# InFoMM C++ Skills Training

## Practical 4 (Lectures 11-13)

This practical is based on the material in lectures 11-13. It builds on the ODE stepper class that you implemented in Practical 3, using templates to enable it to accept generic state types and derivative functions.

1. Template the ODE stepper class you wrote in the previous practical so that it can use a generic vector state type `T`. You may assume that this type has an assignment operator `=` defined which can copy one variable to another, and that it has `+` and `*` operators which allow state variables to be added together and multiplied by scalars.

2. Add another template argument `typename F` that represents a class that can calculate the time derivative of your generic vector state type `T`. You can assume that the class `F` has a function call operator `operator()` which takes 2 arguments, a `const T&` holding the current state, and a `T&` argument which holds the time derivative of the state variable after the operator is called. See the `negx` type below for an example. Make a constructor for your `OdeInt` class that accepts an argument of type `const F&` and saves this in a `private` variable for later use.

(Note: In general, a lightweight class with a function call operator is called a *function object*, or *functor*. A variable of this class can be passed to different functions/classes and used as a function internally. Function object are used heavily in the C++ STL.)

3. Template your `integrate` function using `typename T` and `typename F` representing the same classes in 1. and 2. Add another argument that takes a functor of type `const F&`, used to calculate the derivative.

4. Use your `integrate` function to integrate $dx/dt = -x$ as before, passing it an instance of the following function object. Verify the accuracy of the result.

```
struct negx {
    void operator()(const double &x, double &dxdt) { dxdt = -x; }
};
```

5. Write a partial specialisation for your ODE stepper class that uses a `std::vector` for the state type. Note that this type does not have the same assignment or arithmetic operators you probably have been using, so you will need to modify this code in the specialised class. Now repeat step 5. using a `std::vector<double>` for the state type.

6. Now we will integrate a second order ODE $\ddot{x} = -x$, with initial condition $x = 1$, $\dot{x} = 0$ for time $t = 10$ and using a timestep of $dt = 10^{-3}$. Convert $\ddot{x} = -x$ to the standard form $\frac{d\mathbf{y}}{dx} = f(x, \mathbf{y})$, where $\mathbf{y}$ is now a vector, and implement an appropriate function object that can calculate the rhs function $f(x, y)$ using a `std::vector<double>` for your state type `T`. You can check your answer using the analytical solution $x(t) = cos(t)$.

7. Do the same integration using an Eigen vector (i.e. `Eigen::VectorXd`) for the state type `T`, and verify that it gives the same result.