UNIVERSIDAD
DE MURCIA

Departamento de Matemáticas

Métodos Numéricos. Práctica 2

En esta práctica se ha creado un paquete `computacion.practica2` en nuestro proyecto `MetodosNumericos` que contiene de momento los ficheros `NumeroMaquina.java`,`EjemploNumeros.java` y `Ejercicio12.java`.

1. Analizar el funcionamiento de la clase `NumeroMaquina.java` con la aplicación `EjemploNumeros.java`. Los objetos de la clase `NumeroMaquina.java` van a ser números de una máquina ideal que trabaja en base 10 con `NumeroMaquina.precision` dígitos en la mantisa. Obsérvese que la clase contiene los métodos correspondientes a las operaciones aritméticas en esa máquina ideal, dos métodos para convertir `NumeroMaquina` en `double` y viceversa, un método para convertir en una cadena de caracteres `String` los números de máquina y dos métodos para el cálculo de errores. La clase `EjemploNumeros.java` contiene las órdenes para fijar el número de dígitos en la mantisa y ejemplos de uso de números de máquina.

A continuación, utilizando objetos de la clase `NumeroMaquina` y trabajando con 4 dígitos de precisión:

(a) Calcular los valores de las siguientes operaciones y los errores absolutos y relativos cometidos: $0.6688 \oplus 0.3334$, $1000 \ominus 0.05001$, $2.000 \otimes 0.6667$ y $25.00 \oslash 16.00$.

(b) Buscar ejemplos de números de máquina a, b, c tales que $(a \oplus b) \oplus c \neq a \oplus (b \oplus c)$.

(c) Buscar ejemplos de números de máquina a, b tales que $a \neq b$ y $1 \oslash a = 1 \oslash b$.

(d) Buscar ejemplos de números de máquina a, b, c tales que $a \otimes (b \oplus c) \neq (a \otimes b) \oplus (a \otimes c)$.

(e) Resolver la ecuación $x^2 + 800x + 1 = 0$ utilizando la fórmula $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$. Evaluar los errores cometidos y recalcular las raíces utilizando fórmulas apropiadas que reduzcan los errores.

2. Crear una clase `EpsilonMaquina.java` en la que se calcule el epsilon la de máquina al trabajar con numeros `double`. Dentro de esta clase construiremos un método `main` en el que vamos a buscar el épsilon de la máquina. Para ello realizaremos un bucle en el que iremos sumando a 1 el número x que comenzará valiendo 1 y se transformará en $x = 0.5 * x$ para la etapa siguiente. También guardaremos en el entero k el número de la etapa del bucle, que empezará valiendo 0 y se transformará en $k = k + 1$. Pararemos el bucle cuando $1. + x$ no sea mayor que 1. En este momento $x = 1/2^k$ será menor que el épsilon de la máquina y $2 * x$ será mayor o igual que éste. Finalizaremos buscando el valor exacto del épsilon de la máquina e imprimiéndolo. Por último construiremos dentro de la clase una variable estática `EPSILON` que contenga el épsilon de la máquina y que pueda ser utilizada desde otras clases de java.

3. Construir una aplicación `SumaArmonica.java` que contenga dos métodos

`double sumaAscendente(int n)` y `double sumaDescendente(int n)`

que evalúen las sumas $H_n = \sum_{j=1}^{n} 1/j$. El método `main` calculará los valores de esas sumas para $n = 1000$, $n = 2000$ y $n = 4000$. ¿Cuál de los dos métodos de suma es más estable? Dar una estimación de los errores cometidos. Dar una aproximación de la constante de Euler $c = \lim_{n\to\infty}(H_n - \log n)$.

4. Crear una aplicación con la que se construya una tabla que refleje los valores del polinomio $(x-1)^3$ escrito utilizando las expresiones

(a) $x^3 - 3x^2 + 3x - 1$,

(b) $-1 + x(3 + x(-3 + x))$

para 20 valores equidistribuidos en el intervalo $[0.999999, 1.000001]$. Comparar los errores cometidos.

5. Utilizando su desarrollo de Taylor construir un algoritmo que evalúe la función $\text{sen}(x)$ con una precisión de 13 dígitos. Evaluar la bondad del método comparándolo con `Math.sin(x)` para distintos valores de x. En el directorio raíz de la versión del proyecto contenida en el fichero comprimido `Practica2preliminar.zip` puede encontrarse el texto `practica2ejercicio5(anexo).pdf` con información adicional sobre este problema.

6. La función error $\text{Ferr}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt$ mide la probabilidad de que el resultado de un ensayo esté a menos de x unidades de la media en el supuesto de ensayos con una distribución normal de medio 0 y desviación típica $\sqrt{2}/2$. Puede demostrarse que las siguientes fórmulas son válidas:

(a) $\text{Ferr}(x) = \dfrac{2}{\sqrt{\pi}} \displaystyle\sum_{k=0}^{\infty} \frac{2^k x^{2k+1}}{k!(2k+1)}$,

(b) $\text{Ferr}(x) = \dfrac{2}{\sqrt{\pi}} e^{-x^2} \displaystyle\sum_{k=0}^{\infty} \frac{2^k x^{2k+1}}{1 \cdot 3 \cdot 5 \cdots (2k+1)}$

Escribir un programa en el fichero `Sumas.java` con un método que con datos de entrada $x > 0$ y *precision* produzca una tabla de dos columnas que en cada fila k contenga las evaluaciones de (a) y (b) al considerar los k primeros sumandos de las series, parando cuando el ultimo sumando de la serie (a) sea menor que *precision*. Usar el mismo fichero (método `main`) para comparar las aproximaciones a $\text{Ferr}(x)$ para $x = 1$, 0.5 y 2 con errores menores que $precision = 10^{-12}$.

7. El número e se puede definir como $e = \sum_{n=0}^{\infty} \frac{1}{n!}$. Calcular los errores absolutos y relativos de las aproximaciones de e

(a) $\sum_{n=0}^{8} \frac{1}{n!}$,

(b) $\sum_{n=0}^{10} \frac{1}{n!}$.

Comparar las observaciones con las acotaciones del resto de Lagrange $e - \sum_{n=0}^{k} \frac{1}{n!} = \frac{e^\theta}{(k+1)!}$.

8. Crear una clase ejecutable en la que vamos a analizar dos métodos distintos para evaluar e^x cuando el exponente es negativo usando sumas parciales del desarrollo de Taylor. Construir los métodos

(a) `sumaExp(double x, int k)`, que devuelve $\sum_{i=0}^{k} x^i/i!$ y por tanto converge a $\sum_{i=0}^{\infty} x^i/i! = e^x$,

(b) `invsumaExp(double x, int k)`, que devuelve

$$\frac{1}{\sum_{i=0}^{k} (-1)^i x^i / i!}$$

y por tanto también converge a $1/e^{-x} = e^x$.

Usando $x = -2$ y $x = 5$, analizar el comportamiento de los dos métodos construyendo una tabla con cuatro columnas en cuya fila k aparezcan los valores correspondientes a `sumaExp(double x, int k)`, al error $|e^x -$ SumaExp(double x, int k)$|$, a `invsumaExp(double x, int k)` y al error $|e^x -$ `invsumaExp(double x, int k)`$|$. ¿Cuál de los dos métodos es más preciso?

9. Crear una aplicación que contenga un método `cambioDeBase` que escriba los números reales en coma flotante y base 8, utilizando 15 dígitos en la mantisa. Escribir la representación de los números π y e en base 8 utilizando el método anterior.

10. Crear una clase ejecutable en la que se considere la sucesión definida de forma recurrente por $a_1 = 4$ y

$$a_{n+1} = 2^{2n+3} \frac{-1 + \sqrt{1 + a_n^2/2^{2n+2}}}{a_n}$$

para cada $n \geq 1$. Escribir un programa que escriba los 30 primeros términos de esta sucesión utilizando la fórmula anterior. Observar los valores encontrados y reescribir la formula recurrente con cálculos más estables.(INDICACIÓN: considerar la sucesión auxiliar $y_n = a_n/2^{n+1}$.) Implementar la nueva fórmula en el programa de manera que escriba los 30 primeros términos de la sucesión utilizando las dos fórmulas y puedan compararse los resultados.

11. Diseñar una clase ejecutable `Ejercicio11` que además del método `main` contenga otro método `int base7(String numbase7)`. Este último recibirá una cadena de caracteres que representa a un cierto entero no negativo en base 7 y devolverá su expresión decimal (o -1 si la cadena no representa ningún número en base 7, por ejemplo porque contiene un carácter que no es un dígito). Ilustrar su uso en el método `main` de la clase.

12. La clase `Ejercicio12.java` contiene dos métodos diferentes para obtener los números combinatorios $\binom{m}{n}$ y al calcular $\binom{29}{14}$ se obtienen resultados diferentes. Añadir unos comentarios aclaratorios a la clase explicando la razón.

13. El objetivo del siguiente ejercicio es entender en profundidad el estándar IEEE-754 de representación en binario de los números en coma flotante. En dicho estándar, cada número en coma flotante con doble precisión (los del tipo `double`) se representa en términos de 64 bits (ceros y unos), de los cuales el primero corresponde al signo, los siguientes once al exponente y el resto a la mantisa. Los detalles pueden consultarse, por ejemplo, en http://babbage.cs.qc.edu/courses/cs341/IEEE-754references.html. Crear una clase ejecutable, `IEEE_754.java`, que contenga los siguientes métodos:

(a) Un método `flotanteAbinario` que reciba como dato un `double` y devuelva un `String` con su representación en binario acorde al estándar IEEE-754. Por ejemplo, si se introduce el `double` -3567.82 habrá de devolver la cadena

1|10000001010|1011110111111010001111010111000010100011110101110001

(nótese que se han añadido barras verticales para enfatizar la separación entre signo, exponente y mantisa).

(b) Un método `binarioAflotante`, recíproco del anterior, que reciba como dato un `String` con la estructura antes descrita y devuelva el `double` al que representa.

(c) Métodos respectivos `anterior` y `siguiente` que reciban un `double` como dato y devuelvan, respectivamente, el `double` anterior y el siguiente al dato (nótese que al ser finita la cantidad de números con que trabaja el ordenador tiene sentido ordenarlos de menor a mayor).

(d) Un método `main` en el que se verifiquen los métodos anteriores y en el que se calculen los doscientos últimos números de la máquina (es decir, los doscientos números positivos más grandes) y los doscientos números positivos más pequeños.