Análisis Numérico Matricial. Curso 2021-2022.
Grado en Matemáticas. Universidad de Murcia.
Práctica computacional 3. (4/Marzo/2022)

Mostramos algunos ejemplos de códigos y planteamos ejercicios computacionales.
Observación: No usar tildes en los comentarios de los códigos.

Ortogonalización usando QR

Consideramos la secuencia de Fibonacci $n = 2, 3, 5, ..., Ntotal$. Tomamos $M = 30$ matrices generadas por $rand(n(i))$ en cada dimensión $n(i)$ y calculamos su factorización QR con Gram-Schmidt clásico (GSC), Gram-Schmidt modificado (GSM) y Gram-Schmidt Householder (GSH). Vamos a medir la desviación de la ortogonalidad de cada matriz Q.

Sean $a_1, a_2, ..., a_n$ una familia de n vectores de $\mathbb{R}^m$ y A la matriz $m \times n$ cuyas columnas sean los vectores $(a_j)_{1 \leq j \leq n}$. Supongamos que el rango de A es r. Escribir un código que devuelva una familia ortonormal de r vectores $u_1, u_2, ..., u_r$ de $\mathbb{R}^m$ aplicando el proceso de ortogonalización de Gram-Schmidt a la matriz A. El algoritmo en pseudolenguaje:

```
for p=1:n
        s=0
           for k=1:p-1
                 s=s+<a_p,u_k>u_k
          end
       s=a_p-s
       if ||s|| no cero then
              u_p=s/||s||
          else
              u_p=0
       end
end
```

En el siguiente codigo **gramschmidt.m** se realiza el proceso de Gram-Schmidt

```
function [Q,R] = gramschmidt(A)
      [m,n] = size(A);
       Q = A; R = zeros(n);
       for k = 1:n
          R(1:k-1,k) = Q(:,1:k-1)'*A(:,k);
          Q(:,k) = A(:,k) - Q(:,1:k-1)*R(1:k-1,k);
          R(k,k) = norm(Q(:,k));
          Q(:,k) = Q(:,k)/R(k,k);
      end
end
```

1. Determinar la complejidad computacional del algoritmo (número de multiplicaciones y divisiones para valores grandes de n).

2. Usar el programa **gramschmidt.m** y comprobar este programa con la matriz A definida por

$$n = 5; u = 1 : n; u = u'; c2 = cos(2 * u); c = cos(u); s = sin(u);$$

$$A = [u,\ c2,\ ones(n, 1),\ rand() * c. * c,\ exp(u),\ s. * s];$$

Denotamos por Q la matriz obtenida aplicando el proceso de ortonormalización a A.

   - Calcular QQ' y $Q'Q$. Comentar el resultado
   - Aplicar el algoritmo **GramSchmidt** a Q. Comentar el resultado

3. Cambiar el programa **gramschmidt.m** a un programa **gramschmidt1.m** que devuelva una matriz cuyas primeras r columnas sean los r vectores q_k y las últimas $n - r$ columnas sean cero de forma que $Q' * Q = Id$ y $A = QR$

   - Comprueba su funcionamiento con las matrices
     - A del ejercicio anterior,
     - A=hilb(k) con k=6, 8 y 12,
     - matrices A=rand(floor(10*rand(1))+1,floor(10*rand(1))+1) y con la matriz
     - A=[1,3,-5,2,8,2; 5,-2,-1,8,1,-3; 4,3,-16,23,19,-4; 9,2,-1,-1,7,4]

     escribiendo la norma de $A - QR$ y de $Q' * Q - Id$.
   - Si todavía no paso nada raro añade a A un vector columna con ruido: A=[A,1000*eps*rand(4,1
   - Analiza la identidades $A^tA = R^tR$.

4. Implementa una función

   solveQR(Q, R, b)

   para resolver sistemas lineales $Ax = b$ conocida una factorización QR [Q,R] de $A = QR$ con Q una matriz ortogonal y R una matriz triangular superior.

   La solución de $Ax = b \Longleftrightarrow QRx = b \Longleftrightarrow Rx = Q^tb$, se obtiene al resolver la última ecuación por el método ascendente. Comprueba su funcionamiento.

En este código **modgramschmidt.m** se usa el proceso de Gram-Schmidt modificado

```
function [Q,R] = modgramschmidt(A)
      [m,n] = size(A);
      Q = A; R=zeros(n);
      for k = 1:n
         R(k,k) = norm(Q(:,k));
         Q(:,k) = Q(:,k)/R(k,k);
         R(k,k+1:n) = Q(:,k)'*Q(:,k+1:n);
         Q(:,k+1:n) = Q(:,k+1:n) - Q(:,k)*R(k,k+1:n);
end
```

En el siguiente codigo **testQRortogonalidad.m** se comparan los tres métodos

```
tic;
clear all;
% Numero de puntos a tomar de la secuencia de Fibonacci
long= 12;
% Numero de matrices de muestra en cada dimension
% Creamos 30 matrices aleatorias con de talla n(i)xn(i) del numero
% de la sucesion n(i) y luego calculamos sus descomposiciones mediante
% varios metodos y hayamos la desviacion correspondiente a cada metodo.
muestras=30;
n=[1:long]; % Creamos un array vacio que va a contener la sucesion de Fibonacci
n(1)=2; % Colocamos un 2 en la primera posicion del array
        % de Fibonacci (2) como el primero de la sucesion  (2 3 5 8...)
n(2)=3; % Colocamos el siguiente numero de la sucecion seguidamente
% Mediante este bucle llenamos el array con mas numeros de la sucecion
% de Fibonacci hasta el numero que ocupa la posicion long
for i=3:long
n(i) = n(i-1)+n(i-2); % Metodo recursivo
end
nrms1=zeros(muestras,long); % Almacena errores en cada dim para GSC
nrms2=zeros(muestras,long); % Almacena errores en cada dim para GSM
nrms3=zeros(muestras,long); % Almacena errores en cada dim para GSH


for i=1:long
for j=1:muestras
a = rand(n(i)); % crea una matriz con entradas aleatorias de dimension n(i)
[q1,r1]= gramschmidt(a); % QR clasico via GSC sobre la matriz a
[q2,r2]= modgramschmidt(a); % QR via GSM sobre la matriz a
[q3,r3]= Householdergramschmidt(a); % QR via GSH sobre la matriz a
nrms1(j,i) = norm(q1'*q1-eye(n(i)),inf); % desviacion de ortogonalidad
nrms2(j,i) = norm(q2'*q2-eye(n(i)),inf); % desviacion de ortogonalidad
nrms3(j,i) = norm(q3'*q3-eye(n(i)),inf); % desviacion de ortogonalidad
end
end

% Hacemos graficos de los resultados con ajuste lineal para cada metodo
figure(1);
subplot(1,3,1), % Parte izquierda
loglog(n,nrms1',"r.",n,n.^3/n(long)^3*mean(nrms1(:,long)),'b');
% Linea de orden O(n^3)
title('GSC ortogonalidad desviacion ')
subplot(1,3,2), % Parte central
loglog(n,nrms2',"r.",n,n.^2/n(long)^2*mean(nrms2(:,long)),'b');
% Linea de orden O(n^2)
```

```
title('GSM ortogonalidad desviacion')
subplot(1,3,3), % Parte derecha
loglog(n,nrms3',"r.",n,n.^(1.24)/n(long)^(1.24)*mean(nrms3(:,long)),'b');
title('GSH ortogonalidad desviacion')
% Con 1.24 en nrms3 se ve mejor ajuste de la recta. Orden simple ==1
% La precision se ve segun estan dispersos los puntos para cada valor

% Ahora calculamos los datos necesarios para obtener la figura 2
m=[1:16];
nrms_1=[];
nrms_2=[];
nrms_3=[];
for i=1:16
h=hilb(i); % Generamos las matrices de Hilbert
[q1,r1]= gramschmidt(h); % QR clasico via GSC
[q2,r2]= modgramschmidt(h); % QR via GSM
[q3,r3]= Householdergramschmidt(h); % QR via GSH
nrms_1(i) = norm(q1'*q1-eye(i),inf); % desviacion de ortogonalidad
nrms_2(i) = norm(q2'*q2-eye(i),inf); % desviacion de ortogonalidad
nrms_3(i) = norm(q3'*q3-eye(i),inf); % desviacion de ortogonalidad
end
% En total tenemos 16 matrices de Hilbert de talla mxm donde m va desde 1 hasta 16
% Luego, horizontalmente necesitamos 16 puntos y para el eje de abscisas
% vemos la tendencia con escala logaritmica
figure(2)
semilogy(m,nrms_1,'x',m,nrms_2,'d',m,nrms_3,'o');
xlim([0,17]);
toc;
```

Inspirándose en los códigos **gramschmidt.m** y **modgramschmidt.m** programar **Householdergramschmidt.m**. Obtener la gráfica de la Figura 1 y reproducir la gráfica de la Figura 2.

Ejercicios:

1. Ver

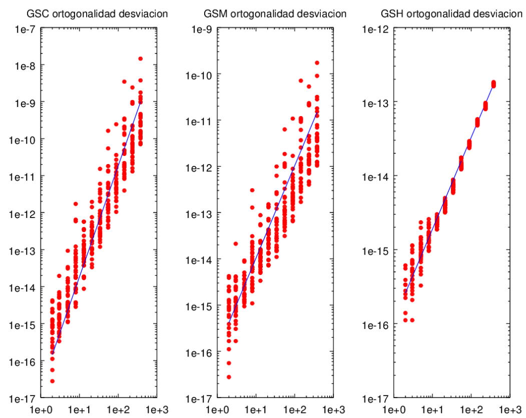

Figura 1: Para cada dimensión $n = 2, 3, 5, ..., F_{14}$ donde F_k es el número de Fibonacci, se calculan 30 matrices aleatorias por el método de Gram-Schmidt clásico (GSC), modificado (GSM) y Householder (GSH). La desviación de la ortogonalidad se calcula mediante el error $\|Q^H Q - I\|_\infty$ y se muestra en una escala loglog con una línea $O(n^3)$ para como referencia para GSC y una línea $O(n^2)$ para GSM. **Se debe completar la figura para GSH**. Siendo estas matrices benignas en principio se observa pérdida de ortogonalidad. Para $n = 377$ hay ortogonalidad sólo en precisión simple, ya que el error es del orden de 10^{-6}

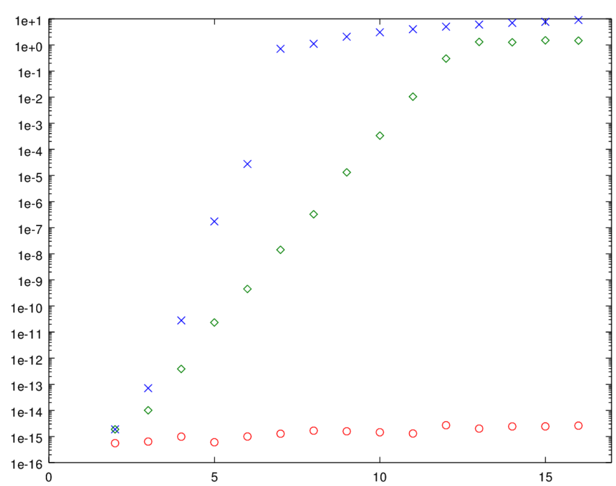

Figura 2: Comparación del método clásico de Gram-Schmidt (GSC) con cruces, del método modificado de Gram-Schmidt (CGM) con diamantes y de Householder (CGH) con círculos para obtener la factorización QR. Se marca $\|Q^H Q - I\|_\infty$ para cada matriz $H_n = QR$ donde H_n es la matriz de Hilbert $n \times n$ y se usa un comando semilogy para obtener la gráfica.