# Assignment 1 Additional Material - Laplacian Functions

The Grey-Scott Equations we are using for assignment 1 are :

* At + 1  = At + (DiffA \* Lap(A)) – AB2 + (FeedA \* (1-A))
* Bt+1 = Bt + (DiffB \* Lap(B)) + AB2 – ((KillB + FeedA) \* B)

You must compute Lap(A) and Lap(B) at each time step.

The Lap(x) function (short for Laplacian, from Pierre-Simon Laplace, the French mathematician who invented it) is an estimate of the impact of nearby chemical concentrations on chemical diffusion. This is an extremely complicated biological phenomenon which we as programmers fortunately do not need to know anything about. We just need to have some equations we can use to implement the correct behaviour.

Oversimplifying greatly, the higher the concentration of a chemical a cell has relative to its surroundings, the faster that chemical will diffuse. The Laplacian estimation quantifies this difference between a cell and its neighbours. There are many very complicated mathematical ways to compute the Laplacian estimations. The different techniques produce different Gray-Scott patterns. To explore the range of patterns the Gray-Scott model can generate, we will use three different Laplacian techniques in assignment 1. They all consider only the eight adjacent neighbours (i.e. one column and/or one row different).

The three techniques you must implement are as follows:

1. Perpendicular neighbours:

This technique considers only the north, south, east and west neighbours of the cell (the perpendicular neighbours). It does not use the diagonal neighbours (e.g. one column up and one column to the left). It computes a sort of difference between the average of the four perpendicular neighbours and the current cell, then multiplies this by some mysterious constant that comes from Biology.

Method:

Compute the following constants (data type should be double):

h = 2.5/127.0

rh = 1.0/h/h

The Laplacian estimate for chemical X for the current cell is then:

rh \* [(the sum of the concentrations of X in the four perpendicular neighbours) - ( 4 \* the concentration of X in the current cell)]

1. Convolution:

The second method computes a "convolution" of the cell and its neighbours. This is apparently a mathematical thing where you multiply each of the nine concentrations (the cell and its eight neighbours) by a constant, and sum the products. It is a sort of weighted average.

Method:

Picture the cell and its eight neighbours as:

|  |  |  |
| --- | --- | --- |
| Neighbour | Neighbour | Neighbour |
| Neighbour | Cell | Neighbour |
| Neighbour | Neighbour | neighbour |

Using the following constants:

|  |  |  |
| --- | --- | --- |
| 0.05 | 0.2 | 0.05 |
| 0.2 | -1 | 0.2 |
| 0.05 | 0.2 | 0.05 |

Compute the Laplacian estimate for chemical X for the current cell by multiplying the concentration of X in each cell by its corresponding constant (i.e. diagonal neighbours by 0.05, perpendicular neighbours by .2 and the cell itself by -1) and summing all the results.

1. Delta Means

This is the simplest estimation technique. It is just the difference between the mathematical average concentration of all the eight neighbours, and the concentration in the current cell.

Note that you will need to make some decision about how to deal with cells in the edge rows and columns, as they will not have all eight neighbours we have been discussing. There are two options: one is to simply not compute the Laplacian for edge cells, and not update them. This will make your simulation behave badly as it approaches the edge. A better solution is to treat the grid as though it wraps around both horizontally and vertically. That is, the left and right most columns are neighbours, as are the bottom and top rows. While this is not physically possible, the computer is quite happy to work this way, and you get much better patterns with this approach.

The different algorithms use different values for DiffA and DiffB, as follows:

|  |  |  |
| --- | --- | --- |
| **Laplacian Algorithm** | **DiffA** | **DiffB,** |
| Perpendicular | .00002 | .00001 |
| Convolution & Delta Means | 1 | 0.5 |