Potential Speed Ups for Ace’s Malaria Code

Running sleepy to profile the program, we can see that the majority of the time (sometimes upwards of 50%!) is spent calling iRandom. It would be good to reduce this dependency.

1. One way would be to generate all the random numbers beforehand, and then just access these as points in memory. The problem with this is that the domain over which the random numbers are generated changes over time, so they can’t simply be generated ahead of time!
2. A way round this is to only update the random number searcher every so often, as the numbers which govern the domain over which the numbers must be generated do not change significantly. The two largest sources of random number generation appear to be those within the LayEggs() and JuvDie() functions, with the former dominating the latter.
3. The problem with only updating the quantities periodically is that this may lead to numbers being picked which are outside of the acceptable range. For example, if there are 1,000 ovipositers and I want to pick 1 of them, then if from previously there were 1010 Ovipositers, it may occur that we pick # 1005.
4. Maybe we can get round this issue by generating the random numbers then just taking their modulus with the relevant quantity. In Matlab this is mod(big\_number,relevant\_number). The idea is that the big number is bigger than the maximum number of ovipositors for example. So, we would generate the random numbers a priori, then select that random number, then do modulo arithmetic on it.
5. The issue with this method is that we are simply replacing one function with another, but we hope that modulo arithmetic is very quick relative to repeated calls to random number generation!
6. The random number generation in LayEggs() chooses a random integer according to the number of Ovipositers in a particular large square. This must be significantly less than 1,000, but to be on the safe side I will use 1000 as the big\_number.
7. Tried the above, but it does not seem to make a significant difference to the speed of the program annoyingly! Need to make sure that the random numbers generated in LayEggs() and JuvDie() actually represent the majority of the random number calls.
8. Now have checked the total number of times IRandom is called – it is 100m s! This dwarfs that which we seem to be getting from the others.
9. Have tried to generate the (0,1) randoms ahead of time, but this doesn’t seem to have a significant effect on the speed that the program runs at. Perhaps it is because storing the random numbers as a long vector means that there is a large search time involved. Perhaps if I were to store the numbers in a more