

Representación de funciones

Curso 2004/05, Fecha:4/11/2004

Enunciado 1 (Plot) *El siguiente programa utiliza las funciones desarrolladas en la práctica anterior. Para ello, se utiliza la sentencia **import** Modulo donde Modulo es el nombre de módulo asignado a dicha práctica.*

El programa genera un fichero SVG representando una función

```
> module Plot where
> import LibXML(vacio, gen, genAs)

> d f = writeFile "fun.svg" (gen "svg" (plot f "blue"))

> tamX, tamY, x0 :: Double
> tamX = 500; tamY = 500; x0 = 10

> plot f c = concat (map linea pares)
>   where linea (x,x') = line (punto x) (punto x') c
>         pares       = [(x,x') | (x,x') ←
>                               zip lista (tail lista)]
>         lista       = [0..tamX]
>         punto x    = (x0 + x, tamY - f x)

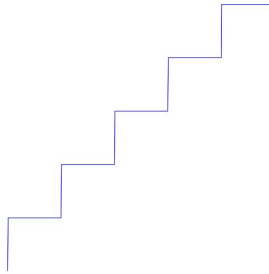
> line (x,y) (x',y') c =
>   vacio "line" [( "x1", show x), ("y1", show y),
>                  ("x2", show x'), ("y2", show y'),
>                  ("stroke", c)]
```

*El programa puede probarse ejecutando, por ejemplo d **cos** para representar la función coseno, o d $(\backslash x \rightarrow x / 2)$ para representar una recta.*

Enunciado 2 (representar) *Representar las siguientes funciones:*

```
f x = 0.2 * x^2 - 2 * x
f x = x * sin x
f x = abs (x * sin x)
f x = 50 * log (x + 1)
f x = 10 * sin x * log (x + 1)
```

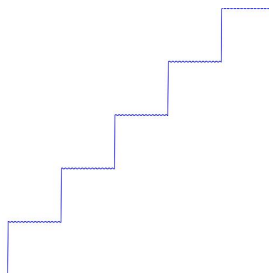
Enunciado 3 (escalera) *Representar una función escalera como la de la figura*



Enunciado 4 (suma) Implementar la función *suma* que toma como argumentos dos funciones y devuelva la función cuyos valores son la suma de los valores de dichas funciones. Tipo:

> *suma*:: (**Double**→**Double**)→(**Double**→**Double**)→(**Double**→**Double**)

Enunciado 5 (escalera rara) Representar la siguiente función



Enunciado 6 (media) Construir una función que tome dos funciones y genere una función cuyos valores son la media de los valores de dichas funciones. Tipo:

> *media*:: (**Double**→**Double**)→(**Double**→**Double**)→(**Double**→**Double**)

Enunciado 7 (comp) Construir una función *comp* que tome dos funciones y genere la función resultante de componer ambas funciones. Tipo:

> *comp*:: (**Double**→**Double**)→(**Double**→**Double**)→(**Double**→**Double**)

- Comprobar que el resultado de componer $f\ x = 4 * x + 1$ y $g\ x = x ^ 2$ es igual a $h\ x = 4 * x ^ 2 + 1$
- Representar la función formada por componer la función $f\ x = x ^ 2$ y $g\ x = \mathbf{sqrt}\ x$

Enunciado 8 (plot2) Construir una función que tome dos funciones y dibuje la representación gráfica de las dos funciones utilizando un color distinto para cada una. Tipo:

> $plot2 :: (Double \rightarrow Double) \rightarrow (Double \rightarrow Double) \rightarrow String$

Enunciado 9 (plotMedia(Opcional)) Construir una función que tome dos funciones y dibuje la representación gráfica de dichas funciones y de la función media Tipo:

> $plotMedia :: (Double \rightarrow Double) \rightarrow (Double \rightarrow Double) \rightarrow String$

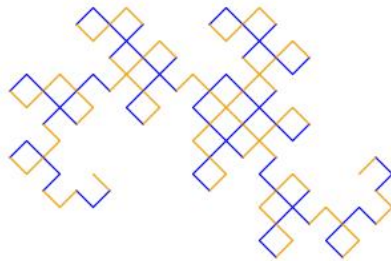
Enunciado 10 (plotLs(Opcional)) Construir una función $plotLs$ que tome una lista de funciones y las represente gráficamente. Tipo:

> $plotLs :: [Double \rightarrow Double] \rightarrow String$

Enunciado 11 (Fractal(Opcional)) El siguiente programa dibuja una imagen fractal en función de un parámetro n que indica el número de iteraciones.

Un ejemplo de llamada sería `drag 7`

A parte de ejecutar este programa, intentar generar otras imágenes fractales similares



```

> dragon (x1,y1) (x2,y2) (x3,y3) n =
>   if n == 1 then
>     line (x1,y1) (x2,y2) "blue" ++
>     line (x2,y2) (x3,y3) "orange"
>   else
>     dragon (x2,y2) (x4,y4) (x1,y1) (n-1) ++
>     dragon (x2,y2) (x5,y5) (x3,y3) (n-1)
>   where
>     (x4,y4) = ((x1 + x3) / 2, (y1 + y3)/2)
>     (x5,y5) = (x3 + x2 - x4, y3 + y2 - y4)
>
> drag = writeFile "fractal.svg" .
>   gen "svg" .
>   dragon (350,180) (250,80) (150,180)

```