InFoMM C++ Skills Training

More practicals (STL, Mex, finite difference)

1. Use the data on http://www.fairtrade.org.uk/get_involved/campaigns/fairtrade_towns/towns_list.aspx to construct a list of "Fairtrade town" objects. This list ought to be stored as an STL vector with records representing name, type (city/town/zone/village) and region of the coun- try.
2. Use the same data to construct two STL multimaps: one keyed on type and one keyed on region. The idea is to enumerate all the Fairtrade villages (or whatever) without having to search through the origin list.

~~Cpp std::multimap map_by_region; . . .~~

3. Can you form the set intersection of all the Fairtrade villages in the South East and write out their names?
4. Copy the example file which has a mexFunction definition (Lecture 10) and save it in a `.cpp` file. Compile it with the mex compiler (or mkoctfile –mex for GNU Octave). Note that you will need to do this on a machine which has a valid version of Matlab (or the octave-headers package). Run the function from the Matlab interface (or from Octave).
5. Write a mexFunction which takes a vector as input and outputs the 2-norm of the vector. Extend the function so that it can optionally take another argument in order to compute the p-norm.
6. Write a mexFunction which multiplies two Matlab matrices. Check your answer against Matlab.
7. Solve the heat equation in 1D using an explicit finite difference method for $u(x, t)$ in $[0, 1]$. The equation is

$$u_t = u_{xx},$$

with Neumann boundary conditions at the end of the domain

$$u_x(0, t) = 0, \quad u_x(1, t) = 0$$

and an initial heat distribution given by:

$$u(x, 0) = cos(2x).$$

This example is discussed at http://commons.wikimedia.org/wiki/Image:Heatequation_exampleB.gif

The explicit finite difference scheme is

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{h^2},$$

where Δt is the timestep and is known to be convergent for step sizes $\frac{\Delta t}{h^2} \leq \frac{1}{2}$. Use a first difference to impose the boundary conditions:

$$\frac{u_0^{n+1} - u_0^n}{\Delta t} = 2\frac{u_1^n - u_0^n}{h^2}.$$

In order for the bookkeeping to work properly you won't want u_j to be updated before it's used in the calculation of u_{j+1}. Write your answer into another vector and then copy other the original at the end of a time-step.

8. Re-factor the code so that this copying does not take place. Use two vectors at each time-step (one for the input and one for the output). Swap them over at the end of each time-step.
9. Re-factor the code to use only one vector with the value of u_j being held to be re-used in the next iteration of the loop.
10. Consider (and implement if you have time) using an implicit (backward Euler) solver for this problem.