  |
| --- | --- | --- |
| Parameter | Input/Output | Description |
| *weights* | Input | Current packet weights |
| *pos* | Input | Current packet positions |
| *dir* | Input | Current packet direction cosines |
| *a* | Input | Absorption coefficient (m-1) |
| *b* | Input | Scattering coefficient (m-1) |
| *active\_packets* | Input | The list of “active” packets |
| *new\_pos* | Output | The updated packet positions |
| *weights* | Output | The updated packet weights |

1. The step size, *S*, for each packet is randomly calculated according to:

Where *q* is a random variable between 0 and 1. See Leathers *et al*. equation 3.4 [2]. (Note: they use the attenuation coefficient, *c*, in place of *b*, but I find that does not give accurate results in my script. Since *S* here is, I believe, related to the mean free path between scattering events *b* seems more appropriate).

1. The weights are updated according to:
2. The new *x,y* and *z* positions are updated according to:

Where µ is the x,y,z direction cosine. See Leathers *et al*. section 3.2 [2].

### Terminate out of bounds packets

This is performed by the function “path\_length\_kill.m”. Its inputs and outputs are as follows:

|  |  |  |
| --- | --- | --- |
| Parameter | Input/Output | Description |
| *pos* | Input | Current packet positions |
| *pmax* | Input | Maximum path length, (m) |
| *new\_active\_index* | Output | The updated list of “active” packets |

Any packets with cumulative path length > *pmax* are terminated. This is done by removing their indices from the list of “Active packets”. However, they are *not* deleted permanently, they are simply no longer used in any further calculations to update their positions etc., which is indicated in the input by setting the corresponding row representing their position and weight to “NaN” for subsequent iterations.

### Terminate or boost low weight packets

The script “abs\_kill.m” finds packets with a low weight (<1e-6) and randomly terminates 90% of them. The script “boost.m” searches the newly updated list of “active packets” for low weight packets. 10% of these packets have their weight boosted by a factor of 10. These steps are done notionally to improve computational efficiency while conserving the net energy within the simulation, as described by [1].

### Scatter packets

This involves randomly generating a new scattering angle for each active packet, and updating the direction cosines accordingly. This process is done by the script “scatter\_packets.m” using the following inputs and outputs.

|  |  |  |
| --- | --- | --- |
| Parameter | Input/Output | Description |
| *dir* | Input | Current packet direction cosines |
| *active\_index* | Input | The list of “active” packets |
| *dir* | Output | Updated packet direction cosines |

1. For isotropic scattering (i.e. where scattering is equally likely into any element of solid angle, not equally likely at every scattering angle) the new scattering angles are calculated using a random variable, *q*, with values between 0 and 1, as follows:

Note that this is not the only way to calculate the probability of a packet scattering into a given angle. This can be replaced with, e.g. an Fournier-Forand phase function, which more closely approximates natural bodies of water.

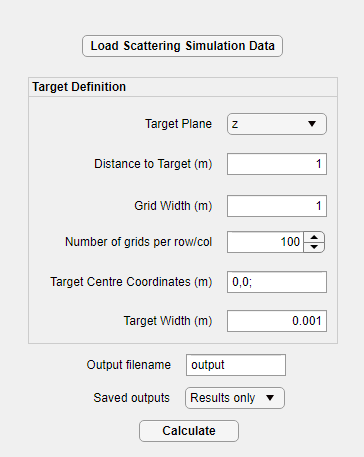
1. The direction cosines are updated using the equations found in Leathers *et al.*, section 4.2 [2]. For simplicity they are not shown here. Note that packets very close to the *z* axis (|µz| > 0.99999) have their direction cosines updated using a simpler approximation.
2. Azimuthal angles are randomised at every scattering step according to:

# photon packet analysis

This app uses the photon packet history generated by the previous app and finds which packets (if any) intercept a “target plane” specified by the user. The target plane is parallel to either the x, y or z plane, and lies at a specified distance (in metres) from the origin. The user is prompted for this information by a GUI.

The two main outputs are a matrix *weightMatrix* which lists the total sum of photon packet weights of all packets that intercepted the target plane at each matrix location, and *Rx\_received­\_total* which lists the total sum of photon packet weights that intercepted a single “target” at the receiver plane (analogous to measuring how many packets reach a target such as a photodiode).

## Inputs



* **Load Scattering Simulation Data** – this opens a dialogue box that allows the .mat files generated by the Monte Carlo app to be loaded for analysis.
* **Target Plane** – at present, the “target” must be specified as lying parallel to one of the *x*, *y* or *z* planes. By default the code assumes that the transmitter is parallel to the *z* plane and emits in the positive *z* direction, therefore the default receiver target is also parallel to the *z* plane.
* **Distance to target (m) –** the distance between the transmitter and receiver.
* **Grid Width (m) –** this is the width of a square area in the target plane, centred at (0,0) that will be analysed to see which photon packets (if any) intercept.
* **Number of grids per row/col** – this specifies how many rows and columns the target grid area will be subdivided into. For example, if Grid Width = 1m, and Number of rows/cols = 100, then the area to be analysed will be a square with sides of 1m, centred at (0,0) and divided into 100×100 grids each 1cm wide.
* **Target centre coordinates (m)** – these are the centre coordinates at which a square “target” (mimicking, for example, a photodiode) will be located.
* **Target width (m)** – this is the width of the square “target”.
* **Output filename** – results will be saved to this .mat file.
* **Saved outputs** – choose to save only the results (default) or “All” i.e. the results plus the full details of every single photon packet that intercepts the targets. Note that saving all can result in a very large output file.
* **Calculate** – this button begins the analysis. A progress bar will be displayed indicating how much data has been analysed, and a popup indicates when this is complete.

## Outputs

* **Rx\_received\_total:** this is the total sum of packets that “hit” the square detector.
* **xGrid** and **yGrid:** together these coordinates define the target grid
* **weightMatrix:** this is a matrix containing the sum of packet hits at each location within the grid.

## script overview

The following gives a description of the two scripts *photon\_packet\_analysis.m* and *target\_plane­\_analysis.m* that this app uses. The former analyses the imported data to find all the packets that intercept the target plane, the latter analysis those packets to generate the outputs.

### photon\_packet\_analysis.m

The script proceeds in three main steps:

1. The path taken by each photon is checked to see if at any point it intercepts the target plane. This is true if the corresponding coordinate for one step is < the corresponding coordinate of the target *and* the subsequent step is > than the target.  
     
   For example, we have two packets with position steps at (0,0,0) and (-1,-5,3) for Packet A, and (0,0,0) and (4,5,1) for Packet B. The target plane lies parallel to the z plane, and is at a distance of 2m. Packet A must intercept this target plane (0<2<3) but Packet B does not.  
     
   Packets that intercept the plane have the relevant pairs of position steps saved to the cell *hits*. Using the above example, Packet A would have (0,0,0) and (-1,-5,3) saved to *hits{1}* and *hits{2}*, respectively. Packets that never intercept have their rows in *hits* set to “NaN”.
2. These pairs of positions specify a line that represents the path taken by the photon packets during the step at which it intercepted the target. By solving this line equation we can find the coordinates at which this line intersects the target plane. Further details of how this is done is provided in the script.  
     
   The positions at which the packets intercept the target plane are saved to the variable c*oordinates*.
3. Finding the distance between *hit{1}* (the packet’s previous location) and the positions at which the packets intercept the target plane (*coordinates*), plus the absorption coefficient, we can adjust the packet weights to find their weight when they hit the target plane. This is saved as *hitweights*.

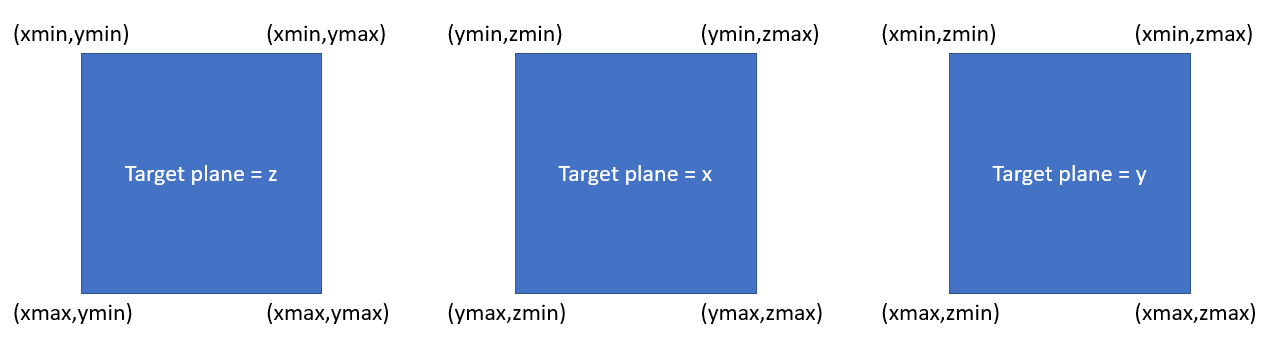
### Target\_plane\_analysis.m

This script takes the outputs from *photon\_packet\_analysis.m*, and calculates:

1. The sum of packet weights that “hit” a square detector, parallel to the target plane, with an area and location specified by the user. This is useful to represent how much total optical power hits a detector.
2. The distribution of packet weights across a grid covering a user-specified area parallel to the target plane. This is useful to represent the distribution of power across a target area. The grid is centred at (0,0) on the target plane.

# Notes

When plotting the output *weightMatrix* using the *contourf* function in MATLAB, it could potentially be confusing how these results are displayed by default. Without rotating or otherwise manipulating *weightMatrix*, here is how the orientation of the *contourf* plots will be, depending on which “target plane” was analysed:



# References

[1] J. M. Stujenske, T. Spellman, and J. A. Gordon, “Modeling the Spatiotemporal Dynamics of Light and Heat Propagation for InVivo Optogenetics,” *Cell Rep.*, vol. 12, no. 3, pp. 525–534, 2015, doi: 10.1016/j.celrep.2015.06.036.

[2] R. A. Leathers, T. V. Downes, C. O. Davis, and C. D. Mobley, “Monte Carlo Radiative Transfer Simulations for Ocean Optics: A Practical Guide,” 2004. [Online]. Available: http://www.stormingmedia.us/42/4266/A426624.html.