Taller 2: Funciones de alto orden

Juan Francisco Díaz Frias Santiago Casañas

Octubre 2022

1. Ejercicios de programación

En ingeniería el concepto de la derivada de una función tiene muchas aplicaciones (¿conoce algunas?). Desde un punto de vista informático calcular la derivada de una función puede ser un ejercicio simbólico (la derivada de $f(x) = x^2$ es $f'(x) = 2x$), o puede ser un ejercicio numérico ($f'(5) = 10, f'(3) = 6, \ldots$).

En este taller nos vamos a enfrentar al ejercicio de, dado un programa que calcula una función f, construir un *derivador* que devuelva un programa que calcule la función f'.

Para ello vamos a usar una ecuación matemática denominada la *ecuación de 5 puntos* que establece que la derivada de una función f en un punto x_0 se puede aproximar por la siguiente ecuación:

$$f'(x_0) = \frac{f(x_0 - 2h) - 8f(x_0 - h) + 8f(x_0 + h) - f(x_0 + 2h)}{12h}$$

tomando $h = 0{,}1$ (ver https://es.slideshare.net/Hbrtx/algoritmos-derivada).

1.1. El derivador genérico

Implemente la función `derivada`, que reciba una función de una variable en los reales definida en Scala, y devuelva la función que calcula la derivada según la ecuación de los 5 puntos mencionada previamente:

```
def derivada(f:Double => Double): Double => Double ={ ...}
```

1.2. Derivadas de funciones compuestas

En el cuadro 1 recordamos algunas reglas de derivación.

Su tarea es implementar las funciones básicas siguientes para derivadas:

$$
\begin{aligned}
c &\overset{'}{\rightarrow} 0 \qquad (c \text{ es constante}) && (1)\\
x &\overset{'}{\rightarrow} 1 && (2)\\
f+g &\overset{'}{\rightarrow} f'+g' && (3)\\
f-g &\overset{'}{\rightarrow} f'-g' && (4)\\
f\cdot g &\overset{'}{\rightarrow} f'\cdot g + f\cdot g' && (5)\\
\frac{f}{g} &\overset{'}{\rightarrow} \frac{f'\cdot g - f\cdot g'}{g^2} && (6)
\end{aligned}
$$

Cuadro 1: Reglas de derivación usadas. Tenga en cuenta que f y g son funciones de la variable de derivación. También tenga en cuenta que $f'(x) = 0$ si f es una constante.

1.2.1. Derivada de la suma

Defina una función `derivadaSuma`, que dadas dos funciones f, g de una variable real definidas en Scala, devuelva la función derivada de $f + g$.

```
def derivadaSuma(f:Double=>Double, g:Double=>Double):Double => Double = {...}
```

1.2.2. Derivada de la resta

Defina una función `derivadaResta`, que dadas dos funciones f, g de una variable real definidas en Scala, devuelva la función derivada de $f - g$.

```
def derivadaResta(f:Double=>Double, g:Double=>Double):Double => Double = {...}
```

1.2.3. Derivada de la multiplicación

Defina una función `derivadaMult`, que dadas dos funciones f, g de una variable real definidas en Scala, devuelva la función derivada de fg.

```
def derivadaMult(f:Double=>Double, g:Double=>Double):Double => Double = {...}
```

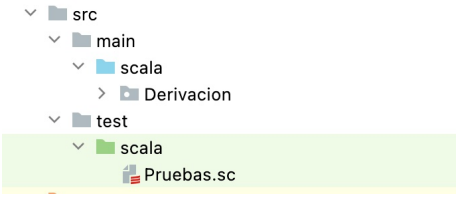
1.2.4. Derivada de la división

Defina una función `derivadaDiv`, que dadas dos funciones f, g de una variable real definidas en Scala, devuelva la función derivada de fg.

```
def derivadaDiv(f:Double=>Double, g:Double=>Double):Double => Double = {...}
```

1.3. Paquete *Derivacion* y *worksheet* de pruebas

Usted deberá entregar dos archivos `package.scala` y `pruebas.sc` los cuales harán parte de su estructura de proyecto `IntellijIdea`, como se muestra en la figura a continuación:



Las funciones correspondientes a cada ejercicio, `derivada`, `derivadaSuma`, `derivadaResta`, `derivadaMult`, y `derivadaDiv` deben ser implementadas en un paquete de Scala denominado `Derivacion`. En ese paquete debe venir un archivo denominado `package.scala` que debe tener la forma siguiente:

```
package object Derivacion {
  def derivada(f:Double => Double): Double => Double ={...}

 def derivadaSuma(f:Double=>Double, g:Double=>Double):Double => Double = {...  }

 def derivadaResta(f:Double=>Double, g:Double=>Double):Double => Double = {...  }

 def derivadaMult(f:Double=>Double, g:Double=>Double):Double => Double = {...  }

 def derivadaDiv(f:Double=>Double, g:Double=>Double):Double => Double = {...  }
}
```

Dicho paquete será usado en un *worksheet* de Scala con casos de prueba. Estos casos de prueba deben venir en un archivo denominado `pruebas.sc` . Un ejemplo de un tal archivo es el siguiente:

```
import Derivacion._
val cte= (x:Double) => 2.0
val f = (x:Double) => (x*x)
val g = (x:Double) => (x*x*x)
val h = (x:Double) => f(x) / g(x)
val j = (x:Double) => g(x) / f(x)
f

val h1 = derivadaDiv(f,g)
val h2 = derivadaDiv1(f,g)
val h3 = derivadaDiv(g,f)
val h4 = derivadaDiv1(g,f)

h1(2)
h2(2)
h3(2)
h4(2)

derivada(cte)(5)
derivada(cte)(6)
```

1.4. Informe del taller - secciones

Todo taller debe venir acompañado de un informe en formato pdf. El informe debe contener la información explícita solicitada en el enunciado.

Para este caso, el informe del taller debe contener al menos tres secciones: informe de funciones de alto orden, informe de corrección y conclusiones.

1.4.1. Informe de funciones de alto orden

Tal como se ha visto en clase, usar funciones de alto orden significa usarlas como parámetro, de forma anónima o como respuesta. Indique en una tabla, para cada función realizada, cuáles de estas tres formas de uso se utilizaron.

1.4.2. Informe de corrección

Es muy importante reflexionar sobre la corrección del código entregado. Para ello se deberá argumentar sobre la corrección de los programas entregados, y también deberá entregar un conjunto de pruebas. Todo esto lo consigna en esta sección del informe, dividida de la siguiente manera:

Argumentación sobre la corrección Para cada función,`derivada`, `derivadaSuma`, `derivadaResta`, `derivadaMult`, y `derivadaDiv`, argumente lo más formalmente posible por qué es correcta. Utilice inducción o inducción estructural donde lo vea pertinente. Estas argumentaciones las consigna en esta sección del informe.

Casos de prueba Para cada función se requieren mínimo 5 casos de prueba donde se conozca claramente el valor esperado, y se pueda evidenciar que el valor calculado por la función corresponde con el valor esperado en toda ocasión.

Una descripción de los casos de prueba diseñados, sus resultados y una argumentación de si cree o no que los casos de prueba son suficientes para confiar en la corrección de cada uno de sus programas, los registra en esta sección del informe. Obviamente, esta parte del informe debe ser coherente con el archivo de pruebas entregado, `pruebas.sc`.

1.5. Fecha y medio de entrega

Todo lo anterior, es decir los archivos `package.scala`, `pruebas.sc`, e `Informe del taller`, debe ser entregado vía el campus virtual, a más tardar a las 10 a.m. del jueves 20 de octubre de 2022, en un archivo comprimido que traiga estos tres elementos.

Cualquier indicación adicional será informada vía el foro del campus asociado a *programación funcional*.

2. Evaluación

Cada taller será evaluado tanto por el profesor del curso con ayuda del monitor, como por 3 compañeros que hayan entregado el taller. A este tipo de evaluación se le conoce

como *Coevaluación*.

El objetivo de la coevaluación es lograr aprendizajes a través de:

- La lectura de lo que otros compañeros hicieron ante el mismo reto. Esto permite contrastar las soluciones propias con las de otros, y aprender de ellas, o compartir mejores maneras de hacer algo con otros.
- Retroalimentar a los compañeros que fueron asignados para evaluar. Al escribir la percepción que tenemos sobre el trabajo del otro, podemos aprender de cómo lo hicieron, y dar indicaciones al otro sobre otra formas de hacer lo mismo o, incluso, felicitarlo por la solución que presenta.
- La lectura de las retroalimentaciones de mis compañeros o del profesor/monitor.

La calificación de cada taller corresponderá entonces:

- en un 80 % a la calificación ponderada que reciba del profesor/monitor, vía una rúbrica de evaluación (pesa 5 veces lo que pesa la de otro estudiante) y de los tres compañeros asignados para evaluarlo.
- en un 20 % a la calificación que el sistema hará del trabajo de evaluación asignado. El Sistema tiene un método inteligente para estimar esa calificación, a partir de las evaluaciones realizadas por cada estudiante y por el profesor/monitor.