23 Jan 2017 – ReSET - Initial draft design for Zoe Fleming

Initial draft design

# Overview of target application

A convection/diffusion model uses historic wind data to reconstruct the paths of ~1000 particles released in a small area about a location *X*, backwards in time from a fixed time *T*. This generates snapshots of particle locations at times [*T, T-dt, T-2dt, …, T-Ndt]* for some finite integer *N*. The particle histories are stored as latitude-longitude positions at these time-steps.  
This is paired to a map assigning a value (A, B, C, …) to each physical location. These values may represent land vs sea, or terrain variables. By identifying the value of the area in which each particle resides at each time-step, queries such as “What fraction of the air above point *X* at time *T* has been above an area with value *A* within the last 3 days?” and “What fraction of the air above point *X* at time *T* was above an area with value *A* 3 days ago?”

# Current implementation

The latitude-longitude grid is subdivided by hand into approximate squares, and then each square is assigned a value in (*A, B, C, …*). This is carried out by hand, and is time-consuming and not easily scalable. Further, the ability to subdivide the latitude-longitude grid breaks down at the singularity at the poles.

# Proposed draft design - Input files

These files are generated beforehand, either as output from the chemical diffusion/convection model, or from a physical Earth map.

## Particle files

### One file per time-step. This contains a list of particle IDs and their position in latitude-longitude coordinates at time-step *t*. This optionally contains further fields of data for each particle, if required.

## Polygon file

### One file for all time. This contains a list of points in latitude-longitude coordinates, which denote the vertices of polygons, in a clockwise ordering. Each vertex has a field denoting the polygon ID to which it belongs. Multiple coincident points are allowed. While not needed as yet, it would be prudent to also include a unique Vertex ID per entry in the file. Polygon IDs are assigned from the interval [0, 1, …, N] where N is the number of polygons. Polygon ID 0 is reserved for the ‘exterior polygon’ containing all areas not contained in any other polygon (Polygon ID 0 can be assigned to any point not in any other polygon). This file to be generated by hand, although tools exist to generate such files from bitmaps or vector graphics.

## Polygon ID-to-Location Value lookup table.

### Consists of a map from Polygon IDs [0, 1, …, N] to Location Values in (A, B, C, …) which defines which Polygon belongs to which Location Value. Location Value may be used to denote a geographical area, e.g. West Africa, or an area defined by say vegetation, land/sea, or any other desired criteria. The population of this table is done by hand, but this should not be a large file.

# proposed draft design -Workflow

The crucial change from the current implementation is the removal of the longitude-latitude grid dependence for the assignment of Location Values. A single transformation of all particle location and vertex location data to a new reference frame is performed at the beginning of the workflow, and then all calculations are performed in this new reference frame.

## Generate the necessary particle files, polygon file, and lookup table.

## Transform all location data of polygon vertices and particle locations to a new reference frame.

### This may be a simple projection from the sphere to a plane, or something more complicated. In choosing this, the following should be taken into account:

#### Do straight lines remain straight under the transformation? (Unlikely). If not, is the error incurred reasonable? Adding more vertices along long straight polygon edges may help reduce the error, but additional edges slow down the later point-in-polygon search.

#### Is the transformation expensive to compute? If so, an initial transformation of all data is recommended. If the transformation is cheap, we could consider doing the transformation on the fly.

## To locate the polygon in which particle *Y* lives, we can use one of several point-in-polygon methods: the initial suggested method is ray-tracing.

### Let the location of particle *Y* be called *Z*. Draw a straight line from *Z* to the edge of the domain (ideally the nearest edge). For each polygon in the polygon file, we search for how many times it crosses an edge of that polygon. Any polygon of which the lines cross the edges an odd number of times, *Z* is inside. Once this polygon is found, we stop the search, and assign the Polygon ID to particle *Y* in a new table, or within a particle file copy.

The table now contains a list of particles and their assigned polygons.

# Summary

Removing the intermediate dependence upon the latitude-longitude grid removes the time-consuming step of hand-labelling squares and is more generalised – it should be simple to convert data files from a given standard to the required one with a simple script, with general polygon shapes. It removes the difficulty of a singularity at the poles.

The potential penalty paid is that the point-in-polygon routine may be more expensive than the current implementation, since point-in-square is a simpler problem to solve. Some review work would need to be done to see what impact this has, and whether the outcome is acceptable in terms of speed (fully considering the human hours saved by removing the hand-assignment of values to squares). Search heuristics could be explored to speed up the point-in-polygon search as required.