ANALISIS NUMERICO
EXAMEN 1

ALEXIS ADRIAN CARRILLO MEDINA (316733780)

1. Punto Flotante

1.1. Considere el conjunto de elementos $\mathcal{F}(2, 2, -1, 2)$ determine:

Todos los posibles valores representables (normalizados), cardinalidad del conjunto, Xmax y Xmin

Todos los posibles valores(normalizados) son los siguientes:

Exponente	Mantisa (10)	Mantisa (11)
-1	$(.10)_2 \times 2^{-1} = \frac{1}{4}$	$(.11)_2 \times 2^{-1} = (.011)_2 = 0.375 = \frac{3}{8}$
0	$(.10)_2 \times 2^{0} = \frac{1}{2}$	$(.11)_2 \times 2^{0} = (.11)_2 = 0.75 = \frac{3}{4}$
1	$(.10)_2 \times 2^{1} = 1$	$(.11)_2 \times 2^{1} = (1.1)_2 = 1.5 = \frac{3}{2}$
2	$(.10)_2 \times 2^{2} = 2$	$(.11)_2 \times 2^{-1} = (11)_2 = 3$

Ahora, calculemos la cardinalidad, para ello recordemos que

$$||\mathcal{F}(2, 2, -1, 2)|| = 2 \cdot (2 - 1) \cdot 2^{2-1} \cdot (2 - (-1) + 1) = 8$$

Xmin y Xmax

$$X_{min} = 2^{-1-1} = \frac{1}{4}$$
$$X_{max} = 2^2 \cdot (1 - 2^{-2}) = 3$$

Overflow y Underflow (tanto positivo como negativo)

Por definición el *underflow* (ya considerando positivo y negativo) esta dado por el intervalo $(-X_{min}, X_{min}) = (-\frac{1}{4}, \frac{1}{4})$ este incluye al 0 por la notación normalizada
Por definición el *Overflow* (ya considerando positivo y negativo) esta dado por el conjunto $(-\infty, -X_{max}) \cup (X_{max}, \infty) = (-\infty, -3) \cup (3, \infty)$

Épsilon de la maquina

Este esta dado por

$$\varepsilon_M = 2^{1-2} = \frac{1}{2}$$

1.2. Sea $\mathcal{F}(2,3,-1,2), a=\frac{3}{8}, b=\frac{5}{4}$, demuestra que $a \in \mathcal{F}$ y $b \in \mathcal{F}$, pero $a+b \notin \mathcal{F}$

Notemos que

$$\frac{3}{8} = 0.375 = (0.011)_2 = (0.110)_2 \times 2^{-1} \in \mathcal{F}(2,3,-1,2)$$

$$\frac{5}{4} = 1.25 = (1.010)_2 = (0.101)_2 \times 2^{1} \in \mathcal{F}(2,3,-1,2)$$

Por otro lado

$$\frac{3}{8} + \frac{5}{4} = \frac{13}{8} = 1.675$$

y notemos que

$$(.110)_2 \times 2^1 = (1.10)_2 = 1.5$$

$$(.111)_2 \times 2^1 = (1.11)_2 = 1.75$$

Es decir, tenemos 2 números $f = 1.5, g = 1.75$ tales que $f < a+b < g$ y $f_2, g_2 \in \mathcal{F}(2,3,-1,2)$ que son consecutivos en $\mathcal{F}(2,3,-1,2)$, entonces $a+b \notin \mathcal{F}(2,3,-1,2)$

2. Estándar IEEE

2.1. Encuentre la expresión hexadecimal en formato IEEE 754 precisión simple del siguiente valor en base 10

10.5

Convirtamos a binario la parte entera, esto es inmediato

$$10 = (1010)_2$$

Ahora, la parte decimal

$$0.5 = (.1)_2$$

Entonces

$$10.5 = (1010.1)_2 = (.10101)_2 \times 2^4 = (.10101000000000000000000)_2 \times 2^4$$

Por lo tanto tenemos que $e = 4 \Rightarrow E = 126+4 = 130 = (10000010)_2$ y tenemos una mantisa (removiendo el bit implícito) dada por $(.01010000000000000000000)_2$. Por lo tanto, la representación en el estándar IEEE 754 de precisión simple de este numero es:

$$\underbrace{0}_{signo}\ \underbrace{10000010}_{exponente}\ \underbrace{01010000000000000000000}_{mantisa}{}_2$$

y como tenemos las siguientes equivalencias en hexadecimal:

$$(0100)_2 = 4 \quad (0001)_2 = 1 \quad (0010)_2 = 2 \quad (1000)_2 = 8 \quad (0000)_2 = 0$$

Entonces, la expresión en hexadecimal del numero en formato IEEE 754 es

$$\underbrace{0}_{signo}\ \underbrace{10000010}_{exponente}\ \underbrace{01010000000000000000000}_{mantisa}{}_2 = 0 \times 41280000$$

2.2. ¿Cual es el valor en base 10 de la siguiente expresión hexadecimal en formato IEEE 754 precisión simple?

0x41240000

Tenemos las siguientes equivalencias para hexadecimal y binario

$$(0100)_2 = 4 \quad (0001)_2 = 1 \quad (0010)_2 = 2 \quad (0000)_2 = 0$$

Entonces, la representación en IEEE 754 es

$$\underbrace{0}_{signo}\ \underbrace{10000010}_{exponente}\ \underbrace{01001000000000000000000}_{mantisa}{}_2 = 0 \times 41240000$$

Notemos que el numero esta normalizado, entonces $E = (10000010)_2 = 130 \Rightarrow e = 130 - 126 = 4$, por lo que el numero representado en binario es

$$(.10100100000000000000000)_2 \times 2^4 = (1010.01000000000000000000)_2 = (1010.01)_2$$

y notemos que $(1010)_2 = 10$ y $(.01)_2 = 0.25$. Por lo tanto, el numero en decimal es

$$(1010.01)_2 = 10.25$$

3. Error de calculo y redondeo

3.1. Evaluar la función $f(x) = x(\sqrt{x+1} - \sqrt{x})$ para $x = \{20, 1000, 100000\}$

Redondear cada resultado intermedio a 6 dígitos decimales. Calcular el error relativo

x = 20

Calculemos

$$\sqrt{x+1} = \sqrt{21} \approx 4.582575\ldots \approx 4.58258$$
$$\sqrt{x} = \sqrt{20} \approx 4.472135\ldots \approx 4.47214$$

Entonces

$$\sqrt{x+1} - \sqrt{x} = \sqrt{21} - \sqrt{20} \approx 0.11044$$
$$\Rightarrow f(x) = x(\sqrt{x+1} - \sqrt{x}) = 20(0.11044) \approx 2.20880$$

Calculemos el error relativo

$$\left|\frac{x - fl(x)}{x}\right| \approx \left|\frac{2.208794799125212275393997125309120362064397150898409081308952661 7 - 2.20880}{2.20879479912521227539399712530912036206439715089840908130895266 17}\right|$$

$$\approx 2.3546210765184704422120246855524075671676092486189008996 03 \times 10^{-6}$$

x = 1000

Calculemos

$$\sqrt{x+1} = \sqrt{1001} \approx 31.63858\ldots \approx 31.6386$$
$$\sqrt{x} = \sqrt{1000} \approx 31.62277\ldots \approx 31.6228$$

Entonces

$$\sqrt{x+1} - \sqrt{x} = \sqrt{1001} - \sqrt{1000} \approx 0.0158$$
$$\Rightarrow f(x) = x(\sqrt{x+1} - \sqrt{x}) = 1000(0.0158) \approx 15.8$$

Calculemos el error relativo

$$|\frac{x - fl(x)}{x}| \approx |\frac{15.807437428955823117356140473797749884502125728086665201607337228 - 15.8}{15.807437428955823117356140473797749884502125728086665201607337228}|$$
$$\approx 0.000470501875414629083095412939774938896445442386727713402843070 5$$

x = 100000

Calculemos

$$\sqrt{x+1} = \sqrt{100001} \approx 316.2293\ldots \approx 316.229$$
$$\sqrt{x} = \sqrt{100000} \approx 316.2277\ldots \approx 316.228$$

Entonces

$$\sqrt{x+1} - \sqrt{x} = \sqrt{100001} - \sqrt{100000} \approx 0.001$$
$$\Rightarrow f(x) = x(\sqrt{x+1} - \sqrt{x}) = 100000(0.001) \approx 100$$

Calculemos el error relativo

$$|\frac{x - fl(x)}{x}| \approx |\frac{158.113487725687856737572772290971766711927074890181666720600060387 - 100}{158.113487725687856737572772290971766711927074890181666720600060387}|$$
$$\approx 0.367542886831446876721653915385733383538421852864207732811826 9694$$

¿Que sucede conforme x va creciendo en magnitud? Proponga algo para evitar esta problemática

Es fácil ver que mientras x crece el error igual va creciendo. Para solucionarlo, vamos a des-racionalizar

$$x(\sqrt{x+1} - \sqrt{x}) = x(\sqrt{x+1} - \sqrt{x})\frac{\sqrt{x+1} + \sqrt{x}}{\sqrt{x+1} + \sqrt{x}} = \frac{x(x+1-x)}{\sqrt{x+1} + \sqrt{x}} = \frac{x}{\sqrt{x+1} + \sqrt{x}}$$

y veamos que en efecto reduce el error. Teniamos que

$$\sqrt{x+1} = \sqrt{100001} \approx 316.2293\ldots \approx 316.229$$
$$\sqrt{x} = \sqrt{100000} \approx 316.2277\ldots \approx 316.228$$

Entonces

$$\sqrt{x+1} + \sqrt{x} = \sqrt{100001} + \sqrt{100000} \approx 632.457$$
$$\Rightarrow \frac{x}{\sqrt{x+1} + \sqrt{x}} = \frac{100000}{632.457} \approx 158.1135\ldots \approx 158.114$$

Por lo que el error relativo es

$$|\frac{x - fl(x)}{x}| \approx |\frac{158.113487725687856737572772290971766711927074890181666720600060387 - 158.114}{158.113487725687856737572772290971766711927074890181666720600060387}|$$
$$\approx 3.239915326085340324128227001517952059671562286585341907905 \times 10^{-6} < 0.3675\ldots$$

3.2. Suponga que $fl(x)$ es una aproximación de x en base 16 con p cifras significativas y usando redondeo. Encuentra la cota para el error relativo

Consideremos x, un numero real, en base 16. Entonces, podemos representar a x de la siguiente manera

$$x = (0.d_1d_2d_2 \dots d_pd_{p+1} \dots) \times 16^s$$

Caso 1: $d_{p+1} < 8$

Entonces, tenemos que

$$x = (0.d_1d_2d_2 \dots d_pd_{p+1} \dots) \times 16^s \approx fl(x) = (0.d_1d_2d_2 \dots d_p) \times 16^s$$

calculemos el error relativo

$$\left|\frac{x - fl(x)}{x}\right| = \left|\frac{(0.d_1d_2...d_pd_{p+1}...)_{16} \times 16^s - (0.d_1d_2...d_p)_{16} \times 16^s}{(0.d_1d_2...d_pd_{p+1}...)_{16} \times 16^s}\right| = \left|\frac{(0.0.....0d_{p+1}...)_{16} \times 16^s}{(0.d_1d_2...d_pd_{p+1}...)_{16} \times 16^s}\right|$$

$$= \left|\frac{(0.d_{p+1}d_{p+2}...)_{16} \times 16^{s-p}}{(0.d_1d_2...d_pd_{p+1}...)_{16} \times 16^s}\right| = \left|\frac{(0.d_{p+1}d_{p+2}...)_{16}}{(0.d_1d_2...d_pd_{p+1}...)_{16}}\right| \times 16^{-p}$$

Suponiendo valores normalizados, el menor valor que puede adquirir el denominador es $(.100\dots)_16$ y lo mayor que puede valor el numerador es $(0.FFF\dots)$, entonces, esto esta acotado por

$$\Rightarrow \left|\frac{x - fl(x)}{x}\right| = \left|\frac{(0.d_{p+1}d_{p+2}...)_{16}}{(0.d_1d_2...d_pd_{p+1}...)_{16}}\right| \times 16^{-p} < \left|\frac{(0.FFF...)_{16}}{(0.1000...)_{16}}\right| \times 16^{-p} = \left|\frac{1}{16^{-1}}\right| \times 16^{-p}$$

$$= 16^{1-p}$$

Caso 2: $d_{p+1} \geq 8$

Entonces, tenemos que

$$x = (0.d_1d_2d_2 \dots d_pd_{p+1} \dots) \times 16^s \approx fl(x) = (0.d_1d_2d_2 \dots d_p + 16^{s-p}) \times 16^s = (0.d_1d_2d_2 \dots d_p) \times 16^s + 16^{-p}$$

calculemos el error relativo

$$\left|\frac{x - fl(x)}{x}\right| = \left|\frac{(0.d_1d_2...d_pd_{p+1}...)_{16} \times 16^s - ((0.d_1d_2...d_p)_{16} \times 16^s + 16^{-p})}{(0.d_1d_2...d_pd_{p+1}...)_{16} \times 16^s}\right|$$

$$< \frac{1}{2}\left|\frac{(0.d_1d_2...d_p)_{16} \times 16^s - ((0.d_1d_2...d_p)_{16} \times 16^s - 16^{-p}}{(0.d_1d_2...d_pd_{p+1}...)_{16} \times 16^s}\right|$$

$$= \frac{1}{2}\left|\frac{16^{-p}}{(0.d_1d_2...d_pd_{p+1}...)_{16} \times 16^s}\right|$$

Suponiendo valores normalizados, el menor valor que puede adquirir el denominador es $(.100\dots)_16$, entonces, esto esta acotado por

$$\Rightarrow \left|\frac{x - fl(x)}{x}\right| < \frac{1}{2}\left|\frac{16^{-p}}{(0.100\dots)_{16}}\right| \times 16^{-p} < \frac{1}{2}\left|\frac{16^{-p}}{(0.1000...)_{16}}\right| \times 16^{-p}$$

$$= \frac{1}{2}\left|\frac{1}{16^{-1}}\right| \times 16^{-p} = \frac{1}{2}16^{1-p}$$

4. Matrices

4.1. Escribe el código en python para calcular la potencia de una matriz de cualquier dimensión. Solo puede usar la función array() de numpy. Explica a que orden de complejidad computacional pertenece este algoritmo. Justifica tu respuesta

Primero una convención, solo podemos sacar potencia de matrices cuadradas, porque si quisiéramos sacar la potencia de una matriz de otra dimensión, no cuadrada, no podríamos multiplicarlas según la definición porque tendríamos una multiplicación de $m \times n$ por $m \times n$ (lo cual no esta definido) y para multiplicarlas necesitamos que sea $m \times n$ por $n \times s$.

La implementación la podemos observar en el archivo ' examen1(Pregunta5).py' para las pruebas y en 'Matriz.py', que esta en el paquete matrices, para los algoritmos.

En el caso de mi implementación, usamos 2 funciones para calcular: una para calcular la multiplicación con otra matriz abstracta mientras se cumplan los requisitos para multiplicar, la cual tiene 3 ciclos anidados dependiendo de la dimensión de la matriz, entonces dicha función tiene complejidad $O(n^3)$, pues el resto son asignaciones; la otra función es para sacar la potencia, se multiplica con ella misma n veces, entonces tendríamos que para esta función habría 4 ciclos anidados, lo que nos da una complejidad total de $O(n^4)$

4.2. Escribe el código de python para calcular la inversa de una matriz cuadrada. Explica a que orden de complejidad computacional pertenece este algoritmo, justifica tu respuesta

La implementación la podemos observar en el archivo ' examen1(Pregunta5).py' para las pruebas y en 'Matriz.py', que esta en el paquete matrices, para los algoritmos.

Si nos fijamos en el algoritmo de inversa, este tiene varias otros algoritmos que llama:

Lo primero que hace es calcular el determinante, en el peor de los casos tenemos una matriz de n por n, entonces, tenemos que hacer recursion n veces para calcular el determinante de la matriz de menores, entonces primero calculamos la matriz de menores, esto tiene complejidad constante (Podemos ver el algoritmo aqui: https://github.com/numpy/numpy/blob/v1.19.0/numpy/lib/function_base.py#L4236-L4414), entonces lo unico que resta es la recursion, calculamos el determinante ahora de una matriz de $n-1 \times n-1$ entradas, asi hasta que llegamos al caso $n = 2$, lo que nos devuelve una complejidad $O(n!)$.

Posteriormente calculamos la matriz de cofactores, el algoritmo para se basa en 2 ciclos anidados y el resto son asignaciones, por lo que tiene complejidad $O(n^2)$.

por ultimo tenemos 2 ciclos anidados para multiplicar por el determinante lo que nos entrega complejidad $O(n^2)$.
Entonces, sumando todos estos algoritmos al final tenemos una complejidad $O(n! + 2n^2) = O(n!)$