HIGHER ORDER FUNCTIONS AS OPERATORS 2

Within this problem, you will implement a higher-order function 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, as in the last one, 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)

The function that we will implement here is the integration operator of Calculus. Specifically, given a unary function *f(x)*, we will return a function that can evaluate a numerical approximation to the integral (area under the curve) of that function over a user-specified interval [x_{start}, x_{end}]. You may remember that the value of the integral of a function over an interval is the total area under the curve that function if considered only for that interval. As in the previous problem, we will create a numerical approximation to this value.

1) 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)). This should return a function that takes (in turn) a Double precision value representing the integration step size *dx*, and returns a function which takes (in turn) a starting point *xStart* and ending point *xEnd*, each represented Double precision values, and which then (finally) returns the result of integrating *fnUnary* from *xStart* to *xEnd* according to the "rectangle method" of integration with step size *dx*. (Please note that in the likely event that (*xStart*-*xEnd*) is not a perfect multiple of *dx*, you may simply take portion of the area of the final "rectangle" that falls within the interval [*xStart*, *xEnd*]). Thus, the computation will consist in phases, where one gets back a function representing the integral, and can then (separately) successively request the value of that integral for various intervals.

The Euler integration scheme that we will use here is particularly simplistic: For each interval of length *dx* over the interval [*xStart*, *xEnd*], we will approximate the value of the function to be integrated (*fnUnary*) as its value of the function. Thus, we can think of the area under the curve *fnUnary* as being approximated by a series of rectangles, each (with the exception of the last) being of width *dx*, and of a height given by the value of *fnUnary* at the value of *x* of the left side of that rectangle. (Thus, we are not using the value of fnUnary at the mid-point, but instead on the left of that rectangle). Please see the following reference for an animated illustration of the method (bearing in mind that the midpoint approximation also discussed in that page is not required):

Due in class September 2, 2016

HIGHER ORDER FUNCTIONS AS OPERATORS 2

https://en.wikipedia.org/wiki/Rectangle_method

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.

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

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

3) Please test out your integral operator on the functions above by evaluating them on the [0, 2] interval, reporting the results. Do these results look reasonable? What limitations (if any) do you see?

4) Using the derivative operator created in the previous exercise, how could you work to build confidence as to whether your integral operator is working?

Due in class September 2, 2016