Actividades 1

Prácticas de Cálculo Numérico I (doble grado)

15 de febrero de 2022

1. Un par de sorpresas con el épsilon máquina

La precisión de un ordenador con los cálculos aritméticos básicos es tan grande en comparación con la nuestra que estamos acostumbrados a que no tenga ningún efecto significativo sobre el resultado de nuestros programas. Sin embargo, hay que tomar esta idea con algo de precaución porque algunos cálculos pueden agrandar esa pequeña imprecisión intrínseca dada por el épsilon máquina hasta hacerla bien notoria. Vamos a verlo a través de dos ejemplos, el primero está adaptado de [QS07] y el segundo es clásico.

Consideremos la gráfica de un polinomio en apariencia inocente con un resultado rarísimo:

```
1  x = linspace(0.99999,1.00001,100);
2  y = x.^5 - 5*x.^4 + 10*x.^3 - 10*x.^2 + 5*x - 1;
3  plot(x,y)
```

No es problema de la discretización. De hecho, al incrementar `100`, la situación empeora. Además, cambiando cualquiera de los signos de la línea 2, el dibujo sí responde a la realidad del nuevo polinomio.

La explicación es que, simbólicamente (con precisión infinita), se tiene $y = (x-1)^5$, pero en nuestro intervalo la x se mueve en 1 ± 10^{-5} y como $\left(10^{-5}\right)^5$ supera el épsilon máquina, `matlab/octave` no es capaz de detectar la cancelación.

Actividad 1.1.1. *Trata de dar más detalles en la explicación anterior. Para ver que lo has entendido, prevé cuál debe ser el orden de la anchura del intervalo alrededor de 1 para empezar a ver cosas raras y comprueba tu predicción.*

Actividad 1.1.2. *Experimenta con lo indicado acerca de la discretización y los cambios de signo, explicando cualitativamente los resultados.*

El segundo ejemplo, tiene que ver con un algoritmo para aproximar π. Si llamamos a_n a la longitud de la mitad del lado del polígono regular de 2^{n+1} lados circunscrito en la circunferencia unidad entonces su semiperímetro es $2^{n+1}a_n$, por tanto $2^{n+1}a_n \to \pi$ y la geometría de la situación sugiere que la convergencia es muy rápida. Para que este método sea útil, deberíamos buscar algún algoritmo para calcular a_n. Claramente $a_n = \tan\left(\pi/2^{n+1}\right)$ y por las fórmulas del ángulo doble con $\alpha = \pi/2^n$

$$a_n = \frac{1-\cos\alpha}{\operatorname{sen}\alpha} = \frac{\frac{1}{\cos\alpha}-1}{\frac{\operatorname{sen}\alpha}{\cos\alpha}} = \frac{\sqrt{1+\tan^2\alpha}-1}{\tan\alpha} = \frac{\sqrt{1+a_{n-1}^2}-1}{a_{n-1}}.$$

Esta fórmula permite calcular a_n de manera recurrente a partir de $a_1 = 1$.

Escribamos dos variantes del algoritmo, siempre partiendo de $a_1 = 1$:

$$a_n = \frac{\sqrt{1+a_{n-1}^2}-1}{a_{n-1}} \qquad \text{y} \qquad a_n = \frac{a_{n-1}}{\sqrt{1+a_{n-1}^2}+1}.$$

Ambas fórmulas son idénticas racionalizando. Sin embargo nos empeñamos en distinguirlas en el siguiente programa llamando `an` a los resultados de la primera fórmula y `anp` a los de la segunda. Cuando lo ejecutamos, vemos que en el quinto término aparece una pequeña divergencia prácticamente inapreciable. El problema es cómo evoluciona.

```
1  an = 1;
2  anp = 1;
3  for k = 2:28
4       an = (sqrt(1+an^2)-1)/an;
5       anp = anp/(sqrt(1+anp^2)+1);
6       res = 2^(k+1)*[an,anp];
7       disp(['Iteración ' num2str(k) ' -> ' num2str(res, ' %.10f')])
8  end
```

El primer algoritmo a partir de un momento dado empieza a hacer cosas raras mientras que el segundo funciona perfectamente, a pesar de que las fórmulas son formalmente idénticas. Concretamente la salida de las cinco últimas iteraciones es:

```
Iteración 24 -> 3.1110567880 3.1415926536
Iteración 25 -> 3.0536247479 3.1415926536
Iteración 26 -> 2.6198372952 3.1415926536
Iteración 27 -> 3.0536247479 3.1415926536
Iteración 28 -> 0.0000000000 3.1415926536
```

ACTIVIDAD 1.1.3. *Da una explicación para esta diferencia tan drástica entre los algoritmos.*

ACTIVIDAD 1.1.4. *Practica con las funciones en* `matlab/octave` *sustituyendo las líneas 4 y 5 por* `an = alg1(an)` *y* `anp = alg2(anp)` *donde* `alg1` *y* `alg2` *están en ficheros separados.*

ACTIVIDAD 1.1.5. *Estudia experimentalmente el error en el algoritmo "bueno" y elabora una teoría que explique el resultado.*

2. Resolución de sistemas con `matlab/octave`

Si lo único que queremos es resolver con `matlab/octave` un sistema de ecuaciones lineal compatible determinado $A\vec{x} = \vec{b}$ con A una matriz (real o compleja) no singular, todo lo que tenemos que hacer es utilizar `A\b` donde `A` y `b` corresponden a A y $\vec{b}$ de la fórmula anterior. Por ejemplo,

```
1 A = [2,1; 1,-2];
2 b = [4;-3];
3 A\b
```

muestra como salida el vector columna `(1,2)`. Si te estás preguntando de dónde viene esa notación tan rara de la barra de división invertida (llamada *backslash*), te tranquilizará saber que la manera natural de definir la división de matrices B/A es BA^{-1} y para la división "al otro lado" $A^{-1}B$ no suena tan extraño cambiar en la notación la dirección de la barra.

El comando funciona incluso si `b` tiene varias columnas, lo que permite resolver varios sistemas que comparten matriz, como se requiere en algunas aplicaciones. Incluso ofrece un resultado cuando la matriz `A` is rectangular, si el número de filas coincide con el de `b`. En esa situación el sistema puede ser incompatible. ¿Qué significa la solución de un sistema que tiene solución? Lo que muestra es un vector que se desvía lo menos posible, en cierto sentido, de una solución. Consideremos un sistema compatible indeterminado y otro incompatible:

```
1 A = [1,2,3;1,2,4];
2 b = [1;2];
3 A\b
4
5 A = [1,2; 1,1; 3,1];
6 b = [1;1;1];
7 A\b
```

En el primer caso hay infinitas soluciones dadas por $(-2-2t, t, 1)$ y la que muestra `matlab/octave` es la de de norma mínima `(-0.4, -0.8, 1)`. En el segundo caso, la salida es una aproximación de `(1/5, 7/15)`, que es el vector que al ser multiplicado (en columna) por `A` se queda más cerca de `b`.

Una pregunta muy natural es qué sentido tiene que exista la notación rara `A\b` cuando podríamos escribir simplemente `inv(A)*b` o `A^-1*b` donde `inv(A)` y `A^-1` son dos formas de escribir la inversa. La respuesta, es que se da la paradoja de que en general es más costoso hallar una inversa que resolver un sistema y con `A\b` le avisamos a `matlab/octave` que nos da igual no conocer la inversa y se centra en los algoritmos, más rápidos, para resolver sistemas sin inversas.

De entre esos algoritmos, el que aprendimos todos en primero es la reducción de Gauss. En `matlab/octave` el comando `rref` (abreviatura de *reduced row echelon form*) produce la forma escalonada reducida de una matriz. Es decir, lo que se llama reducción de Gauss-Jordan [HVZ12] (continuar el método de Gauss hasta que la matriz formada por las columnas pivote sea la identidad. Si A es una matriz invertible, la forma escalonada reducida de $(A|I)$ es $(I|A^{-1})$, con I la matriz identidad, por tanto es posible hallar inversas indirectamente con el comando `rref`.

ACTIVIDAD 1.2.1. *Calcula la inversa de* `[1,2,-1; 2, 5, 1; -1,1,11]` *usando Gauss-Jordan con* `rref` *y con el comando directo* `inv`.

3. La descomposición LU con `matlab/octave`

Una forma de presentar la reducción de Gauss es la factorización o descomposición LU. En `matlab/octave` se obtiene con el comando `lu`, pero hay que tener en cuenta que el resultado no corresponde a la factorización que habrás visto en primer lugar en la clase de teoría, sino que es la descomposición LU con pivotaje. Por ejemplo:

```
1 A = [1,2,1;3,4,5;5,8,1];
2 [L,U] = lu(A);
3 disp('L =')
4 disp(L)
5 disp('U =')
6 disp(U)
7 disp(L*U)
```

Es verdad que `L*U` recupera `A`, es una verdadera factorización, pero `L` no es triangular inferior. La razón es que lleva incorporada la matriz de permutación que corresponde a los pivotes empleados.

Si queremos que `L` sea realmente triangular inferior, debemos separar la permutación permitiendo un tercer argumento de salida de `lu`:

```
1 A = [1,2,1;3,4,5;5,8,1];
2 [L,U,P] = lu(A);
3 disp('L =')
```

```
4  disp(L)
5  disp('U =')
6  disp(U)
7  disp(L*U)
8
9  disp(P*A)
```

En breve, el resultado de `lu` con tres argumentos de salida, es la factorización LU de `A` con las filas reordenadas. Tal reordenación es interesante desde el punto de vista de la eficiencia numérica cuando se trabaja con matrices grandes.

ACTIVIDAD 1.3.1. *Busca dos ejemplos de matrices* 2×2 *de forma que* `[L,U] = lu(A)` *produzca una matriz triangular superior* `L` *para la primera y no para la segunda. Si no tienes suerte probando al azar, te ayudará saber, o recordar, que lo de "pivotaje" se refiere al tamaño relativo de los pivotes al aplicar reducción de Gauss.*

4. Programando la descomposición LU básica

Nuestro propósito ahora es hacer nuestro propio código que para efectuar la descomposición LU "pura", esto es, sin pivotes. El plan ahora es simplemente traducir el pseudocódigo visto en la clase de teoría. Más adelante, lo refinaremos un poco. La traducción es totalmente inmediata salvo recordar o aprender que `size` con un argumento extra indica solo la dimensión seleccionada. Es decir, si `A` es una matriz 20×21 entonces `size(A)` es `[20,21]`, `size(A,1)` es 20 y `size(A,2)` es 21.

```
1  A = [1,2,1;3,4,5;5,8,1];
2  n = size(A,1);
3  U = A;
4  L = eye(n);
5
6  for k = 1:n-1
7      for ii = k+1:n
8          L(ii,k) = U(ii,k)/U(k,k);
9          for j = k:n
10             U(ii,j) = U(ii,j) - L(ii,k)*U(k,j);
11         end
12     end
13 end
14
15 disp('L =')
16 disp(L)
17 disp('U =')
18 disp(U)
19 disp(L*U)
```

Utilizar `ii` en vez de `i` es solo una manía mía para no sobreescribir el valor `i`$= \sqrt{-1}$ de `matlab/octave`. Con la matriz de prueba de la primera línea,

el resultado debe ser

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 5 & 1 & 1 \end{pmatrix} \qquad \text{y} \qquad U = \begin{pmatrix} 1 & 2 & 1 \\ 0 & -2 & 2 \\ 0 & 0 & -6 \end{pmatrix}.$$

Multiplicando estas dos matrices obtenemos `A` sin que medie ninguna matriz de permutación. Esta comprobación es la razón de ser de la última línea del código.

Referencias

[HVZ12] E. Hernández, M. J. Vázquez, and M. A. Zurro. *Álgebra lineal y geometría*. Pearson, Madrid, 2012. 3a ed.

[QS07] A. Quarteroni and F. Saleri. *Cálculo Científico con MATLAB y Octave*. Springer, Milan, 2007.