IIP (E.T.S. de Ingeniería Informática)
Curso 2016-2017

Práctica 6. Iteración: realización de una clase de utilidades

Duración: dos sesiones

Profesores de IIP
Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València

Índice

1. Objetivos y trabajo previo a la sesión de prácticas

En algunas ocasiones el procesador con el que se trabaja no tiene predefinidas funciones matemáticas de uso habitual. En ese caso, no disponemos de operaciones trigonométricas o logarítmicas, que son muchas veces precisas para cálculos tan habituales como los de posición o de desplazamiento en pequeños elementos robóticos, drones, etc. Esto es muy frecuente cuando los procesadores con los que se trabaja son de poca potencia; como pasa, por ejemplo, en muchos microcontroladores (dispositivos corrientes en teclados, ratones, móviles de poca potencia, etc.). Cuando esto ocurre, hace falta implementar nuestras propias funciones matemáticas cuando nos sean necesarias.

Tomando como ejemplo lo anterior, en esta práctica resolverás algunos problemas numéricos para los que es necesario hacer uso de estructuras iterativas. En concreto, realizarás una clase de utilidades que contendrá métodos para el cálculo aproximado de dos funciones conocidas: la raíz cuadrada y el logaritmo de un valor.

Tras haberlas implementado, deberás comprobar su corrección comparándolas con las predefinidas en el lenguaje Java. Finalmente, deberás representar gráficamente ambas funciones utilizando una librería que te proporcionamos.

Para realizar adecuadamente la práctica es conveniente que hayas estudiado previamente el ejemplo 8.6 y el problema 21 del capítulo del libro "Empezar a programar usando Java" (3ª edición)[1]. Entender la solución de ambos te facilitará la resolución de las actividades propuestas.

Actividad inicial

Debes crear un proyecto en *BlueJ*, `pract6`, correspondiente a esta práctica, en el espacio de trabajo de la asignatura IIP. Hecho esto, debes crear en el mismo una clase `IIPMath`. El propósito de dicha clase será que incluyas en ella algunos métodos para calcular las funciones propuestas. Escribe, además, los comentarios de documentación necesarios en la cabecera de la clase.

2. Problema 1: Raíz cuadrada

La siguiente recurrencia permite calcular la raíz cuadrada, t, de cierto valor no negativo x:

$$t_1 = \frac{1+x}{2}, \qquad t_{i+1} = \frac{t_i + x/t_i}{2}$$

La recurrencia consiste en una aproximación sucesiva, $t_1 \ldots t_n$, al valor deseado (raíz cuadrada de x), donde el último término, t_n, es el más próximo a la raíz de x de entre todos los términos, t_i, generados.

2.1. Precisión de los cálculos. Condición de terminación

Es posible probar que la sucesión de términos $t_1 \ldots t_n \ldots$, se aproxima estrictamente a la raíz del número x, por lo que el error en el cálculo decrece estrictamente con cada nuevo término calculado.

Dados dos términos consecutivos t_{i-1} y t_i $(i > 1)$ se puede considerar su diferencia como una medida del error que se comete cuando se ha calculado el término i-ésimo. Por lo tanto, si deseas realizar el cálculo de la raíz cuadrada con cierto error máximo, bastará con que termines el proceso cuando la diferencia entre dos términos consecutivos sea menor que dicho error[2].

Siguiendo esto, se puede establecer un criterio de terminación de la iteración: cuando la diferencia entre dos términos consecutivos sea menor que el error deseado.

Actividad #1

Teniendo en cuenta la recurrencia anterior, implementa en la clase `IIPMath` un método público para calcular la raíz cuadrada de cierto valor x, no negativo, con un error máximo ϵ, siguiendo para ello el perfil siguiente:

```
/** Devuelve la raiz cuadrada de x >= 0, con error epsilon > 0. */
public static double sqrt(double x, double epsilon)
```

Implementa en la clase `IIPMath`, haciendo uso del método anterior, otro método público que sobrecargue al primero, para calcular la raíz cuadrada de cierto valor x, no negativo, con un error máximo 1e-15, con el perfil siguiente:

```
/** Devuelve la raiz cuadrada de x >= 0, con error 1e-15. */
public static double sqrt(double x)
```

Debes documentar adecuadamente cada uno de los métodos. Para ello, además del comentario en el que destaques lo que realiza el método, deberás describir también sus parámetros y el resultado del mismo (usando, respectivamente, el *tag* `@param` para describir los parámetros y el `@return` para el resultado). Por ejemplo, podrías documentar el método anterior como sigue:

[1] Ejemplo 9.6, problema 24 en la 2ª edición.

[2] Ten en cuenta que el error con el que puedas realizar el cálculo estará limitado por la precisión, o número de dígitos con los que trabajes. Un valor `double` en Java tiene una precisión de unos 16 dígitos.

```
/**
 * Devuelve la raiz cuadrada de x >= 0, con error 1e-15.
 * @param x. El valor, que debe ser mayor o igual a 0.
 * @return double. La raiz de x con error maximo 1e-15.
*/
```

Para comprobar que el código realizado es correcto, puedes utilizar el evaluador de expresiones del *BlueJ* (*Code Pad*) para comparar la función raíz realizada: `IIPMath.sqrt(double)`, con la que proporciona Java: `Math.sqrt(double)`.

3. Problema 2: Logaritmo de un valor

Para calcular el logaritmo natural de un valor cualquiera $x \in R^+$, se suele utilizar en primer lugar el siguiente desarrollo en serie que permite calcular el logaritmo de cierto z, con $1/2 \leq z < 1$. Sea:

$$y = \frac{1-z}{1+z} \tag{1}$$

entonces se conoce que:

$$log(z) = -2\sum_{i=1}^{\infty} \frac{y^{2i-1}}{2i-1} \tag{2}$$

Si se representa el término i-ésimo ($1 \leq i$) del desarrollo de la suma anterior por u_i, entonces:

$$log(z) = -2(u_1 + u_2 + u_3 + \cdots + u_k + u_{k+1} + \cdots + u_n + R_n)$$

esto es, toda la serie puede representarse como suma de términos u_i, junto con un resto R_n, que representa la suma de los términos restantes, posteriores al n-ésimo. O lo que es lo mismo:

$$log(z) = -2\sum_{i=1}^{n}(u_i) + R_n$$

3.1. Cálculo de los términos

El método que se construya deberá calcular cada uno de los términos u_i de la expresión anterior. Sustituyendo en (2), se puede obtener el primer término del sumatorio, u_1, que vale y. También sustituyendo en (2), se obtiene para los dos términos consecutivos cualesquiera: u_k y u_{k+1}, las siguientes expresiones:

$$u_k = \frac{y^{2k-1}}{2k-1} \qquad u_{k+1} = \frac{y^{2k+1}}{2k+1}$$

A partir de las mismas se puede observar que se cumple la siguiente relación entre dos términos consecutivos cualesquiera:

$$u_{k+1} = y^2 \frac{2k-1}{2k+1} u_k \tag{3}$$

Como se ve, es posible ahorrar cálculos si cada nuevo término se obtiene a partir del inmediatamente anterior, en lugar de calcularlo de forma independiente.

3.2. Precisión de los cálculos. Condición de terminación

Por características propias de la serie se puede demostrar que el error total que se comete (R_n), es siempre menor que el valor del último término calculado (u_n).

Por ello, si se desea que el error cometido sea menor que cierto ϵ, basta con calcular un término tras otro (sumándolos) hasta que se llegue a uno con valor inferior a dicho ϵ[3].

[3]Una vez más, ten en cuenta que el error con el que puedas realizar el cálculo estará limitado por la precisión, o número de dígitos con los que trabajes.

Actividad #2

Teniendo en cuenta la recurrencia en (3), implementa en la clase `IIPMath` un método público para calcular el logaritmo de cierto valor z, $1/2 \leq z < 1$, con un error máximo ϵ, siguiendo para ello el perfil siguiente:

```
/** Devuelve log(z), 1/2 <= z < 1, con un error epsilon > 0. */
public static double logBase(double z, double epsilon)
```

3.3. Cálculo general del logaritmo de un valor

Conocido el cálculo anterior (que permite determinar el logaritmo de un valor en el intervalo $[1/2, 1[$), veamos cómo se aplica para calcular el logaritmo de cualquier valor $x \in R^+$.

Dado un valor x, no negativo cualquiera, es posible transformarlo en un valor z en el intervalo $[1/2, 1[$ bien dividiéndolo tantas veces como sea necesario por 2 (cuando el valor original de x sea mayor o igual a 1), bien multiplicándolo las veces necesarias por 2 (cuando el valor original de x sea menor que $1/2$).

Esto es, se tendrá que x puede expresarse como:

$$x = 2^m z \qquad (1/2 \leq z < 1) \tag{4}$$

siendo m un valor entero positivo o negativo. Además, tras aplicar logaritmos a las dos partes de la ecuación anterior, se tiene:

$$log(x) = mlog(2) + log(z) \tag{5}$$

Permitiendo realizar el cálculo deseado que en resumen consistirá, para cierto x, en:

1. Si $x \in [1/2, 1[$ aplicar directamente el método `logBase(double, double)`.
2. Si $x \notin [1/2, 1[$, entonces:
   a) Si $x \geq 1$, dado que para que se cumpla la ecuación (4) se tiene que $z = x/2^m$, calcular z y m dividiendo x por 2 el número de veces necesario (m veces) para reducirlo a un valor en $[1/2, 1[$ (este valor reducido será el valor de z).
   b) Si $x < 1/2$, puede calcularse z y m mediante productos sucesivos de x por 2 en lugar de con divisiones. También puede utilizarse que $-log(1/x) = log(x)$. Por ejemplo, si se desea calcular $log(0{,}2)$ $(0{,}2 < 1/2)$ se tiene que $-log(5) = log(0{,}2)$. Y para calcular $log(5)$ se está en las condiciones del caso anterior.
3. Aplicar la ecuación (5) para, dado $log(z)$, calcular el logaritmo deseado. Téngase en cuenta que $log(2)$ es un valor conocido, cuya aproximación figura en el enunciado de la actividad siguiente:

Actividad #3

En primer lugar, define previamente el valor aproximado de $log(2)$ (0,6931471805599453) como una constante en tu programa.

A continuación, teniendo en cuenta las expresiones en (4) y (5), implementa en la clase `IIPMath` un método público para calcular el logaritmo de cierto valor x, positivo, con un error máximo ϵ, según:

```
/** Devuelve log(x), x > 0, con un error epsilon > 0. */
public static double log(double x, double epsilon)
```

Implementa en la clase `IIPMath`, haciendo uso del método anterior, otro método público que sobrecargue al anterior, para calcular el logaritmo de cierto x, positivo, con un error máximo 1e-15, según:

```
/** Devuelve log(x), x > 0, con un error 1e-15. */
public static double log(double x)
```

Al igual que antes, documenta adecuadamente cada uno de los métodos, destacando lo que realizan. Usa, los *tag*s `@param` y `@return` para describir sus parámetros y resultado.

4. Prueba de tus funciones Raíz y Logaritmo

Puedes comparar las funciones que has realizado con las que proporciona el lenguaje Java en su clase estándar `Math` para, de esa manera, comprobar la precisión de tus resultados.

Para facilitarte esta tarea, se te proporciona la clase `PruebaIIPMath` que debes descargar al proyecto BlueJ en el que realizas la práctica y que te permite comparar para un `x` dado, introducido por ti, el resultado de calcular su raíz y su logaritmo utilizando los métodos que has hecho (`IIPMath.sqrt(double x)` y `IIPMath.log(x)`) con los dos definidos en Java (`Math.sqrt(double x)` y `Math.log(x)`).

Actividad #4

Ejecuta la clase `PruebaIIPMath` para comprobar que los resultados de las funciones que has realizado son correctos (comparables a los de la versión del Java estándar) con distintos valores, tanto pequeños (p.e. del orden de $10^{-12}, 10^{-6}, 10^{-3}$), grandes (p.e. del orden de $10^{3}, 10^{6}, 10^{12}$), como valores en el rango de las unidades y de las decenas. Por supuesto, puedes comprobar los resultados para cualquier otro valor en el rango que tu creas.

5. Representación gráfica de las funciones

Para poder mostrar gráficamente funciones como las que has implementado, se te proporciona una librería predefinida para representar gráficamente puntos, líneas, entre otros elementos, en un espacio bidimensional.

En la figura 1, tienes una muestra de esta representación hecha mediante el dibujo punto a punto de dos funciones: $sen(x)$ y $sen(x)/x$ en el intervalo $[-1, 4\pi]$.

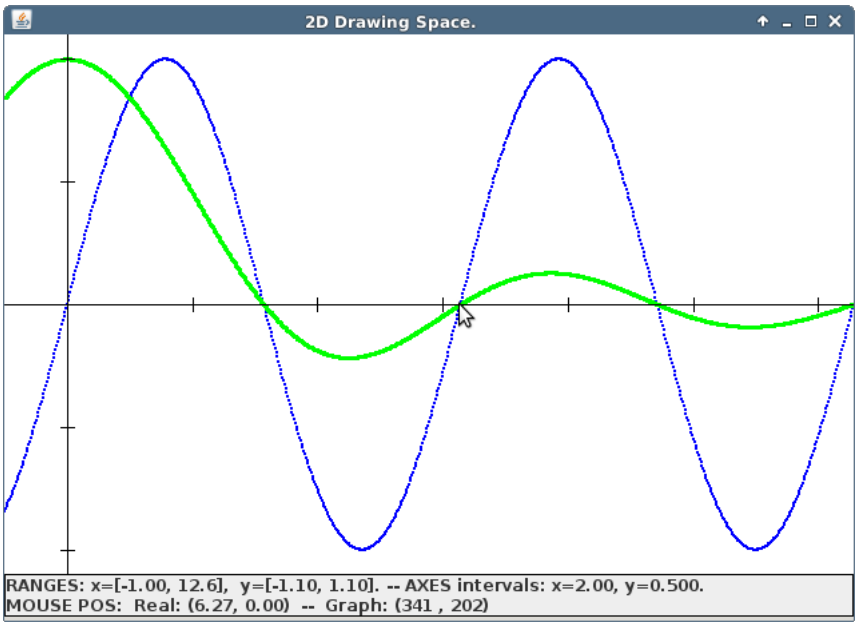

Figura 1: Representación punto a punto de sen(x) y sen(x)/x en $[-1, 4\pi]$.

Un dibujo punto a punto de los valores de cierta función f, se obtiene calculando para distintos valores posibles de x (en el intervalo de representación que se desee) los valores $f(x)$ correspondientes y representando gráficamente el par $(x, f(x))$ mediante la primitiva de representación gráfica correspondiente (método `drawPoint(double x, double y)` de la clase `Graph2D`).

En el fichero `Graph2D.html` se te proporciona la documentación de la clase `Graph2D` mediante la que es posible realizar representaciones gráficas de puntos y líneas en un espacio bidimensional.

Para poder utilizar la clase anterior debes instalar el paquete `graph2D`, que se te proporciona en un fichero en formato `jar` (`graphLib.jar`). Debes descargar el fichero (`graphLib.jar`) en tu carpeta `iip`.

Una vez lo hayas hecho, reconfigura el entorno BlueJ para, en `preferencias/librerias`, señalarle al BlueJ dónde encontrar la librería (el fichero `graphLib.jar` que has dejado en tu `iip`).

También se te proporciona un ejemplo de uso de `graph2D.Graph2D`, en el fichero `Graph2Dtest.java` que debes incluir en tu proyecto y que al ejecutarse mostrará una ventana similar a la de la figura 1. Puedes basarte en él para resolver las actividades que se te plantean a continuación.

Adicionalmente, en el anexo al final de este boletín se describe cómo representar gráficamente una de las dos funciones $(sen(x)/x)$ de forma similar a como aparece en `Graph2Dtest.java` y se representa en la figura 1.

Actividad #5

Representa gráficamente en el intervalo: $x \in [-1, 15]$ e $y \in [-3, 4]$, usando puntos, los valores de las funciones que has desarrollado (raíz y logaritmo). En la figura 2 tienes una muestra del resultado correspondiente a esta actividad.

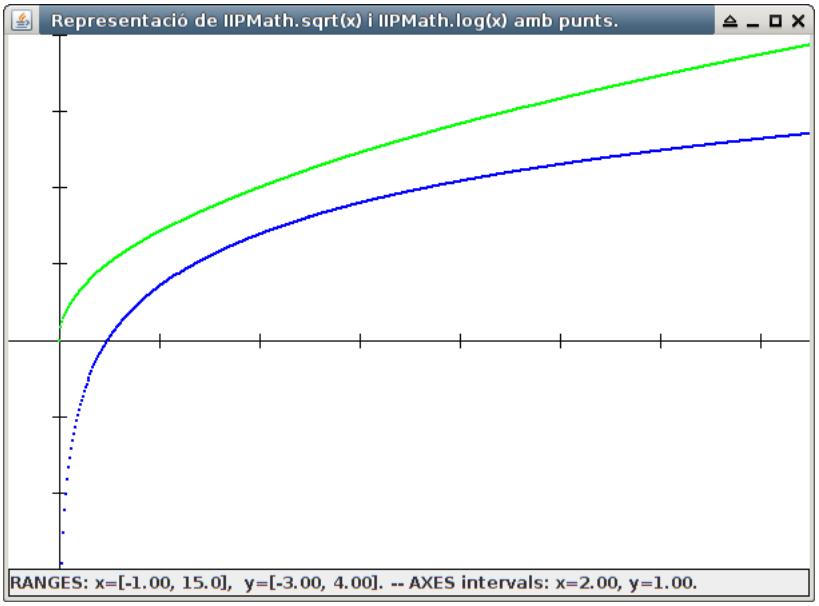

Figura 2: Representación de IIPMath.sqrt(x) e IIPMath.log(x) en $[-1, 15]$.

Actividad #6

Representa gráficamente en el intervalo: $x \in [-1, 15]$ e $y \in [-3, 4]$, mediante líneas, esto es, uniendo cada dos puntos consecutivos de la gráfica mediante una línea, los valores de las dos funciones que has desarrollado.

NOTA: En ambas actividades debes tener en cuenta que la raíz no está definida en $[-1, 0[$ y que el logaritmo no está definido en $[-1, 0]$.

6. Anexo: Ejemplo de uso de la clase `Graph2D`

Como ejemplo, se muestra el uso de `Graph2D`, para representar, la función $sen(x)/x$, como se ha visto en la figura 1.

En primer lugar es necesario al comienzo del fichero Java en el que se escriba el programa, incluir la directiva de importación de la clase gráfica:

```
// Importa la clase Graph2D (en el paquete graphIIP).
import graphIIP.Graph2D;
```

una vez hecho esto, ya es posible definir objetos de dicha clase y operar sobre ellos en la clase que desarrollemos. Por simplicidad, usamos la constructora con menor número de argumentos: valores mínimos y máximos posibles de x y de y. Para facilitar su uso posterior, definimos previamente variables con dichos valores mínimos y máximos, esto es:

```
// Definir intervalo de valores para x y para y:
double xMin =  -1;
double xMax = Math.PI * 4;
double yMin = -1.1;
double yMax = +1.1;
// Crear espacio de dibujo con las dimensiones deseadas:
Graph2D gd1 = new Graph2D(xMin, xMax , yMin, yMax);
// Cambiar el grosor de los elementos a 2 (por defecto es 1):
gd1.setThickness(2);
```

A continuación, recorrer cada `x` posible (en su intervalo de definición) representando gráficamente el punto (x, Math.sin(x)/x). Para ello, incrementar poco a poco cada valor de `x` desde su valor inicial hasta el final. Usamos una variable (`delta`) para mantener el valor de dicho incremento:

```
// Calcular el incremento en cada paso de x (delta):
double delta = (xMax - xMin) / Graph2D.INI_WIDTH;
```

donde la constante `Graph2D.INI_WIDTH` representa el número de puntos existentes inicialmente en el eje de abscisas en la ventana gráfica (400).

Finalmente, recorremos cada `x` posible, calculando para dicho valor la función correspondiente y representando gráficamente dicho par:

```
// Recorrer cada punto en x, calcular f(x) y dibujar (x, f(x)):
for (double x = xMin; x <= xMax; x = x + delta) {
    double y = Math.sin(x) / x;     // y es el valor de la funcion
    gd1.drawPoint(x, y);            // representar (x, y)
}
```

El resultado de la ejecución del código anterior, se muestra a continuación en la figura 3.

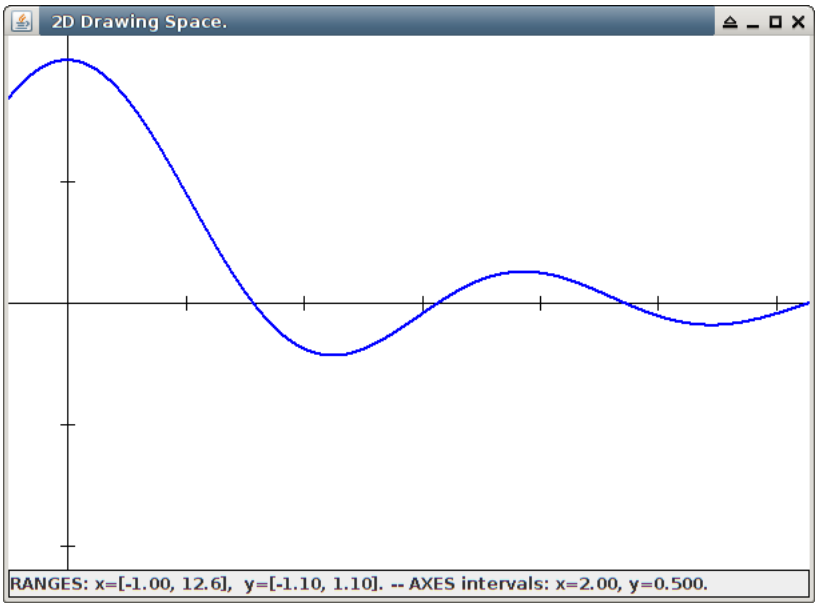

Figura 3: Representación de sen(x)/x en $[-1, 4\pi]$.