Classes

Q4.1 In the lectures we defined a Point2D class, as follows

```cpp
class Point2D
{
public:
    Point2D(double x_, double y_)
    {
        x = x_; y = y_;}
    double DistanceToOrigin()
    {
        return std::sqrt(x*x+y*y);
    }
private:
    double x, y;
};
```

Copy this definition into a program, and check that it is working as intended by creating two Point2D objects, one with \(x = 3, y = 4\) and one with \(x = 12, y = 5\). Use the DistanceToOrigin function to display the distance of these points from the origin.

Q4.2 In the definition of Point2D, \(x\) and \(y\) are private member variables, and so cannot be accessed from outside the class (e.g. in main). Add public member functions void SetX(double x_) and double GetX() to get and set the values of \(x\). Do the same for \(y\). Write some code to test these functions.

Q4.3 Try creating a Point2D object without any parameters:

```cpp
Point2D c;
```

You will find that this produces a compiler error, since we have defined a constructor of Point2D that requires two parameters\(^1\). Create another Point2D constructor that takes no parameters and initialises \(x\) and \(y\) to zero, so that the code above compiles.

**Save your code for the Point2D class** as we will extend it further next week.

Derived classes and virtual functions

Q4.4 See the example code section of the course website for the file virtual_functions.cpp. This code implements the trapezium rule for integration using virtual functions to allow any integrand to be passed to the integration function. Copy this code into a new program and compile it. Suppose now that we want to evaluate

\[
\int_0^{10} e^{-x^2} \, dx.
\]

\(^1\)Recall that if no constructor at all is defined, a default constructor that takes no arguments and does nothing is assumed by C++ – but this default constructor is not available if we define a constructor of any sort in the class.
Make a class that derives from \texttt{Integrand} to implement this particular integrand, and call \texttt{Integrate} with the appropriate parameters to integrate it.

Q4.5 Suppose that we now want to evaluate
\[ \int_{0}^{10} \sin(\kappa x) e^{-x^2} \, dx \]
where \( \kappa \) is a parameter. Make a class that derives from \texttt{Integrand} to implement this integrand. If the parameter \( \kappa \) is stored as a member variable of your new class, you should not need to change either the \texttt{Integrand} class or the \texttt{Integrate} function. Is it better for this member variable \( \kappa \) to be \texttt{public} or \texttt{private}? Make a constructor for your new class that sets the value of \( \kappa \), and evaluate the integrand for various values of \( \kappa \) between 0 and 2.

Q4.6 Currently, the integrand is passed into the \texttt{Integrate} function by reference. However, we don’t want to modify the integrand object within the \texttt{Integrate} function, so it would be better to pass it as a \texttt{const} reference.

Recall that a variable passed into a function by \texttt{const} reference cannot be modified within that function. For a simple data type such as \texttt{int} or \texttt{double}, it is clear what we can do with a variable passed by \texttt{const} reference: we can access its value, but not modify it (by assigning it to another value). For a class data type (for example, if we pass into a function a \texttt{const} reference to a \texttt{Point2D}), the picture is not so clear. Some member functions of a class may modify the class (such as the \texttt{SetX} function of the \texttt{Point2D} object), but some member functions do not (such as the \texttt{DistanceToOrigin} function). We should only be able to call the latter member functions of a \texttt{const} instance of a class.

By default, C++ assumes that all member functions of a class modify its value, and so no member functions of a \texttt{const} instance of a class can be called. We can indicate that a member function does not modify the class by adding the word \texttt{const} after the member function declaration, e.g.

```cpp
1 double DistanceToOrigin() const
2 {
3     //...
4 }
```

With this addition, the member function can now be executed on a class passed by \texttt{const} reference.

Modify the \texttt{Integrate} function so that the \texttt{Integrand} is passed by \texttt{const} reference, and make the appropriate change to the \texttt{Integrand} class to allow this.

Q4.7 Recall from example sheet 2 the Newton method for finding roots of a nonlinear equation \( f(x) = 0 \). Starting from some initial guess of the location of the root \( x_0 \), we iterate
\[ x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \]

For a good enough initial guess, the sequence \( x_n \) converges to the root of \( f \).
Write a function `NewtonSolve` to perform the Newton iteration on a function $f$ that is passed in to `NewtonSolve` as a parameter. Use the same technique of virtual functions used to pass the integrand into the `Integrate` function in Q4.4, but this time the base class will need to provide a pure virtual function interface for both the function $f$ and its derivative $f'$. Use your Newton solver function to evaluate the roots of

$$x = e^{-x} \quad \text{and} \quad x = \cos(x).$$

Q4.8 In the previous question we used a pure virtual function for the derivative function $f'$. We could instead use a (non-pure) virtual function, with a default implementation in the base class that uses a finite-difference derivative to evaluate $f'$. Modify your base class to include this default implementation of the derivative function, and demonstrate that it produces very similar numerical results to those obtained in the previous question.