HIGHER ORDER FUNCTIONS AS OPERATORS 1

Within this problem, you will implement two higher-order functions in Scala. The function will each accept a function as an argument (amongst other things) and – further – returns a function.

In pursuing this home exercise, you may wish to make use of several useful facts regarding Scala:

a) recall that one can define a method that is curried ("staged") to accept successive arguments using syntax showing successive parameter lists. An example would be the following:
   def curriedFn(argA: Int, argB: Double)(argC:String) = *Some expression here*
b) one can write a method taking a unary function as an argument using the syntax such as the following:
   def higherOrderFunction(fn: Double => Double) = *Some expression here*
c) one can define an anonymous function (closure) using the syntax
   args => body
   where args can be a single parameter (e.g., x: Double) or a tuple of multiple parameters (e.g., (x: Double, y: Double))
d) a one dimensional Vector of *n* double-precision values can be easily created using the expression (among others), where *fnOfIndex* stands for a function accepting one argument (giving the index within the vector, starting from 0) and returning a double-precision value
   Vector.tabulate(n)(fnOfIndex)

1) The function that we will implement is the derivative operator of Calculus. Specifically, given a unary function *f(x)*, we will return a function that can evaluate a numerical approximation to the derivative (slope) *df/dx* of that function at a certain point x_0. You may remember that the value of the derivative of a function at a point is the slope of the function at that point – how quickly *f* is rising with *x*.

Given a function *f(x)*, we can evaluate a numerical approximation to the derivative at point x_0 as follows:

$$(f(x_0+\varepsilon)-f(x_0))/((x_0+\varepsilon)-x_0) = (f(x_0+\varepsilon)-f(x_0))/\varepsilon$$

Where "ε" (epsilon) is some small constant (e.g., 1E-2). Please note that the denominator, $(x_0+\varepsilon)-x_0$, indicates the change in x, while the numerator $f(x_0+\varepsilon)-f(x_0)$. indicates the change in the function f(x) over that interval between x_0 and $(x_0+\varepsilon)$.

Please create a Scala method (using def) that implements the derivative operator. This function should take in a Double returning unary function *fnUnary* (which serves as *f(x)*) and an epsilon (serving as ε), and *return a unary function* taking the value of *x0* (x_0) at which to evaluate the derivative. Given *x0*, that returned function will simply return $(f(x_0+\varepsilon)-f(x_0))/\varepsilon$ at that point x_0.

2) Using a value of epsilon (ε) of 1E-3 for all, please create named functions (declared using Scala's with *val* declaration) representing derivatives of each the following functions:

f(x)=1
f(x)=x
f(x)=2*x
f(x)=x*x

3) Please test out your derivative operator on the above by taking the derivative of successive values. Please first create a vector from going from -1 to 1, with steps of size 0.25. By using the map operator in Scala, please apply each of the derivatives that you created in step 2 to the test

points in turn, reporting the results. Do these results look reasonable? What limitations (if any) do you see?

3) Now create a numerical *integration operator*. This operator (implemented as a Scala method, using def) should first take in an Double returning unary function *fnUnary* (which serves as f(x)) and a Double precision value representing the integration step size dx, and then return a function which takes a starting point *xStart* and ending point *xEnd*, each represented Double precision values, and which then returns the result of integrating *fnUnary* from *xStart* to *xEnd* according to Euler integration with step size *dx*.

The currying requested above can be implemented by returning explicit functions, or by Scala's "curried" successive parameter lists, and using the "_" operator (as shown in class) to treat the results of the partially applied curried function as a function.