Introduction to C++

Practical 4 (Lectures 7-8)

Section 1

The error class and vector class designed in lecture 7 and a Makefile are available in the Git repository under `practical_day_4_code/`. This class of vectors includes the following:

```
- some constructors;
- a destructor;
- overloading of the binary operators +, -, * and /;
- overloading of the unary operators + and {;
- overloading of the operators = and ( ); and
- the functions norm and length.
```

Exceptions have been used to check that the vectors used are of the correct size.

1. Write a class of matrices that can be used with this vector class. Examples of features that should definitely be included are
   - A constructor
   - A copy constructor
   - Overloading of the binary operators + and - to define addition and subtraction of matrices
   - Overloading of the = assignment operator
   - Overloading of the unary - operator
2. Examples of other features you may wish to include are
   - Overloading of the () operator
   - Overloading of the * operator to define multiplication of matrices by scalars, vectors and other matrices
   - The function size that returns the size of a matrix (note that size is a vector)
   - Overloading of the / operator to define division of a matrix by a scalar and a vector - i.e. if $A * x = b$ then $x = b/A$. Use Gaussian elimination for solving $A * x = b$
   - Simple Matlab-style functions such as `size`, `eye` and `diag`
   - A more complex Matlab-style function such as `cgs` or `gmres`

Section 2

This practical builds on the ODE stepper class that you implemented in the previous days practical, 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 state vector `T`. You may assume that this vector type is similar to the vector type `Vector` given in the previous section, i.e. it has an assignment operator `=` defined which can copy one variable to another, `+` and `*` operators which allow state variables to be added together and multiplied by scalars, and a function is defined `length(x)` that returns the length of the vector `x`.

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 y, and a `T&` argument which holds dy/dx 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 lambda function is an example of a function object. 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 $dy/dx = -y$ as before, passing it an instance of the following function object. Verify the accuracy of the result.

```
struct negx {
    double operator()(const double &y) { return -y; }
};
```

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