InFoMM C++ Skills Training

Practical 3 (Lectures 7-10)

The first part of this practical covers writing a simple ODE integration stepper class. When implementing a solution to any given problem, it is important to think about how to break apart the problem into sub-problems to solve individually. For the case of ODE integration, you wish to solve the following problem:

$$\frac{dy}{dx} = f(x, y)$$

This naturally leads to two different sub-problems, (1) writing code to implement the rhs function $f(x, y)$, and (2) writing code to discretise the derivative $\frac{dy}{dx}$. We will use these sub-problems to inform the design of our ODE stepper class.

1. Write an ODE integration stepper class with the following properties:
   - A state variable `y` represented by a `double` type
   - A internal timestep `dx` represented by a `double`
   - A `dydx` function that implements the rhs function $f(y) = -y$ (i.e. sub-problem (1) discussed above).
   - A `step` function that takes a single integration step (i.e. sub-problem (2)) using an explicit Euler stepping scheme, and updates the internal state variable `y`.
2. Make the state variable and timestep variables `private`. Write one function that allows the user to set these private variables, and another function that can be used to access them (i.e. getter and setter functions). Only allow the timestep to be set to a positive and non-zero value.
3. Once you have a class that takes a single ODE integration step, the next problem to solve is how to integrate y over a given time range. Write an `integrate` function that takes an initial state `y0`, an integration time `x1`, and a step size `dx`. Use your stepper class to implement the function
4. Write test code which uses your integrate function to integrate $dy/dx = -y$ and verify it correctly calculates the solution $y(x) = exp(-x)$

The error class and vector class designed in lectures 9-10 and a Makefile are available in the Git repository under `linear_algebra/`. 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.

5. 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

6. 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`