Tropezando con MatLab

1. Introducción

El objetivo de este texto es poner al estudiante de Computación 1 en contacto con problemas que pueden surgir al utilizar la computadora como herramienta de cálculo. Muchos de estos problemas radican en la naturaleza "finita" de la computadora: a pesar de que con el correr del tiempo su capacidad (almacenamiento y procesamiento) ha crecido exponencialmente, siempre existirá un límite tanto en la precisión como en el tamaño de los números que maneja.

Este texto asume que el lector maneja conceptos básicos de programación, y que conoce el entorno Matlab, Octave o Scilab.

2. ¿El oráculo se equivoca?

Es importante estar alerta al momento de modelar la solución a un problema dado, debido a que pequeños errores numéricos pueden conducir a resultados catastróficos. Los ejemplos que veremos a continuación pretenden ilustrar esta situación.

2.1. Problemas

A la hora de efectuar una operación aritmética entre números reales con una computadora, el resultado puede no ser el esperado. Para entender por qué ocurre esto, debemos recordar que la computadora posee una cantidad limitada de bits para representar un número. Esto implica que al efectuar una determinada operación, es posible que parte de la información se pierda en el proceso. Si esto ocurre, la precisión del resultado calculado será menor que la precisión del resultado analítico. En este caso, lo único que podemos hacer es reducir esta pérdida y ser conscientes de que los resultados que obtenemos pueden tener un error.

2.1.1. Suma

Comencemos con un problema sencillo, la suma de números. Analíticamente, es sencillo verificar que $1-x-1+x$ da 0 para cualquier valor de x. Esto lo podemos calcular en Matlab. Ahora bien, sería esperable que al realizar la operación $1-aux-1+aux$ utilizando Matlab, el resultado también fuera siempre igual a cero. Sin embargo, por sorprendente que parezca, esto no siempre es así. Si le asignamos a la variable aux valores al azar, constatamos que el resultado a veces es cero y a veces no. ¿Por qué ocurre esto?
Como se vio en el curso, los números reales son representados en punto flotante: hay una cierta cantidad fija de bits para la mantisa y otra cantidad fija para el exponente. Para realizar una suma los exponentes deben ser iguales y para que esto ocurra es necesario ajustar alguna de las mantisas. Como la cantidad de bits para la mantisa es finita, la pérdida de información es inevitable.

El siguiente ejemplo permite ilustrar el mismo problema desde otro ángulo. En la escuela aprendimos que la suma cumple la propiedad conmutativa: $x + y = y + x$. Sin embargo ¿esta propiedad siempre se cumple al trabajar con números reales en una computadora? Nuevamente, es posible constatar que, para ciertos valores de aux, no es lo mismo $aux - aux + 1 - 1$, que $1 - aux - 1 + aux$ o que $1 + aux - aux - 1$. ¿Como habrán sumado los docentes los puntajes de mi último examen?

Esto implica que el resultado computacional de una suma sencilla puede diferir enormemente del resultado aritmético. Más aun, si cambiamos la forma de la operación puede cambiar el resultado. A continuación se presenta en forma gráfica este fenómeno. En la Figura 1 vemos el resultado de aplicar la operación $aux - aux + 1 - 1$ asignando valores aleatorios a la variable aux. En este caso, en particular vemos que el resultado es el esperado desde el punto de vista aritmético.

Por otro lado la Figura 2 muestra el resultado que se obtiene al cambiar la forma de la operación.

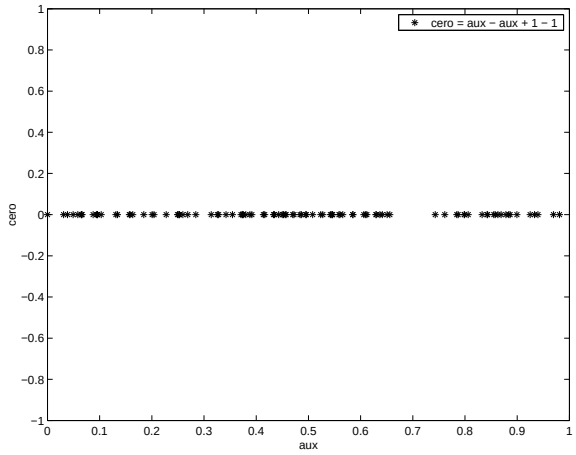
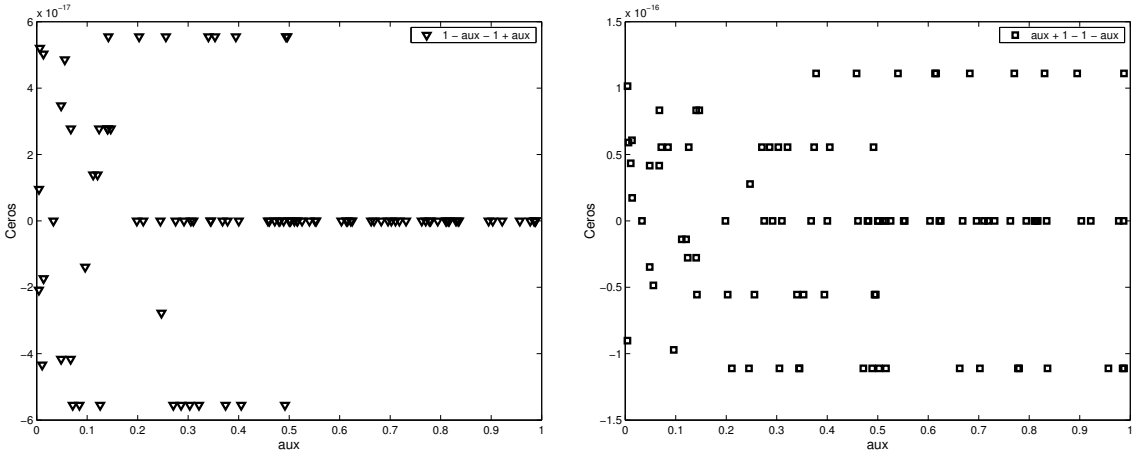


Figura 1: En este caso en particular se cumple la propiedad conmutativa de la suma



a) $1 - aux - 1 + aux$ b) $aux + 1 - 1 - aux$

Figura 2: No se cumple la propiedad conmutativa de la suma

En esta última gráfica se observa que el orden en que se realizan las sumas afecta el resultado (que matemáticamente debería ser cero). También se puede apreciar para cada operación la forma en que se distribuye el error y los valores mínimos y máximos que alcanza.

Este fenómeno se debe en primer lugar a la manera en que MatLab agrupa los terminos de la operación al momento de su evaluación. En segundo lugar, a cómo se realizan las operaciones en punto flotante. En la Figura 1 los términos se agrupan de la siguiente manera: $(((aux - aux) + 1) - 1)$. La primera suma da cero. Luego, Matlab suma 1 y finalmente suma -1. En este caso no fue necesario el ajuste de la mantisa. Sin embargo, en el caso utilizado para generar la Figura 2, primero se suma $1 - aux$. Dependiendo del valor de aux, puede ser necesario ajustar la mantisa y que se pierda precisión.

2.1.2. Sen(x)

El error numérico también puede ocurrir en otro tipo de operaciones: por ejemplo, en el cálculo del seno. La definición de esta función nos dice que $\text{sen}(\pi) = 0$ y $\text{sen}(\frac{\pi}{2}) = 1$. En general, $\text{sen}(n\pi) = 0$ y $\text{sen}((n + \frac{1}{2})\pi) = 1, \quad \forall\, n \in \mathbb{N}$. ¿Qué sucede al utilizar la función provista por Matlab para calcular el seno? El Cuadro 1 muestra los valores de $\text{sen}(n\pi)$ que devuelve esta herramienta para distintos valores de n.

El cuadro pone en evidencia la diferencia entre el resultado calculado y el que esperaríamos obtener. El problema se debe a que π no puede ser representado exactamente. Por lo tanto, el resultado de la función $\text{sen}\,\pi$ es el resultado de calcular el seno de un número próximo a π. No es la función seno la que esta fallando: nuevamente,

n	sen($n\pi$)
0	0
10^0	$1{,}2246 \times 10^{-16}$
10^1	$-1{,}2246 \times 10^{-15}$
10^2	$1{,}9644 \times 10^{-15}$
10^4	$-4{,}8568 \times 10^{-13}$
10^6	$-2{,}2319 \times 10^{-10}$
10^{20}	0,70511

Tabla 1: Valores de seno para múltiplos enteros de π

el problema es el límite en la precisión. Al multiplicar π por un entero cada vez más grande el error se amplifica, ya que el cálculo de $n\pi$ dista cada vez más de ser un múltiplo de π.

Este tipo de error podría comprometer seriamente el resultado de un determinado trabajo. Por ejemplo, al modelar alguna situación física que demande una variable angular, como el modelado de un motor a combustión.

Una solución a este problema es normalizar el término a ser evaluado. En este caso, trabajar siempre entre 0 y 2π.

2.1.3. El software del cajero

Para finalizar, este último ejemplo muestra cómo el límite en la precisión numérica puede afectar el comportamiento de un cajero automático, si el programador no fue lo suficientemente cuidadoso a la hora de programar el mecanismo de devolución de dinero.

El problema a resolver es el siguiente: se quiere implementar un programa que entregue dinero utilizando la menor cantidad de billetes. En Matlab, el problema podría modelarse como una función que tome como parámetros la cantidad de dinero a entregar. Como forma de generalizar el tipo de valores que administra el cajero, la función recibe también un vector cuyas entradas son los distintos tipos de billetes disponibles, ordenados de mayor a menor. Un ejemplo de este vector, al que podría llamarse vector de configuración, es el siguiente: [1000, 500, 200, 100, 50, 10, 5, 1]. La función devuelve otro vector conteniendo la cantidad de billetes a devolver, respecto a cada posición del vector de configuración. Continuando con el ejemplo, si la función retorna [2, 0, 3, 0, 0, 0, 0, 0] significa que el cajero deberá entregar dos mil seiscientos pesos en dos billetes de mil y tres de doscientos.

Para este problema se propone como posible solución el siguiente programa recursivo, implementado en Matlab (Código 1):

```
function c=menorcantidad1(N,v)
%asumiendo que existen billetes y monedas para generar el vuelto

if abs(N) == 0 %da error
    c=zeros(1,length(v))
elseif N >=v(1)
    c=[floor(N/v(1)),menorcantidad1(N-(floor(N/v(1))*v(1)),v(2:length(v)))]
else
    c=[0,menorcantidad1(N,v(2:length(v)))]
end
```

Código 1, ¿a qué se debe la ejecución incorrecta?

El programa asume que la cifra ingresada como parámetro puede ser representada como una combinación lineal de los elementos del vector de configuración. Si la cifra N es mayor que el primer valor del vector de configuración, la función divide N por el monto del primer billete y se queda con la parte entera. En caso contrario se queda con cero. Ese valor se agrega al principio vector de salida, el cual se concatena con la llamada recursiva a la función. Esta llamada se realiza con la nueva cifra y con el vector de billetes sin el primer billete. De esta

forma, continúa hasta llegar al paso base.

Si este proceso lo realizáramos de forma manual o analítica llegaríamos a que el paso base debería ser para N igual a cero. En otras palabras, desde el punto de vista analítico el programa funciona correctamente. Sin embargo la ejecución de este programa por una computadora puede fallar. Esto se debe a que las operaciones que se realizan pierden precisión.

El Cuadro 2 presenta los valores de N a medida que avanza la recursión, para un vector de configuración $v = [1\ 0{,}5\ 0{,}1]$ y $N = 2{,}20$. Se observa que, en el paso 4, cuando se debería alcanzar el paso base, N no es estrictamente cero, sino que es igual a $1{,}665 \times 10^{-16}$.

Paso	N	Vector
1	2.20	[1 0.5 0.1]
2	0.20	[0.5 0.1]
3	0.20	[0.1]
4	$1{,}665 \times 10^{-16}$	[]

Tabla 2: Valores de N y vector de billetes para cada paso, en el Código 1

Luego del paso 4, N debería tomar el valor cero. Sin embargo debido a las operaciones realizadas obtenemos un número muy pequeño pero que no es cero. Esto lleva al programa a pedir la posición cero del vector v. Como no existe, es aquí donde se detiene la ejecución del programa y no se obtiene el resultado esperado. Cabe señalar que no siempre ocurre este error: si N hubiese sido 5 no se habría detenido la ejecución del programa y se hubiera obtenido el resultado esperado.

Teniendo en cuenta la situación anterior se presenta el Código 2:

```
function c=menorcantidad1(N,v)
%asumiendo que existen billetes y monedas para generar el vuelto

if abs(N) <= 0.00001
    c=zeros(1,length(v))
elseif N >=v(1)
    c=[floor(N/v(1)),menorcantidad1(N-(floor(N/v(1))*v(1)),v(2:length(v)))]
else
    c=[0,menorcantidad1(N,v(2:length(v)))]
end
```

Código 2, este nuevo paso base permite que la ejecución se correcta.

Este nuevo código garantiza que se alcance el paso base al verificar que N se encuentre en un cierto entorno de cero. Notar que N en el paso base podría llegar a tomar un valor negativo.

¿El Código 2 resuelve el problema? Si consideramos otro ejemplo donde $N = 0{,}30$, el Código 2 falla por otro motivo. Esto se debe a que la operación `floor(0.3/0.1)` da como resultado 2, en vez de 3, el valor deseado. Lo que sucede es que, nuevamente, el resultado de la división no es exactamente 3 (sino algo menor que 3). Si ejecutamos el siguiente comando en MatLab:

`>> 0.3/0.1 -3`,

el resultado es

`ans = -4.440892098500626e-016`.

No podemos concluir a priori si el problema ocurre al dividir, si ocurre al representar los decimales, o si ocurren ambos. Una posible solución sería evitar trabajar con valores en punto flotante, por ejemplo, trabajar con

números enteros.

3. Conclusiones

A lo largo de las distintas carreras y profesiones nos encontraremos con distintos lenguajes de programación. Sin embargo el estudiante debe comprender que lo importante no es el lenguaje en sí mismo. Lo importante es saber cómo modelar el problema y elaborar las diferentes estrategias de programación, cómo y en qué situaciones emplearlas.

Finalmente, debe quedar claro que la computación es solamente una herramienta más, que es necesario saber utilizar criteriosamente y ser conscientes que pueden existir diferentes problemas al utilizarla.