Algoritmos para el cálculo de cuadrados mínimos

Juan Pablo De Rasis

22 de febrero de 2019

Dados $m, n \in \mathbb{N}$ tales que $m \geq n$, una matriz $A \in \mathbb{C}^{m \times n}$ de rango n y un vector $b \in \mathbb{C}^m$, sabemos que el sistema $Ax = b$ admite una solución $x \in \mathbb{C}^n$ si y solo si b pertenece al subespacio generado por las columnas de A. Cuando no se satisface dicha propiedad, podemos tratar de encontrar $x \in \mathbb{C}^n$ tal que $\|Ax - b\|_2$ sea mínimo, y sabemos que dicho x satisface $A^*Ax = A^*b$. Por consiguiente, podemos programar en `Octave` una función que resuelva este problema de la siguiente manera:

```
function x = cuadmin(A,b)

x=(A'*A) \ (A'*b);
```

Este procedimiento, con alguna variante, fue utilizado hasta la década de 1960, cuando se popularizó un novedoso procedimiento basado en la factorización QR que mejoraba numéricamente el problema[1]. La problemática subyacente a este procedimiento consiste en la condición de la matriz A^*A, la cual suele estar muy mal condicionada. Esto se verá reflejado en que pequeñas perturbaciones en las entradas de la matriz A, como las obtenidas al calcular sus flotantes para hacer cálculos en la máquina o bien las generadas por errores experimentales de medición, se verán amplificados al calcular A^*A y esto redundará en grandes errores en la solución obtenida. El objetivo es analizar cómo mejorar el condicionamiento de este problema y soslayar los grandes condicionamientos.

1 Factorización QR y cuadrados mínimos

Dados $m, n \in \mathbb{N}$ tales que $m \geq n$ y una matriz $A \in \mathbb{C}^{m \times n}$ de rango n, sabemos que es posible factorizar A como el producto entre una matriz unitaria Q y una matriz triangular superior R. Conocemos dos métodos para encontrar unas tales Q y R.

Conocemos el método de ortonormalización de Gram-Schmidt, el cual nos da una matriz unitaria $Q \in \mathbb{C}^{m \times n}$ y una matriz triangular superior inversible $R \in \mathbb{C}^{n \times n}$. Esto es lo que se conoce como *una factorización QR* reducida *de A*.

Si utilizamos el método de los reflectores de Householder, obtenemos, en cambio, una matriz unitaria $Q \in \mathbb{C}^{m \times m}$ y una matriz triangular superior $R \in \mathbb{C}^{m \times n}$. Esto es lo que se conoce como *una factorización QR* completa *de A*.

En general, una factorización QR de A se dice *reducida* si $Q \in \mathbb{C}^{m \times n}$ y $R \in \mathbb{C}^{n \times n}$, y se dice *completa* si $Q \in \mathbb{C}^{m \times m}$, $R \in \mathbb{C}^{m \times n}$, y es tal que si suprimimos las últimas $m - n$ columnas de Q y las últimas $m - n$ filas de R (las cuales son trivialmente nulas pues R debe ser triangular superior), obtenemos una factorización reducida.

En lo que sigue, consideraremos A y su factorización QR reducida. Analicemos cómo nos queda el método usual de cuadrados mínimos reescrito en términos de Q y R.

[1] Trefethen, Lloyd N. & David Bau, III. *Numerical Linear Algebra*. Philadelphia, Society for Industrial and Applied Mathematics, 1997, p. 83.

Dado un vector $b \in \mathbb{C}^m$, para encontrar $x \in \mathbb{C}^n$ que minimice $\|Ax - b\|_2$ debemos resolver el sistema $A^*Ax = A^*b$. Observemos que, como $A = QR$, entonces $A^*A = R^*Q^*QR = R^*R$, de donde, como $Q^*Q = I_n$, obtenemos el sistema $R^*Rx = R^*Qb$. Por consiguiente, el método usual de cuadrados mínimos nos lleva a resolver un sistema lineal cuya matriz asociada tiene condición $\mathfrak{N}(R^*R)$.

Sin embargo, dado que R es inversible, R^* también lo es, de donde el sistema puede simplificarse a $Rx = Q^*b$, y ahora tenemos un sistema equivalente con condición $\mathfrak{N}(R)$, que resulta preferible.

1.1 Implementación en Octave

Programaremos una nueva función que implemente el algoritmo precedente para calcular cuadrados mínimos. Antes debemos señalar que el comando `qr` de `Octave` nos ofrece una factorización QR completa, dado que el método de los reflectores de Householder es numéricamente mucho más estable que el método de ortonormalización de Gram-Schmidt. Para arreglar este problema y obtener la factorización reducida, nos valdremos de un comando útil para obtener submatrices de una matriz dada A.

Si en `Octave` tenemos introducida una matriz `A` y queremos quedarnos con la submatriz dada por las filas comprendidas entre la i-ésima y la j-ésima, y por las columnas comprendidas entre la m-ésima y la n-ésima, debemos efectuar la función

```
A(i:j,m:n)
```

Por ejemplo: si tenemos introducida la matriz `A=[1,2,3;3,2,1;4,4,4;7,8,9;8,4,0]`, que no es otra cosa que $A = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 4 & 4 & 4 \\ 7 & 8 & 9 \\ 8 & 4 & 0 \end{pmatrix}$, entonces el comando `A(3:5,2:3)` nos devolverá la matriz $\begin{pmatrix} 4 & 4 \\ 8 & 9 \\ 4 & 0 \end{pmatrix}$.

Si solamente deseamos quedarnos con una submatriz que no elimina ninguna fila pero sí elimina columnas, simplemente utilizamos la misma función y, al momento de especificar las filas con las que nos quedamos en la función de `Octave`, simplemente colocamos `:` y no ponemos ningún número. Así, por ejemplo, si buscásemos quedarnos únicamente con las primeras dos columnas de A, el comando `A(:,1:2)` nos dará el resultado buscado. Análogamente podemos quedarnos con las filas 2, 3 y 4 utilizando el comando `A(2:4,:)`.

Esta función nos permite, dada la factorización QR completa que nos ofrece `Octave`, obtener una factorización reducida y, de esta forma, implementar nuestro algoritmo de cuadrados mínimos vía la descomposición QR, del siguiente modo:

```
function x = cuadminqr(A,b)

[m,n]=size(A);
[Q,R]=qr(A);
Q1=Q(:,1:n);
R1=R(1:n,:);

x=(R1 \ (Q1'*b));
```

2 Descomposición en valores singulares y cuadrados mínimos

Dados $m, n \in \mathbb{N}$ tales que $m \geq n$ y $A \in \mathbb{C}^{m \times n}$ de rango n, es posible encontrar una matriz diagonal real Σ tal que sus entradas en la diagonal son todas estrictamente positivas, y dos matrices unitarias U y V tales que $A = U\Sigma V^*$. Esto se conoce como *descomposición en valores singulares*.

Al igual que con la factorización QR, esta descomposición admite una versión *reducida*, en la que $U \in \mathbb{C}^{m \times n}$ y $\Sigma, V^* \in \mathbb{C}^{n \times n}$, y una versión *completa*, en la que $U \in \mathbb{C}^{m \times m}$, $\Sigma \in \mathbb{C}^{m \times n}$ y $V^* \in \mathbb{C}^{n \times n}$, y de tal forma que al quedarnos con las primeras n columnas de U y las primeras n filas de Σ, obtenemos una factorización reducida.

En lo que sigue, consideraremos únicamente la versión *reducida* de la descomposición en valores singulares $A = U\Sigma V^*$.

Dado un vector $b \in \mathbb{C}^m$, para encontrar $x \in \mathbb{C}^n$ que minimice $\|Ax - b\|_2$ debemos resolver el sistema $A^*Ax = A^*b$. Observemos que, como $A = U\Sigma V^*$, entonces

$$A^*A = V\Sigma U^* U\Sigma V^* = V\Sigma^2 V^*$$

con lo cual el vector x satisface $V\Sigma^2 V^* x = V\Sigma U^* b$.

Ahora, observemos que Σ es inversible, pues es una matriz diagonal cuadrada (ya que nuestra descomposición es reducida) y su diagonal está conformada por entradas estrictamente positivas. Por lo tanto, multiplicando por $\Sigma^{-1}V^*$, obtenemos el sistema equivalente $\Sigma V^* x = U^* b$, que está mucho mejor condicionado.

Más aún, dado que Σ es diagonal y V es unitaria, resulta algorítmicamente preferible resolver el sistema $\Sigma y = U^* b$, el cual se resuelve trivialmente al ser Σ diagonal, y posteriormente tomar $x = Vy$, de forma que $\Sigma V^* x = \Sigma V^* V y = \Sigma y = U^* b$.

2.1 Implementación en Octave

Al igual que con la descomposición QR, el comando `svd` nos ofrece una descomposición completa en valores singulares. Por consiguiente, para reproducir nuestro algoritmo en `Octave`, requeriremos modificar la descomposición que nos ofrece para obtener de ella una descomposición reducida. Esto lo logramos de la siguiente manera:

```
function x=cuadminsvd(A,b)

[m,n]=size(A);
[u1,s1,V]=svd(A);
U=u1(:,1:n);
S=s1(1:n,:);

y=S \ (U'*b);
x=V*y;
```

3 Testando los métodos

Introducimos el vector `b=[1;1;1;1]`. Comencemos con `A=[1,1,1;1,2,3;5,7,6;9,0,4]`, y apliquemos los tres comandos `cuadmin(A,b)`, `cuadminqr(A,b)` y `cuadminsvd(A,b)`. Observamos que en los tres casos obtenemos la misma solución. Esto obedece al hecho de que este problema no está mal condicionado.

Repetimos el procedimiento utilizando `A=[1,1,eps;eps,10^10,80;4,0,8;0,0,1]`. Observamos que la función `cuadmin` nos ofrece una solución muy alejada de la real, en tanto que los comandos `cuadminqr` y `cuadminsvd` nos ofrecen la misma solución, la cual es mucho más acertada.

Finalmente, repetimos para `A=[1,1,1;eps,10^7,eps;1000,1000,1000;1,1,1-eps]`. Observamos que, si bien el comando `cuadmin` nos ofrece una solución muy alejada de la real y los comandos `cuadminqr` y `cuadminsvd` nos ofrecen una solución más acertada (aunque ambos nos devuelven respuestas notablemente distintas), en este caso el comando `cuadminqr` tuvo problemas al ofrecer esa solución en tanto el sistema lineal a resolver quedó compatible indeterminado debido a errores dados por aproximación. Esto no ocurrirá en el comando `svd` nunca, dado que en nuestro algoritmo estamos resolviendo específicamente un sistema lineal diagonal con elementos no nulos en la diagonal. Es por eso que este comando no devolvió ningún tipo de inconveniente[2].

[2]Estos experimentos fueron realizados en la versión 4.2.1 de `Octave`, con lo cual los resultados observados se adaptan a lo que se espera de dicha versión del programa

4 ANEXO: Resolución del Ejercicio 5 de la Práctica 6

Se nos pide implementar en `Octave` un programa que reciba dos vectores x, y en $\mathbb{R}^n$ para algún $n \in \mathbb{N}$, calcule la tabla de diferencias divididas y devuelva el polinomio de grado a lo sumo $n-1$ que interpola los puntos $\{(x_i, y_i) : i \in [1, n] \cap \mathbb{N}\}$. El siguiente programa en Octave resolverá este problema, llamando `T` a la tabla de diferencias divididas y devolviendo como resultado un vector que debe interpretarse como los coeficientes del polinomio, ordenados de tal forma que la primera entrada sea el coeficiente principal y la última entrada sea el término independiente. Más aún, este programa nos ofrecerá el gráfico del polinomio interpolador en la región donde se encuentran los nodos que interpolamos.

Dejamos a cargo del lector interpretar y comprender lo que realiza el programa y por qué funciona.

```
function P=difdiv(x,y)

n=length(x);
T=zeros(n,n+1);

for i=1:n
T(i,1)=x(i);
T(i,2)=y(i);
end

for j=3:n+1
for i=1:n-j+2
T(i,j)=(T(i+1,j-1)-T(i,j-1))/(x(i+j-2)-x(i));
end
end

P=[zeros(1,n-1),T(1,2)];

for j=2:n
W=1;
for i=1:j-1
W=conv([1,-x(i)],W);
end
W=[zeros(1,n-length(W)),W];
P=P+T(1,j+1)*W;
end

t=linspace(x(1),x(n),5000);
f=zeros(1,5000);

for i=1:5000
f(i)=polyval(P,t(i));
end

plot(t,f)
```