Métodos Numéricos Avanzados

Trabajo Práctico 1

Segundo Cuatrimestre 2016

Random Sparse Matrix

Autores:

José Carlos Noriega Defferrari	-	51231
Juan Marcos Bellini	-	52056
Franco Cesar Prudhomme	-	54263
Luis Ignacio Marzoratti	-	55449
Raffael Rudolf Düll	-	58598

Palabras clave:

Random Sparse Matrix, Matriz rala, Autovalores, QR, Gram-Schmidt, MVMRAN, MATRAN

30 de Septiembre de 2016

Índice

1. Resumen

Las matrices dispersas son aquellas cuyos elementos son, en su mayoría, iguales a cero. Conceptualmente, son matrices que representan sistemas con bajo acoplamiento entre sus elementos. El foco principal del trabajo se centra en la manipulación de dichas matrices, y el cálculo de sus autovalores.

2. Introducción

El almacenamiento y procesamiento de matrices de gran tamaño por medio de una computadora, utilizando los métodos tradicionales, requiere una gran cantidad de recursos (tiempo y memoria). Es por ello que se han diseñado algoritmos específicos para matrices dispersas (o también llamadas, matrices ralas)[1].

El siguiente informe trata acerca de la generación de matrices dispersas aleatorias (*Random Sparse Matrix*, en inglés), y el cálculo de sus autovalores. A continuación, se detallarán los algoritmos utilizados, las dificultades encontradas, y las decisiones tomadas para solucionar dichos problemas. Luego se mostrarán los resultados obtenidos, y se elaboraran conclusiones en base a ellos.

3. Metodología

3.1. Implementación

La implementación de los métodos fue realizada en su totalidad en el lenguaje de programación *Octave*, utilizando la plataforma *GNU Octave*[2], debido a su gran eficiencia en cuanto a la manipulación de matrices, además de su facilidad y simplicidad de uso.

El cálculo de autovalores fue llevado a cabo por el algoritmo QR[3], el cual se basa en la descomposición QR de una matriz. Dicha descomposición fue efectuada a través de tres métodos distintos: Gram-Schmidt, Gram-Schmidt modificado y Rotaciones de Givens.

3.2. Construcción de la matriz dispersa aleatoria (RSM)

La construcción de matrices ralas aleatorias fue realizada por un programa que recibe dos parámetros, N y NZR. El primero representa la dimensión de dicha matriz. Cabe destacar que el resultado es una matriz cuadrada de $N \times N$. El segundo parámetro, NZR, indica la cantidad de valores distintos de 0 (cero) por columna.

Para que el programa funcione correctamente, el valor de N debe ser mayor o igual al valor de NZR. De hecho, cuanto menor sea la cantidad de valores distintos de 0 por columna, mayor sera la dispersidad de la matriz resultante.

A continuación se presenta el algoritmo implementado:

[1] `https://es.wikipedia.org/wiki/Matriz_dispersa`
[2] `https://www.gnu.org/software/octave/`
[3] `https://es.wikipedia.org/wiki/Algoritmo_QR`

```
1  function RSMMatrix = generateRSM(N, NZR)
2      RSMMatrix = zeros(N);
3      for columnIndex = 1 : N
4          for i = 1 : NZR
5              do
6                  rowIndex = ceil(rand() * N);
7              until (RSMMatrix(rowIndex, columnIndex) == 0)
8              do
9                  RSMMatrix(rowIndex, columnIndex) = rand() * 2 - 1;
10             until (RSMMatrix(rowIndex, columnIndex) != 0)
11         end
12     end
13 end
```

Algoritmo 1: Construcción de matrices dispersas aleatorias.

El método genera una matriz cuadrada de $N \times N$ inicializada con todos sus valores en 0. Luego recorre dicha matriz de izquierda a derecha generando tantos valores aleatorios – en filas aleatorias – como el valor de NZR. Dichos valores aleatorios pertenecen al conjunto $[-1, 1]$.

Se puede ver en la linea 7 que el programa verifica que una posición ya ocupada por un valor no nulo sea ignorada, volviendo a seleccionar una fila al azar. Además, también se comprueba que el valor aleatorio a asignar en la posición elegida no sea 0 – linea 10.

3.3. Descomposición QR

Se han implementado tres métodos distintos para obtener la descomposición QR de matrices. De esta manera se pudo realizar un *benchmark* entre los mismos. Cada método recibe como parámetro una matriz cualquiera – llamada matriz A – y retorna las matrices Q y R en un vector resultado.

3.3.1. Descomposición QR por Gram-Schmidt

El primer algoritmo implementado es la descomposición QR a través del método de ortogonalización de Gram-Schmidt. Para el mismo, se toman las columnas de la matriz A como vectores que se quieren ortonormalizar, y se aplica el método de Gram-Schmidt. Los vectores resultantes forman las columnas de la matriz Q. Dicha matriz tiene la propiedad de ser ortogonal, por lo que el cálculo de la matriz R puede efectuarse como $R = Q^T\ A$.

Matemáticamente, el método se puede resumir en las siguientes ecuaciones[4]:

$$\vec{q_1} = \frac{\vec{a}_1}{\|\vec{a}_1\|} \tag{1}$$

[4] Apuntes de cátedra: Descomposición QR, sección 2

$$\vec{w}_i = \vec{a}_i - \sum_{j=1}^{i-1} proj_{\vec{w}_j} \vec{a}_i \quad con\ i\ \in \{2, 3, ..., N\} \tag{2}$$

$$\vec{q}_i = \frac{\vec{w}_i}{\|\vec{w}_1\|} \tag{3}$$

siendo $\vec{a_i}$ la i-ésima columna de A, y $\vec{q_i}$ la i-ésima columna de Q. Una vez obtenida ésta última, R se puede calcular como se ha dicho anteriormente: $R = Q^T\ A$.

A continuación se encuentra la implementación del algoritmo:

```
1  function [Q R] = gramSchmidtQRDecomposition(matrix)
2      Q = matrix;
3      for i = 1 : columns(matrix)
4          for j = 1 : (i - 1)
5              scalarProduct = matrix(:, i)' * Q(:, j);
6              Q(:, i) -= scalarProduct * Q(:, j);
7          end
8          if (norm(Q(:,i)) < 0.00001)
9              Q(:,i) = zeros(rows(Q),1);
10         else
11             Q(:, i) /= norm(Q(:, i));
12         end
13     end
14     R = Q' * matrix;
15 end
```

Algoritmo 2: Descomposición QR por Gram-Schmidt.

El programa copia la matriz A de entrada en la matriz Q, utilizando esta última como contenedora de los vectores ortonormalizados. En la i-ésima iteración externa se toma un nuevo vector columna de A – llamado $\vec{a_i}$ – y se lo ortonormaliza respecto de los $i - 1$ vectores ya ortonormales contenidos en las primeras $i - 1$ columnas de Q ($\vec{q}_1, \vec{q}_2, ..., \vec{q}_{i-1}$). En la j-ésima iteración interna se calcula la proyección de $\vec{a_i}$ sobre $\vec{q_j}$.

Como se puede ver en la linea 11, en el i-ésimo paso se normaliza el vector $\vec{q_i}$ obtenido, en caso de no ser el vector nulo. De esta manera, la siguiente iteración puede calcular las proyecciones de manera más simple (no tiene que dividir el vector $\vec{q_j}$ por el cuadrado de su norma). En caso de que el vector sea el nulo (o dicho de otra manera, en caso de que la norma del vector sea 0), se lo deja como esta. Para no trabajar con valores pequeños, se le asigna una columna de 0s a la matriz Q.

Sin embargo, este método tiene una gran desventaja. Produce vectores de norma pequeña, lo cual puede llevar a grandes errores numéricos, que luego son arrastrados y amplificados en cada paso del algoritmo. Es por ello que se ha implementado una modificación al proceso de ortogonalización de Gram-Schmidt.

3.3.2. Descomposición QR por Gram-Schmidt modificado

El problema del método clásico de Gram-Schmidt es la gran cantidad de errores numéricos que se pueden producir al ser utilizado en una computadora. El resultado es un conjunto de vectores que no son del todo ortogonales. Debido a esto, se le hace una modificación al proceso de Gram-Schmidt. En este caso, las matrices Q y R se construyen a medida que avanza el algoritmo. Además, se modifica – en caso de no haberla resguardado – la matriz A. A continuación una breve explicación matemática[5].

Por cómo se define la factorización QR, cada elemento de A se puede expresar según el siguiente producto de matrices:

$$a_{ij} = \begin{bmatrix} q_{i1} & q_{i2} & \cdots & q_{in} \end{bmatrix} \begin{bmatrix} r_{1j} \\ r_{2j} \\ \vdots \\ r_{nj} \end{bmatrix} = \sum_{k=1}^{n} q_{ik} r_{kj} \tag{4}$$

siendo n la cantidad de columnas de Q,y de filas de R, q_{ij} el elemento en la fila i y la columna j de Q, y r_{ij} es el elemento en la fila i y la columna j de R. Tener en cuenta que las matrices A y Q son matrices de $n \times m$. La matriz R es de $n \times n$, con la particularidad de ser triangular superior.

Continuando con el razonamiento, la j-ésima columna de la matriz A – o $\vec{a}_j$ – se puede expresar segun:

$$\vec{a}_j = \begin{bmatrix} \sum_{k=1}^{n} q_{1k} r_{kj} \\ \sum_{k=1}^{n} q_{2k} r_{kj} \\ \vdots \\ \sum_{k=1}^{n} q_{mk} r_{kj} \end{bmatrix} \tag{5}$$

A su vez, la matriz A se la puede expresar como:

$$A = [\vec{a}_1 \ \vec{a}_2 \cdots \vec{a}_n] = \begin{bmatrix} \sum_{k=1}^{n} q_{1k} r_{k1} & \sum_{k=1}^{n} q_{1k} r_{k2} & \cdots & \sum_{k=1}^{n} q_{1k} r_{kn} \\ \sum_{k=1}^{n} q_{2k} r_{k1} & \sum_{k=1}^{n} q_{2k} r_{k2} & \cdots & \sum_{k=1}^{n} q_{2k} r_{kn} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{k=1}^{n} q_{mk} r_{k1} & \sum_{k=1}^{n} q_{mk} r_{k2} & \cdots & \sum_{k=1}^{n} q_{mk} r_{kn} \end{bmatrix} \tag{6}$$

Sin embargo, por cómo opera el algoritmo de ortogonalización de Gram-Schmidt, es evidente que las primeras k columnas de A se puede expresar en función de las primeras k columnas de Q. Expresado según la sintaxis de *Octave*:

[5] Apuntes de cátedra: Descomposición QR, sección 2.1

$$
\begin{aligned}
A(:,1:k) &= [\vec{a}_1\ \vec{a}_2 \cdots \vec{a}_k] \\
&= \begin{bmatrix} \sum_{l=1}^{k} Q(1,l)R(l,1) & \cdots & \sum_{l=1}^{n} Q(1,l)R(l,k) \\ \sum_{l=1}^{k} Q(2,l)R(l,1) & \cdots & \sum_{l=1}^{n} Q(2,l)R(l,2) \\ \vdots & \ddots & \vdots \\ \sum_{l=1}^{k} Q(m,l)R(l,1) & \cdots & \sum_{l=1}^{n} Q(m,l)R(l,k) \end{bmatrix} \\
&= \sum_{l=1}^{k} Q(:,l)R(l,:) = \sum_{l=1}^{k} \vec{q}_l \vec{r}_l^{\,T}
\end{aligned} \tag{7}
$$

Nótese que, para simplificar la notación en la ecuación (7), se piensa a la matriz R de la siguiente manera:

$$
R = \begin{bmatrix} \vec{r}_1^{\,T} \\ \vec{r}_2^{\,T} \\ \vdots \\ \vec{r}_n^{\,T} \end{bmatrix} \tag{8}
$$

En base a lo anterior, se puede definir una nueva matriz $A^{(k)} \in \mathbb{R}^{m\times(n-k+1)}$ según:

$$
A - \sum_{l=1}^{k-1} \vec{q}_l \vec{r}_l^{\,T} = \sum_{l=k}^{n} \vec{q}_l \vec{r}_l^{\,T} = \begin{bmatrix} 0 & A^{(k)} \end{bmatrix} \tag{9}
$$

Si se llama $\vec{x}$ a la primera columna de $A^{(k)}$, y B a la matriz formado por el resto de las columna (o sea, $A^{(k)} = \begin{bmatrix} \vec{x} & B \end{bmatrix}$), y se multiplica miembro a miembro por $\vec{q}_k^{\,T}$, se obtiene la siguiente ecuación:

$$
\begin{aligned}
\vec{q}_k^{\,T} \sum_{l=k}^{n} \vec{q}_l \vec{r}_l^{\,T} &= \sum_{l=k}^{n} \vec{q}_k^{\,T} \vec{q}_l \vec{r}_l^{\,T} = \vec{r}_k^{\,T} \\
&= \vec{q}_k^{\,T} \begin{bmatrix} 0 & A^{(k)} \end{bmatrix} = \vec{q}_k^{\,T} \begin{bmatrix} 0 & \vec{x} & B \end{bmatrix} = \begin{bmatrix} 0 & \vec{q}_k^{\,T}\vec{x} & \vec{q}_k^{\,T} B \end{bmatrix}
\end{aligned} \tag{10}
$$

Por lo tanto, se tiene que

$$
\vec{r}_k^{\,T} = \begin{bmatrix} 0 & \vec{q}_k^{\,T}\vec{x} & \vec{q}_k^{\,T} B \end{bmatrix} \tag{11}
$$

Como $\vec{q}_k^{\,T}$ debe tener norma unitaria, entonces $|R(k,k)| = \left\|\vec{q}_k^{\,T}\vec{x}\right\|_2$. Por lo tanto, se pueden fijar las siguientes relaciones:

$$
R(k,k) = \|\vec{x}\|_2 \tag{12}
$$

$$
\vec{q}_k = \frac{\vec{a}}{\|\vec{x}\|_2} = \frac{\vec{x}}{R(k,k)} \tag{13}
$$

De esta forma, se puede calcular $A^{(k+1)}$ como

$$A^{(k+1)} = B - (\vec{q}_k \vec{r}_k^T)(k+1:n) \tag{14}$$

Gracias a ésta última ecuación, la ecuación (14), se puede implementar el siguiente algoritmo.

```
function [Q R] = modifiedGramSchmidtQRDecomposition(matrix)
    Q = matrix;
    cols = columns(matrix);
    for i = 1 : cols
        R(i, i) = norm(matrix(:, i));
        Q(:, i) = matrix(:, i) / R(i, i);
        for j = i + 1 : cols
            R(i, j) = matrix(:, j)' * Q(:, i);
            matrix(:, j) -= R(i, j) * Q(:, i);
        end
    end
end
```

Algoritmo 3: Descomposición QR por Gram-Schmidt modificado

3.3.3. Descomposición con rotaciones de Givens

Otra forma de calcular la descomposición QR de una matriz A es a través de matrices de rotación de Givens[6]. Una matriz de rotación tiene la siguiente forma:

$$G_\theta = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \tag{15}$$

Estas matrices se pueden utilizar para anular elementos de una matriz cualquiera.

Matriz de Rotación de Givens

Sin perder generalidad, suponiendo que se quiere anular el elemento a_{qp}, con p <q, de la matriz A, se genera la matriz G de la siguiente manera:

$$G(k,k) = \begin{cases} \cos\theta, & \text{si } k = p, q \\ 1, & \text{en otro caso} \end{cases} \tag{16}$$

$$G(k,k) = \begin{cases} +\sin\theta, & \text{si } k = p \text{ y } l = q \\ -\sin\theta, & \text{si } k = 1 \text{ y } l = p \\ 0, & \text{en otros casos en que } k \neq l \end{cases} \tag{17}$$

Como se puede ver, G se parece a la matriz identidad, difiriendo en las filas p y q. A esta matriz se la llama matriz de rotación de Givens. Es una matriz ortogonal. Además,

[6] Apuntes de cátedra: Descomposición QR, sección 4

al pre-multiplicarla por la matriz A, solo afecta las filas p y q de ésta última. Llamando $c = \cos\theta$ y $s = \sin\theta$, se puede establecer la siguiente relación:

$$\begin{aligned} G([p,q],[p,q])A([p,q],[p,q]) &= \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} A_{pp} & A_{pq} \\ A_{qp} & A_{qq} \end{bmatrix} \\ &= \begin{bmatrix} cA_{pp} + sA_{qp} & cA_{pq} + sA_{qq} \\ -sA_{pp} + cA_{qp} & -sA_{pq} + cA_{qq} \end{bmatrix} \end{aligned} \tag{18}$$

Se desea que se cumpla la siguiente condición:

$$-sA_{pp} + cA_{qp} = 0 \tag{19}$$

Para ello, basta con hacer

$$\tan\theta = \frac{s}{c} = \frac{A_{qp}}{A_{pp}} \tag{20}$$

Sin embargo, no se necesita calcular el valor de θ para calcular c y s. Definiendo $t = \tan\theta$:

$$\begin{aligned} 1 + t^2 = 1 + \frac{s^2}{c^2} &= \frac{c^2 + s^2}{c^2} = \frac{1}{c^2} \\ &\Rightarrow c = \frac{1}{\sqrt{1+t^2}}, s = ct \end{aligned} \tag{21}$$

También se podría definir:

$$z = \frac{1}{\tan\theta} = \frac{c}{s} = \frac{A_{pp}}{A_q q} \tag{22}$$

En este caso, se procedería de la siguiente forma:

$$\begin{aligned} 1 + z^2 = 1 + \frac{c^2}{s^2} &= \frac{s^2 + c^2}{s^2} = \frac{1}{s^2} \\ &\Rightarrow s = \frac{1}{\sqrt{1+z^2}}, c = sz \end{aligned} \tag{23}$$

Según cuál sea el caso, conviene trabajar con la ecuación (21), o con la ecuación (23). Suponiendo que, por errores de redondeo, se trabaje con

$$\hat{A}_{qp} = A_{qp} + \epsilon_1, \hat{A}_{pp} = A_{pp} + \epsilon_2 \tag{24}$$

con $\epsilon_1, \epsilon_2 \geq 0$. Llamando $\hat{c}_1, \hat{s}_1$ a los valores obtenidos utilizando la ecuación (21) y las definiciones establecidas en la ecuación (24), se obtienen los siguientes resultados:

$$\hat{t} = \frac{\hat{A}_{qp}}{\hat{A}_{pp}} \approx t\left(1 + \frac{\epsilon_1}{\hat{A}_{qp}} - \frac{\epsilon_2}{\hat{A}_{pp}}\right) \tag{25}$$

$$\hat{c}_1 = \frac{1}{\sqrt{1+\hat{t}^2}} \approx c\left[1 + s^2\left(\frac{\epsilon_1}{\hat{A}_{qp}} - \frac{\epsilon_2}{\hat{A}_{pp}}\right)\right] \tag{26}$$

$$\hat{s}_1 = \hat{c}_1\hat{t} \approx s\left[1 + s^2\left(\frac{\epsilon_1}{\hat{A}_{qp}} - \frac{\epsilon_2}{\hat{A}_{pp}}\right)\right] \tag{27}$$

haciendo uso reiterado del desarrollo en serie de Taylor[7] de primer orden, y la hipótesis de que $|\epsilon_i| \ll 1, i \in \{1, 2\}$. De manera similar, llamando $\hat{c}_2, \hat{s}_2$ a los valores obtenidos utilizando la ecuación (23) y las definiciones establecidas en la ecuación (24), y nuevamente haciendo uso reiterado del desarrollo en serie de Taylor de primer orden, se obtienen:

$$\hat{z} = \frac{\hat{A}_{pp}}{\hat{A}_{qp}} \approx z\left(1 + \frac{\epsilon_2}{\hat{A}_{pp}} - \frac{\epsilon_1}{\hat{A}_{qp}}\right) \tag{28}$$

$$\hat{s}_2 = \frac{1}{\sqrt{1+\hat{z}^2}} \approx s\left[1 + c^2\left(\frac{\epsilon_2}{\hat{A}_{pp}} - \frac{\epsilon_1}{\hat{A}_{qp}}\right)\right] \tag{29}$$

$$\hat{c}_2 = \hat{s}_2\hat{z} \approx c\left[1 + c^2\left(\frac{\epsilon_2}{\hat{A}_{pp}} - \frac{\epsilon_1}{\hat{A}_{qp}}\right)\right] \tag{30}$$

Tomando como ejemplo s, al comparar el error absoluto utilizando las ecuaciones (21) y (23):

$$\frac{|\Delta s_1|}{|\Delta s_2|} = \frac{|\hat{s}_1 - s_1|}{|\hat{s}_2 - s_2|} \approx \frac{|s|^3\left|\frac{\epsilon_1}{A_{qp}} - \frac{\epsilon_2}{A_{pp}}\right|}{|s|\,c^2\left|\frac{\epsilon_2}{A_{pp}} - \frac{\epsilon_1}{A_{qp}}\right|} = \frac{s^2}{c^2} = \left(\frac{A_{qp}}{A_{pp}}\right)^2 \approx \left(\frac{\hat{A}_{qp}}{\hat{A}_{pp}}\right)^2 \tag{31}$$

se puede concluir que, si $\left|\hat{A}_{qp}\right| < \left|\hat{A}_{pp}\right|$, conviene utilizar la ecuación 21. En caso contrario, conviene la ecuación 23.

Todo este razonamiento se puede resumir en el siguiente algoritmo:

```
1  function [c s] = givensrotation(a, b)
2      if (b == 0)
3      c = 1;
4      s = 0;
5      else
6          if (abs(b) > abs(a))
7              r = a / b;
8              s = 1 / sqrt(1 + r**2);
9              c = s * r;
10         else
```

[7] https://es.wikipedia.org/wiki/Serie_de_Taylor

```
            r = b / a;
            c = 1 / sqrt(1 + r**2);
            s = c * r;
        end
    end
end
```

Algoritmo 4: Cálculo de los elementos c y s de una matriz de rotación de Givens

Como se puede ver, el algoritmo recibe dos valores: a y b. El primero de ellos, a, representa el valor A_{pp}. Por otro lado, b representa A_{qp}.

Factorización QR utilizando matrices de rotación de Givens

Como se dijo al comienzo de esta sección, las matrices de rotación de Givens pueden ser utilizadas para calcular la descomposición QR de una matriz A. Lo que de debe hacer es anular todos los elementos de la matriz por debajo de la diagonal principal utilizando rotaciones de Givens. A continuación se encuentra la implementación del algoritmo:

```
1  function [Q R] = givensRotationQRDecomposition(A)
2      m = rows(A);
3      Q = eye(m);
4      R = A;
5
6      for j = 1 : n
7          for i = m : -1 : (j + 1)
8              G = eye(m);
9              [c, s] = givensrotation(R(i - 1, j), R(i, j));
10             G([i - 1, i], [i - 1, i]) = [c - s; s c];
11             R = G' * R;
12             Q = Q * G;
13         end
14     end
15 end
```

Algoritmo 5: Descomposición en QR con rotaciones de Givens

Como se puede ver en el código, en la linea 9 se hace uso de la función definida en el algoritmo 4 de este documento.

3.4. Cálculo de autovalores

Para calcular los autovalores de la matriz se utilizó un método global de aproximación, utilizando las descomposiciones QR mencionadas anteriormente. Se siguió un documento[8] sobre técnicas iterativas para encontrar los autovalores, contemplando el hecho de que los mismos pudieran ser tanto valores reales como complejos.

[8] http://www.win.tue.nl/casa/meetings/seminar/previous/_abstract051109_files/presentation_full.pdf

Se emplearon la técnica de corrimiento simple para acelerar la convergencia de los valores cuando los módulos de los autovalores están cerca el uno del otro, y la de corrimiento doble para contemplar los valores complejos.

El método opera de la siguiente manera. Primero calcula la descomposición QR de A, y luego obtiene una nueva matriz $A' = R * Q$. Se tienen en cuenta los elementos de la diagonal para calcular los autovalores (siguiendo el algoritmo iterativo referenciado en la bibliografía), teniendo en cuenta si el autovalor es real o complejo.

```
1  function eigenvalues = calculateEigenvalues(A, error, decomposition)
2      eigenvalues = [];
3      n = columns(A);
4      current = 0;
5      do
6          previousA = A;
7          [Q R] = decomposition(A);
8          A = R * Q;
9          if (condition(error, A(n - current, n - current - 1), A(n -
               current - 1, n - current - 1), A(n - current, n - current)))
10             eigenvalues(n - current, 1) = A(n - current, n - current);
11             A = A(1:n - current - 1, 1:n - current - 1);
12             current++;
13         elseif (condition(error, A(n - current -1, n - current - 2), A(n
               - current - 1, n - current - 1), A(n - current - 2, n -
               current - 2)))
14             [r1 r2] = getValues(A, n, current);
15             eigenvalues(n - current - 1, 1) = r1;
16             eigenvalues(n - current, 1) = r2;
17             A = A(1:n - current - 2, 1:n - current - 2);
18             current += 2;
19         end
20     until (current >= n - 2)
21
22     if (current == n - 2)
23         [r1 r2] = getValues(A, n, current);
24         eigenvalues(n - current - 1, 1) = r1;
25         eigenvalues(n - current, 1) = r2;
26         A = A(1:n - current - 2, 1:n - current - 2);
27     elseif (current == n - 1)
28         eigenvalues(1) = A(1, 1);
29     end
30 end
31
32 function bool = condition(error, value1, value2, value3)
33     bool = abs(value1) < (error * (abs(value2) + abs(value3)));
34 end
35
36 function [r1 r2] = getValues(A, n, current)
37     w = A(n - current - 1, n - current - 1);
38     x = A(n - current - 1, n - current);
```

```
39      y = A(n - current, n - current - 1);
40      z = A(n - current, n - current);
41      rootValues = roots([1 -(w+z) (w*z)-(x*y)]);
42      r1 = rootValues(1);
43      r2 = rootValues(2);
44  end
```

Algoritmo 6: Implementación del método QR de obtención de autovalores

4. Resultados y conclusiones

Se han realizado dos *benchmarks* sobre los algoritmos implementados. El primero fue realizado sobre el método de obtención de autovalores de la matriz A utilizando los tres algoritmos de cálculo de la factorización QR de dicha matriz. Se evaluó el tiempo – en segundos – que le toma hacer el cálculo de los autovalores, en función del tamaño de la matriz de entrada. Se han utilizado matrices cuadradas de $n \times n$.

Las pruebas se realizaron con una baja densidad de valores no nulos por columna (0.25), repitiendo el proceso 30 veces. De esta manera se puede obtener un promedio que se ajusta a la realidad. Los resultados – que se pueden observar en la figura 1 de este documento – muestran que el método de factorización QR más rápido es el algoritmo de ortonormalización de Gram-Schmidt clásico. Luego sigue el método modificado de Gram-Schmidt. Por último, el algoritmo que más tiempo le lleva terminar es el de las rotaciones de Givens. El resultado es evidente. A mayor estabilidad numérica que posea el algoritmo, mayor costo computacional tiene. A la hora de seleccionar uno de ellos, el usuario debe analizar la situación, y elegir qué desea sacrificar: tiempo o precisión.

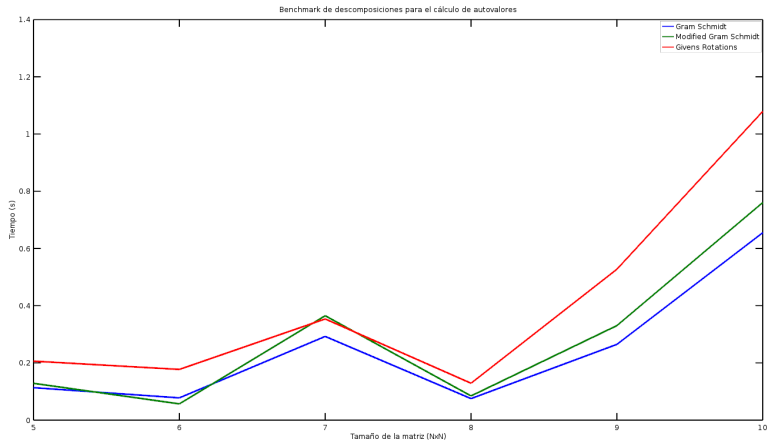

Figura 1: Resultados del *benchmark* realizado al algoritmo de cálculo de autovalores

El segundo *benchmark* realizado fue sobre el método mostrado en el algoritmo 1

de de este documento. Se evaluó la función que genera matrices dispersas aleatorias. Para el mismo se mantuvo constante el valor NZR, logrando una densidad de valores no nulos por columna de 0.25, y variando el valor de N, para ver cuánto tiempo – nuevamente en segundos – tarda el programa en devolver la matriz. Los resultados se pueden ver en la figura 2.

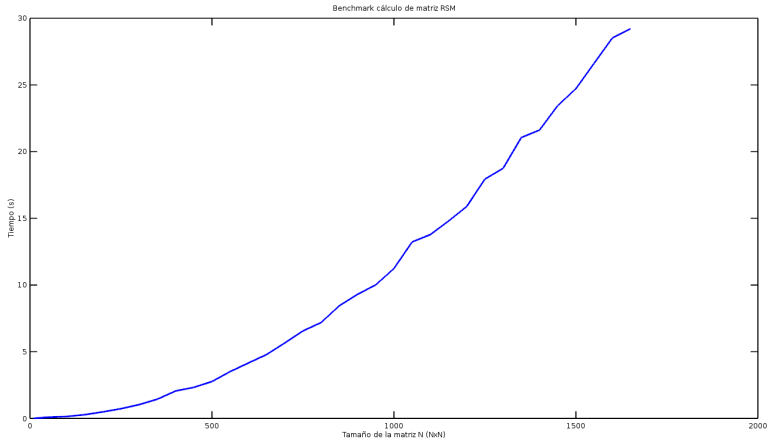

Figura 2: Resultados del *benchmark* realizado al programa de generación de matrices dispersas aleatorias

Por último, tomando el método de descomposición QR con el que mejor resultados se obtuvieron, se procedió a verificar el valor de N máximo para el cual el programa retorna en un tiempo razonable. Los resultados se pueden observar en la tabla 1. Se puede notar que para valores de N<40, el algoritmo tarda alrededor de $60s$. También se puede observar que con valores entre 40 y 60, el tiempo es razonable, variando entre 1 minuto y 5 minutos. Sin embargo, con valores de N>60, el programa se vuelve muy lento, creciendo dicho tiempo a medida que crece el valor de N.

N	Tiempo $[s]$
10	5.32730
15	0.72258
20	3.95338
25	16.42223
30	16.62530
35	20.04132
40	73.99459
45	51.05616
50	120.50823
55	221.21567
60	391.32867
65	540.01190
70	892.09711

Tabla 1: Tiempo de ejecución para obtener autovalores de matrices de $N \times N$ para distintos valores de N

5. Bibliografía

- Introducción a Random Sparse Matrix: http://www.staff.science.uu.nl/~bisse101/Book/PSC/psc4_7.pdf
- Matrix Market MATRAN: http://math.nist.gov/MatrixMarket/data/NEP/matran/matran.html
- Técnicas iterativas para encontrar autovalores: http://www.win.tue.nl/casa/meetings/seminar/previous/_abstract051109_files/presentation_full.pdf
- Wikipedia https://www.wikipedia.org
- Apuntes de cátedra: Descomposición QR