Actividades 6

Prácticas de Cálculo Numérico I (doble grado)

29 de marzo de 2022

1. Soluciones de mínimos cuadrados

Una de las aplicaciones más interesantes de la descomposición QR es que permite resolver aproximadamente sistemas incompatibles, es decir, que no tienen solución. Ya vimos que la barra invertida de `matlab/octave` lo hacía. Cambiando un poco el ejemplo visto entonces,

```
1 A = [1,2; 1,1; 3,1];
2 b = [2;8;6];
3
4 disp('solución matlab/octave')
5 A\b
```

da como salida el vector $(2,1)^t$ a pesar de que $A\vec{x} = \vec{b}$ no tiene solución.

La clave teórica es que, usando propiedades de la proyección ortogonal, si $A \in \mathcal{M}_{n\times m}(\mathbb{C})$ con $m < n$ tiene rango máximo (igual a m), entonces

$$\|\vec{b} - A\vec{x}\| \quad \text{se minimiza para} \quad \vec{x} = (A^\dagger A)^{-1} A^\dagger \vec{b}.$$

Con la norma usual, esta es la *solución de mínimos cuadrados*. La matriz $(A^\dagger A)^{-1}A^\dagger$ funciona como una especie de "inversa aproximada" de la matriz no cuadrada A. Se la conoce como la *inversa de Moore-Penrose*.

El punto a destacar es que la descomposición QR simplifica estos cálculos. Para A como antes, la descomposición QR completa será de la forma

$$A = QR \quad \text{con} \quad Q \in \mathcal{M}_{n\times n}(\mathbb{C}) \quad \text{y} \quad R = \begin{pmatrix} T \\ O \end{pmatrix} \in \mathcal{M}_{n\times m}(\mathbb{C}),$$

donde $T \in \mathcal{M}_{m\times m}(\mathbb{C})$ es triangular superior. Entonces, usando $Q^\dagger Q = I$,

$$(A^\dagger A)^{-1}A^\dagger \vec{b} = (R^\dagger R)^{-1} R^\dagger Q^\dagger \vec{b} = (T^\dagger T)^{-1} T^\dagger Q_m^\dagger \vec{b} = T^{-1} Q_m^\dagger \vec{b}$$

donde Q_m son las m primeras columnas de Q. Invertir una matriz triangular superior es muy sencillo. Incluso pasando este importante punto por alto,

las tres expresiones anteriores para $(A^\dagger A)^{-1}A^\dagger\vec{b}$ constituyen simplificaciones del cálculo directo. El siguiente código recoge estas posibilidades:

```
A = [1,2; 1,1; 3,1];
b = [2;8;6];

disp('solución matlab/octave')
A\b

[Q,R] = qr(A);
m = size(A,2);
T = R(1:m,:);
Qm = Q(:,1:m);
% Solución mín. cuad -> inv(A'*A)*A'*b

disp('solución QR simplificada 1')
inv(R'*R)*R'*Q'*b

disp('solución QR simplificada 2')
inv(T'*T)*T'*Qm'*b

disp('solución QR simplificada 3')
inv(T)*Qm'*b
```

Por supuesto, lo más eficiente es calcular $T^{-1}Q_m^\dagger\vec{b}$ mediante sustitución regresiva como solución del sistema $T\vec{x} = Q_m^\dagger\vec{b}$. Recuerda que la resolución de sistemas con matrices triangulares superiores por medio de sustitución regresiva fue parte de una actividad anterior. Por si no tienes el código a mano, una posibilidad es:

```
function x = utrs(A, b)
    % Resuelve sistema con matriz triangular superior
    n = size(A,1);
    x = zeros(n, 1);
    x(n) = b(n)/A(n,n);
    for ii = n-1:-1:1
        x(ii) = (b(ii) - A(ii,(ii+1):n)*x((ii+1):n)
            ↪ )/A(ii,ii);
    end
end
```

El uso de la descomposición QR para hallar soluciones aproximadas de sistemas incompatibles no es un artificio académico. Si miras la documentación o [QS07, §5.6], verás que la barra inversa de `matlab` opera de esta forma en los casos incompatibles.

Actividad 6.1.1. *Escribe un código eficiente que halle la solución de mínimos cuadrados llamando a* `utrs`.

Si m es muy pequeño, puede no merecer la pena meterse en la descomposición QR. Un ejemplo destacado con $m = 2$ es la recta de regresión [DLH03]. La situación es que tenemos una lista de datos formando un vector $\vec{y} \in \mathbb{R}^n$ que queremos ajustar linealmente con unos datos de partida $\vec{x}$. Es decir, buscamos un a y un b tales que y_i sea aproximadamente $a + bx_i$,

la recta $y = a + bx$ es la *recta de regresión*. Introduciendo la matriz X de dimensiones $n \times 2$ que tiene como primera columna unos y como segunda columna $\vec{x}$, lo que buscamos es un $\vec{\beta} \in \mathbb{R}^2$ tal que $\vec{y} \approx X\vec{\beta}$ con $\vec{\beta} = (a, b)^t$. Según lo anterior, la solución de mínimos cuadrados es $\vec{\beta} = (X^tX)^{-1}X^t\vec{y}$.

El siguiente código hace el cálculo y muestra gráficamente el resultado sobre un ejemplo en el que $\vec{y}$ es una perturbación del doble de $\vec{x}$.

```
1  N = 6
2  x = (1:N)';
3  y = 2*x + 4*rand(N,1);
4
5  n = size(x,1);
6  X = [ones(n,1), x];
7
8  c = inv(X'*X)*X'*y;
9
10 plot(x,y,x,c(1)+c(2)*x)
```

ACTIVIDAD 6.1.2. *Escribe una función* `rrerr` *que dados dos vectores de datos* `x` *e* `y` *devuelva el error absoluto máximo, sin signo, cuando se aproxima* `y` *por la recta de regresión. ¿Se cumple* `rrerr(x,y)=rrerr(y,x)` *en general?*

2. SVD y optimización

Recuerda de la teoría que la descomposición en valores singulares SVD (por *Singular Value Decomposition*) de una matriz A es la factorización $A = UDV^\dagger$ donde U y V son matrices unitarias (su inversa coincide con su traspuesta conjugada, indicada mediante $\dagger$) y D es una matriz cuyos únicos elementos no nulos son $d_{ii} \geq 0$, estos son los llamados *valores singulares* que dan nombre a la descomposición. En `matlab/octave`, el comando `svd`, utilizado como en el siguiente ejemplo, permite obtener la SVD.

```
1  A = rand(10,7);
2  [U,D,V] = svd(A);
3  norm(A-U*D*V','fro')
```

Como es evidente por la primera línea, la descomposición no requiere que la matriz sea cuadrada. La última línea es la comprobación de que el error es pequeño. Se ha escogido la norma de Frobenius solo para recordar el comando.

Si se llama a `svd` sin recuperar la salida con tres matrices, se obtiene solo el vector de valores singulares. Así las líneas 4 y 7 del siguiente código muestran el mismo resultado.

```
1  A = rand(3,5);
2  [U,D,V] = svd(A);
```

```
3  disp('Completa')
4  disp( diag(D) )
5  disp('Solo valores singulares')
6  v = svd(A);
7  disp(v)
```

Parte del interés de la SVD es que permite resolver algunos problemas de optimización. En particular, en la teoría has visto que la solución de mínimos cuadrados de un sistema incompatible $A\vec{x} = \vec{b}$ viene dada por

$$\vec{x} = \sum_{k=1}^{r} \sigma_k^{-1} \langle \vec{b}, U^{(k)} \rangle V^{(k)}$$

donde r es el rango, el superíndice (k) indica la columna k-ésima y σ_k son los valores principales. El producto escalar es el usual, definido en el caso complejo con el convenio del álgebra lineal. Si $A \in \mathcal{M}_{n\times m}(\mathbb{C})$ con $r = m < n$, la situación típica en la que nos habíamos fijado, esto es lo mismo que $VE(U_m)^\dagger \vec{b}$ donde E es la matriz diagonal $m \times m$ que tiene $e_{ii} = \sigma_i^{-1}$ y U_m indica que nos quedamos solo con las primeras m columnas. Veamos en un ejemplo que se obtiene lo mismo que ya obtuvimos con QR o con la barra de `matlab/octave`.

```
1  A = [1,2; 1,1; 3,1];
2  b = [2;8;6];
3  [n,m] = size(A);
4  [U,D,V] = svd(A);
5  E = diag( diag(D) );
6  V*inv(E)*U(:,1:m)'*b
7  A\b
```

Las dos últimas líneas producen una salida idéntica, $(2, 1)^t$. Invertir `E` no le cuesta esfuerzo a `matlab/octave` porque sabe que es una matriz diagonal. Si trabajáramos con un lenguaje de programación sin esta estructura de datos, lo eficiente será multiplicar las columnas de V con los σ_k^{-1}.

ACTIVIDAD 6.2.1. *Comprueba si lo anterior da el mismo resultado que* `A\b` *también cuando trabajamos con números complejos. Para ello, parte de una matriz* `A` *aleatoria compleja* `2N` $\times$ `N` *y de un vector* `b` *de dimensión* `2N` *y estudia si ambos resultados tienen una diferencia despreciable.*

Otra aplicación de la SVD relacionada con la optimización es el *teorema de Eckart-Young* (a veces se añade el nombre de Mirsky) que afirma que si tenemos una matriz A, la mejor aproximación A_r de rango $r \leq \mathrm{rg}(A)$, en el sentido de que la norma de Frobenius de $A - A_r$ es mínima, se obtiene mediante $A_r = \sum_{k=1}^{r} \sigma_k U^{(k)} (V^{(k)})^\dagger$. Esto es lo mismo que anular en la SVD de A todos los valores singulares de índice mayor que r. Normalmente el

resultado se enuncia para matrices reales pero el caso complejo es similar y está esencialmente hecho en [Mir60].

El siguiente código muestra un ejemplo:

```
A = [1,2,3; 1,1,2; 3,1,4; 3,1,4];

r = 2;
[U,D,V] = svd(A);
for k=r+1:min(size(A))
        D(k,k)=0;
end

Ar = U*D*V';

disp(Ar)
disp(norm(A-Ar,'fro'))
```

Como en este ejemplo la matriz de partida tiene rango 2, se obtiene `Ar` igual a `A`. Si cambiamos la tercera línea por `r = 1`, obtendremos una matriz con todas sus filas proporcionales.

ACTIVIDAD 6.2.2. *Para una matriz real aleatoria de tamaño* 100×200 *fijada, dibuja una gráfica que muestre cómo cambia la norma de Frobenius de* $A - A_r$ *cuando* $1 \leq r < 100$.

Un tema relacionado es que, en este mismo sentido, si $A = UDV^\dagger$ es la SVD de una matriz cuadrada A, entonces $UV^\dagger$ es la matriz unitaria que mejor aproxima a A, de nuevo en el sentido de la norma de Frobenius. El siguiente código ejemplifica esto gráficamente. Se dibuja la circunferencia unidad centrada en $(2, 0)$, se le aplica una transformación lineal que la convierte en una elipse y $UV^\dagger$ es un giro que superpone la circunferencia y la elipse.

```
A = [0.546, -0.722; 0.783, 0.427];
t = linspace(0,2*pi,300);
x = [2+cos(t); sin(t)];
y = A*x;

[U,D,V] = svd(A);
z = U*V'*x;

figure(1)
plot(x(1,:), x(2,:))
hold on
plot(y(1,:), y(2,:))
hold on
plot(z(1,:), z(2,:))
hold off
axis('equal')
```

ACTIVIDAD 6.2.3. *Modifica el código anterior para que la figura de partida sea el cuadrado definido por la frontera de* $[2, 3] \times [0, 1]$.

Las matrices U y V de la SVD tienen también una interpretación natural

dentro de la teoría básica del álgebra lineal. Limitándonos al caso $A \in \mathcal{M}_{m\times n}(\mathbb{C})$ con $m < n$ y rango igual a m, las m columnas de U forman una base ortonormal de la imagen y las $n - m$ últimas columnas de V forman una base ortonormal del núcleo. El siguiente código comprueba estas afirmaciones sobre una matriz compleja aleatoria.

```
M = 4;
N = 7;
% Matriz aleatoria compleja MxN
A = rand(M,N) + i*rand(M,N);
[U,D,V] = svd(A);
% Últimas N-M columnas de V
disp( 'Base ortonormal del núcleo' )
disp( V(:,M+1:N) )
% Comprobación de que están en el núcleo
% y de que son ortonormales
disp(' Estas cantidades deben ser pequeñas' )
disp( norm( A*V(:,M+1:N) ) )
disp( norm(V(:,M+1:N)'*V(:,M+1:N)-eye(N-M)) )
```

En la teoría no has visto ningún algoritmo eficiente para calcular la SVD que podamos implementar (si tienes curiosidad, mira [SB93, §6.7]). Sin embargo, sí te han contado un método para hallarla "a mano" diagonalizando $A^\dagger A$, aunque no es muy eficiente. En resumen, la diagonalización da V y U es una matriz unitaria que cumple $AV = UD$.

ACTIVIDAD 6.2.4. *Implementa en* `matlab/octave` *el algoritmo anterior usando que* `[V,D] = eig(S)` *da la diagonalización* $S = VDV^\dagger$ *con* V *unitaria para una matriz hermítica* S*. Halla* U *y* D *con* `qr` *y trata de justificar por qué sabemos que la* D *hallada de esta forma debe tener ceros fuera* d_{ii}*, lo cual va más allá de ser triangular (en rigor, trapezoidal) superior.*

Referencias

[DLH03] Julián De La Horra. *Estadística Aplicada*. Díaz de Santos, tercera edición, 2003.

[Mir60] L. Mirsky. Symmetric gauge functions and unitarily invariant norms. *Quart. J. Math. Oxford Ser. (2)*, 11:50–59, 1960.

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

[SB93] J. Stoer and R. Bulirsch. *Introduction to numerical analysis*, volume 12 of *Texts in Applied Mathematics*. Springer-Verlag, New York, second edition, 1993. Translated from the German by R. Bartels, W. Gautschi and C. Witzgall.