1 Parallelization

The NSE version of the code is taking too long, and is not using much parallelization. I want to look into more ways that we can use parallelization to improve the performance of the code.

1.1 Chop up the Genome

According to Cedric and Mike, Wei Chen has done this in the past, and only sees significant improvement up to 6 or 8 processes. I don't see the parallel code anywhere in the current code (removed?)

1.2 Chop up the C code

The C code is taking up $\approx 40\%$ of the runtime, largely because it has to exponentiate and normalize a huge amount of data. Mike doesn't want to start initializing new threads at the C level, which is understandable. Of course we could start the threads earlier, but we don't want them to just be idling until the C code.

1.3 Use Parallelization on the MCMC

We could take multiple steps with the MCMC using different processors for each step. Multiple proposal values will let us converge faster, but we'll get diminishing returns on adding more processors. I can't imagine getting much advantage beyond 3 processors or so.

2 The DIEL

The only way I see the cubfits package going into the DIEL is with some kind of parallel tempering being applied. The module won't really want to exchange with any other modules, and doesn't need any sensitivity analysis or optimization (at this point).

As a note: any application of Mike's code to the DIEL will definitely require R to be a compatible part of the DIEL.

2.1 Parallel Tempering

The cubfits code is an MCMC. So we could use the same sort of parallel tempering that the Ising model uses. The performance could be improved by asynchronus parallel exchanges. This would be especially good when we move to the cubappr model (with no ϕ values), and we have to propose parameters from the beginning. We'll definitely be running multiple chains.