Anexo A

Demostración de la ecuación 2.6

La ecuación A.1 se ha obtenido experimentalmente. Durante este apéndice se expone una deducción formal de la misma.

$$N_{tx} = c_2^n + \left\lceil \frac{n}{2} \right\rceil + 1 \tag{A.1}$$

Para comenzar, se ha de recordar la propiedad de uniformidad en las transmisiones de pulsos por parte de los nodos a lo largo de la secuencia, esto es, el número de transmisiones de cada nodo a lo largo de ella ha de ser el mismo en la medida de lo posible. Una manera fácil de obtener esta propiedad es agrupar las transmisiones en bloques de transmisión dentro de la propia secuencia principal. Un bloque de transmisión está compuesto por un número de emisiones de pulsos igual al número de nodos que componen la red. Dentro de cada bloque de transmisión, cada nodo transmite su pulso una vez. Por ejemplo, si $n = 4$, posibles bloques de transmisión pueden ser $'\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}'$, o $'\mathbf{2}, \mathbf{4}, \mathbf{1}, \mathbf{3}'$. Si la red contiene $n = 5$ nodos, un bloque de transmisión podría ser $'\mathbf{1}, \mathbf{4}, \mathbf{5}, \mathbf{3}, \mathbf{2}'$. Esta aproximación es ideal, ya que este hecho no suele suceder exactamente, pero puede ser tomada, ya que observando la secuencia en su totalidad, es equivalente debido a la uniformidad. Los bloques de transmisión que no contengan transmisiones de ciertos nodos y por otra parte, contengan más de una transmisión de algún otro, se verán compensados por otro bloque posterior. Por lo tanto, la aproximación de los bloques de transmisión de pulsos es válida para demostrar la ecuación A.1. De esta forma, aparece una variable auxiliar n_{blocks}, la cual es el número de bloques transmitidos en una secuencia.

Una vez definida esta variable, dos ejemplos son analizados paso a paso. El primer escenario contiene $n = 5$ nodos, para representar a redes con un número impar de

dispositivos, y el segundo escenario contiene $n = 4$, para representar un numero par. En cada paso, se escribe el número de ecuaciones obtenidas en cada nodo. La metodología empleada para el recuento de ecuaciones es la misma explicada en el capítulo 2. La tabla A.1 muestra todo el proceso. La primera fila contiene el número de la transmisión analizada, la segunda almacena la secuencia y la primera columna muestra el número de nodo. Los bloques de transmisión se dividen mediante una doble línea vertical. La tabla en sí, muestra el número de ecuaciones que ha generado cada nodo en cada paso.

Number of transmission	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Schedule	**1**	**2**	**3**	**4**	**5**	**1**	**3**	**1**	**4**	**2**	**5**	**2**	**1**	**5**
Node 1	0	1	2	3	4	4	5	5	6	7	8	9	9	10
Node 2	0	0	1	2	3	4	5	6	7	7	8	8	9	10
Node 3	0	1	1	2	3	4	4	5	6	7	8	9	10	11
Node 4	0	1	2	2	3	4	5	6	6	7	8	9	10	11
Node 5	0	1	2	3	3	4	5	6	7	8	8	9	10	10

Tabla A.1: Número de ecuaciones en cada paso, $n = 5$

Puede observarse que al final del primer bloque, se obtienen 3 ecuaciones, al acabar el bloque dos, este número es 7, y en el bloque tres, las ecuaciones suben hasta 11. De esta forma se deduce que al final de cada bloque, el número de ecuaciones generadas responde a la expresión A.2

$$N_{eq}(n_{blocks}) = N_{tx}(n_{blocks}) - n_{blocks} - 1 \tag{A.2}$$

Donde N_{tx} es el número de transmisiones y n_{blocks} es el número de bloques transmitidos. Al final de cada bloque, el número de transmisiones es igual a la longitud del bloque multiplicada por el número de bloques transmitidos. Así, la ecuación A.2 puede reescribirse como:

$$N_{tx}(n_{blocks}) = n_{blocks} \times n \tag{A.3}$$

$$N_{eq}(n_{blocks}) = n_{blocks}(n-1) - 1 \tag{A.4}$$

De la ecuación A.4, se obtiene la expresión de n_{blocks} para cada valor de N_{eq}:

$$n_{blocks} = \left\lceil \frac{N_{eq}+1}{n-1} \right\rceil \tag{A.5}$$

Mediante la ecuación A.5, se obtiene el número de bloques para el mínimo número de transmisiones de pulsos. El mínimo número de ecuaciones ha de ser menor que el número de ecuaciones contenidas en los n_{blocks} transmitidos, esto es:

$$N_{eq}(n_{blocks}) \geq N_{eq} \Rightarrow N_{eq} + x = N_{eq}(n_{blocks})$$

$$x = N_{eq}(n_{blocks}) - N_{eq} \tag{A.6}$$

Por lo tanto,, el número de transmisiones requeridas es igual al número de transmisiones en función del número de bloques, es decir, $n_{blocks} \times n_{nodes}$, menos una variable **x**.

$$N_{tx} = N_{tx}(n_{blocks}) - x \tag{A.7}$$

Introduciendo A.4, A.5 y A.6 en A.7:

$$N_{tx} = c_2^n + 1 + \left\lceil \frac{c_2^n + 1}{n-1} \right\rceil = c_2^n + 1 + \left\lceil \frac{n}{2} + \frac{1}{n-1} \right\rceil \tag{A.8}$$

Donde N_{eq} ha sido sustituido por c_2^n. Observando los dos miembros dentro del $\lceil . \rceil$ de forma separada, se puede concluir lo siguiente. En este caso, el número n de nodos tomado es impar, luego el primer miembro es siempre un un número entero más $\frac{1}{2}$. El segundo miembro, $\frac{1}{n-1}$ oscila entre valores $\left(0, \frac{1}{2}\right]$ para números n impares y mayores que 1. Al realizar suma de ambos términos, se está añadiendo un número positivo menor que $\frac{1}{2}$ al primer miembro, luego nunca se alcanzará el siguiente número entero, por lo tanto el segundo miembro de $\lceil . \rceil$ puede ser suprimido, llegando a la ecuación A.1.

$$N_{tx} = c_2^n + 1 + \left\lceil \frac{n}{2} \right\rceil = \frac{n(n-1)}{2} + 1 + \left\lceil \frac{n}{2} \right\rceil$$

Una vez discutido el caso de número de nodos impares, se aborda el caso de un numero n par. Para ello, se usa como ayuda el caso de $n = 4$. La tabla A.2 muestra un análisis similar al realizado en la tabla A.1. La secuencia empleada en este caso es $'\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{1}, \mathbf{4}, \mathbf{2}, \mathbf{4}, \mathbf{1}, \mathbf{3}, \mathbf{2}'$.

Number of transmission	1	2	3	4	5	6	7	8	9	10
Schedule	**1**	**2**	**3**	**1**	**4**	**2**	**4**	**1**	**3**	**2**
Node 1	0	1	2	2	3	4	5	5	6	7
Node 2	0	0	1	2	3	3	4	5	6	6
Node 3	0	1	1	2	3	4	5	6	6	7
Node 4	0	1	2	3	3	4	4	5	6	7

Tabla A.2: Número de ecuaciones en cada paso, $n = 4$

Este análisis se hace por separado porque en el caso de número par de nodos aparece un pequeño detalle a tener en cuenta. En este caso, el mínimo número de transmisiones es igual a $n_{blocks} + 1$. Observando las tablas A.1 o A.2 en las transmisiones $n_{blocks} + 1$ se ve que todos los nodos han generado el mismo número de ecuaciones. Esto significa que en la siguiente emisión de un pulso por parte del nodo que toque, el mínimo número de ecuaciones generadas será el mismo ya que ese nodo que ha transmitido no generará una y se quedará con las mismas. Este hecho puede observarse en la tabla A.2 donde en la

transmisión 9, todos los nodos han generado 6 ecuaciones. Entonces, el nodo 2 transmite, quedándose en la décima tranmisión con el mismo número de pulsos. Si es el caso de que el número de nodos en la red es par, el número de ecuaciones necesario para generar un sistema completo coincide exactamente en una transmisión que se puede expresar como $n_{blocks} + 1$, es decir, donde todos los nodos poseen las mismas ecuaciones. Quiere decir que al definir la variable x del caso anterior, hay que tener en cuenta que es necesaria una transmisión menos. De esta forma, x se define como:

$$x = N_{eq}(n_{blocks}) - N_{eq} + 1 \tag{A.9}$$

De esta forma, empleando A.4, A.5 y A.9 en A.7, la expresión del número mínimo de transmisiones necesarias en este caso es:

$$N_{tx} = c_2^n + \left\lceil \frac{c_2^n + 1}{n-1} \right\rceil \tag{A.10}$$

Como en el caso anterior, la parte interior de $\lceil . \rceil$ se puede manipular de la siguiente forma:

$$\frac{c_2^n + 1}{n-1} = \frac{n}{2} + \frac{1}{n-1} = 1 + \frac{n}{2} + \frac{1}{n-1} - \frac{n-1}{n-1} = 1 + \frac{n}{2} - \frac{n-2}{n-1}$$

y A.7 puede reescribirse como:

$$N_{tx} = c_2^n + 1 + \lceil \frac{n}{2} - \frac{n-2}{n-1} \rceil \tag{A.11}$$

Se sabe que n es par, luego el primer miembro del interior de $\lceil . \rceil$ es un entero. El segundo miembro toma valores que oscilan entre $[0, 1)$ para número de nodos mayor que 2. En este caso al realizar la resta, se le está restando a un número entero un número positivo menor que 1, luego al redondear hacia arriba siempre quedará el mismo valor, pudiéndose suprimir el segundo miembro dentro del $\lceil . \rceil$. Así, A.7 es obtenida:

$$N_{tx} = c_2^n + 1 + \lceil \frac{n}{2} \rceil = \frac{n(n-1)}{2} + 1 + \lceil \frac{n}{2} \rceil$$

Donde N_{eq} se ha sustituido por c_2^n.

De esta forma, se ha probado que la expresión 2.6 es correcta.

Anexo B

Descomposición QR

La descomposición QR es una forma de descomponer una matriz de tamaño genérico $M \times N$ en el producto de dos matrices, $\mathbf{Q}$ y $\mathbf{R}$, cada una con distintas características:

$$\mathbf{A} = \mathbf{QR} \tag{B.1}$$

Donde $\mathbf{A} = [\mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_N}]$. Cada componente $\mathbf{a_i}$ es un vector columna que forma parte de la matriz. La matriz $\mathbf{Q} = [\mathbf{q_1}, \mathbf{q_2}, \ldots, \mathbf{q_M}]$ es una matriz de tamaño $M \times M$ donde todos los vectores columna son ortogonales entre sí, esto es, el producto escalar entre ellos da cero como resultado.

$$< \mathbf{q_i}, \mathbf{q_j} >= 0, \forall i \neq j \tag{B.2}$$

La matriz $\mathbf{Q}$ tiene como peculiaridad que su inversa es igual a su traspuesta. Esto es una ventaja importante dado que invertir esta matriz es computacionalmente muy simple.

$$\mathbf{Q}^{-1} = \mathbf{Q}^T \tag{B.3}$$

La matriz $\mathbf{R} = [\mathbf{r_1}, \mathbf{r_2}, \ldots, \mathbf{r_N}]$ es una matriz triangular superior de tamaño $M \times N$, donde todos los elementos por debajo de su diagonal principal son iguales a cero. La principal ventaja de obtener una matriz de estas características es que permite el uso de métodos como la sustitución, con relativa simplicidad computacional, para obtener su inversa.

Una vez definido el concepto de la descomposición QR, es hora de introducir diversas formas de realizar este tipo de descomposición de matrices. Como la complejidad computacional es un tema muy a tener en cuenta a la hora de implementar algoritmos en hardware, se ha realizado un análisis detallado de la misma en cada uno de los métodos expuestos. Tres son los distintos métodos introducidos, rotaciones de Givens, la transformación de Householder y la ortogonalización de Gram-Schmidt.

B.1 La ortogonalización de Gram-Schmidt

El primer método expuesto es la ortogonalización de Gram-Schmidt. Este método difiere de lo descrito anteriormente en el tamaño de las matrices. En este caso, la descomposición se realiza de forma $(M \times N) = (M \times N)(N \times N)$. Este método consiste en generar vectores columna ortonormales mediante el cálculo de proyecciones vectoriales. Cada vector nuevo se obtiene teniendo en cuenta todos los obtenidos previamente hasta ese instante, esto es, el primer vector columna es independiente, el segundo depende de la proyección del primer vector con el segundo, el tercero depende de las proyecciones del primero y segundo sobre el tercero, y así sucesivamente hasta el último vector. La ecuación B.4 muestra la forma de calcular la proyección de un vector $\mathbf{a}$ sobre otro vector $\mathbf{e}$

$$proj_{\mathbf{e}}(\mathbf{a}) = < \mathbf{e}, \mathbf{a} > \mathbf{e} \tag{B.4}$$

Siendo $<,>$ el operador producto escalar definido como:

$$< \mathbf{e}, \mathbf{a} >= e_1 a_1 + e_2 a_2 + e_3 a_3 + \cdots + e_M a_M \tag{B.5}$$

Una vez definido cómo calcular las proyecciones sobre los vectores, el procedimiento para generar vectores ortogonales es el siguiente:

$$\begin{aligned}
\mathbf{u}_1 &= \mathbf{a}_1 \\
\mathbf{u}_2 &= \mathbf{a}_2 - proj_{\mathbf{e}_1}(\mathbf{a}_2) \\
\mathbf{u}_3 &= \mathbf{a}_3 - proj_{\mathbf{e}_1}(\mathbf{a}_3) - proj_{\mathbf{e}_2}(\mathbf{a}_3) \\
&\cdots \\
\mathbf{u}_N &= \mathbf{a}_N - \sum_{j=1}^{N} proj_{\mathbf{e}_j}(\mathbf{a}_N)
\end{aligned}$$

Siendo $\mathbf{a}_k$ el vector columna k de la matriz original $\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3. \cdots, \mathbf{a}_N]$, $\mathbf{u}_k$ el nuevo vector ortogonal asociado al vector $\mathbf{a}_k$, y $\mathbf{e}_k$ es el vector unitario correspondiente al vector $\mathbf{u}_k$, calculado como:

$$\mathbf{e}_k = \frac{\mathbf{u}_k}{\|\mathbf{u}_k\|} \tag{B.6}$$

Donde $\| * \|$ es el operador norma. La matriz $\mathbf{Q}$ se forma concatenando cada vector $\mathbf{e}_k$ por columnas.

$$\mathbf{Q} = [\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \cdots, \mathbf{e}_N] \tag{B.7}$$

Para generar **R**, se necesitan varios productos escalares. La equation B.8 se emplea para obtener **R**:

$$\mathbf{R} = \begin{bmatrix} < \mathbf{e_1}, \mathbf{a_1} > & < \mathbf{e_1}, \mathbf{a_2} > & < \mathbf{e_1}, \mathbf{a_3} > & \cdots \\ 0 & < \mathbf{e_2}, \mathbf{a_2} > & < \mathbf{e_2}, \mathbf{a_3} > & \cdots \\ 0 & 0 & < \mathbf{e_3}, \mathbf{a_3} > & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{B.8}$$

B.1.1 Complejidad computacional de la ortogonalización de Gram-Schmidt

Para comenzar con el análisis de la complejidad, el primer paso es observar la ecuación del cálculo de las proyecciones. La tabla B.1 resume el número de operaciones al calcular la proyección de un vector sobre otro, en este caso.

Equation	**Multiplications**	**Additions**	**Divisions**	**Square roots**
$< \mathbf{e}, \mathbf{a} >$	M	M-1	0	0
$< \mathbf{e}, \mathbf{a} > \mathbf{e}$	M	0	0	0
Total projection	2M	M-1	0	0

Tabla B.1: Complejidad del cálculo de una proyección vectorial

El siguiente paso es calcular el número de operaciones necesarias para normalizar un vector. La tabla B.2 muestra los resultados.

Equation	**Multiplications**	**Additions**	**Divisions**	**Square roots**
$\|\mathbf{u}\| = \sqrt{u_1^2 + u_2^2 + \cdots + u_M^2}$	M	M-1	0	1
$\frac{\mathbf{u}}{\|\mathbf{u}\|}$	0	0	M	0
Total normalization	M	M-1	M	1

Tabla B.2: Complejidad del cálculo de una normalización vectorial

Una vez discutida la complejidad de las proyecciones y la normalización, en análisis se hace paso a paso. La tabla B.3 muestra la complejidad de cada paso en el algoritmo.

Vector	**Mults**	**Adds**	**Divs**	**Sqrts**
$\mathbf{u}_1 = \mathbf{a}_1; \mathbf{e}_1 = \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|}$	M	M-1	M	1
$\mathbf{u}_2 = \mathbf{a}_2 - proj_{\mathbf{e}_1}(\mathbf{a}_2); \mathbf{e}_2 = \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|}$	3M	3M-2	M	1
$\mathbf{u}_3 = \mathbf{a}_3 - proj_{\mathbf{e}_2}(\mathbf{a}_3) - proj_{\mathbf{e}_2}(\mathbf{a}_3); \mathbf{e}_3 = \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|}$	5M	5M-3	M	1
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\mathbf{u}_N = \mathbf{a}_N - \sum_{j=1}^{N} proj_{\mathbf{e}_j}(\mathbf{a}_N); \mathbf{e}_N = \frac{\mathbf{u}_N}{\|\mathbf{u}_N\|}$	(2N-1)M	(2N-1)M-N	M	1

Tabla B.3: Complejidad de cada paso de la ortogonalización de Gram-Schmidt

En este punto, el número de operaciones de cada paso es conocido. Para obtener la complejidad total del algoritmo ha de sumarse la contribución de cada paso al número

de operaciones.

Multiplicaciones

$$\begin{aligned} N_{mults} &= \sum_{n=1}^{N}(2n-1)M \\ &= 2M\left[\frac{N(N+1)}{2}\right] - MN \\ &= MN^2 \end{aligned} \tag{B.9}$$

Sumas

$$\begin{aligned} N_{mults} &= \sum_{n=1}^{N}(2n-1)M - N \\ &= 2M\left[\frac{N(N+1)}{2}\right] - MN - \frac{N(N+1)}{2} \\ &= MN^2 - \frac{N^2}{2} - \frac{N}{2} \end{aligned} \tag{B.10}$$

Divisiones

$$\begin{aligned} N_{mults} &= \sum_{n=1}^{N} M \\ &= MN \end{aligned} \tag{B.11}$$

Raíces cuadradas

$$\begin{aligned} N_{mults} &= \sum_{n=1}^{N} 1 \\ &= N \end{aligned} \tag{B.12}$$

Todas estas expresiones se muestran en la tabla B.4:

Tomando todos los tipos de operaciones como operaciones de punto flotante (*flops*), la complejidad total del algoritmo puede ser analizada. La última fila de la tabla B.4 muestra la complejidad total de la ortogonalización de Gram-Schmidt.

La tabla B.5 indica el orden de las operaciones en el caso $M = N$.

Gram-Schmidt	**Operations**	**Order**
Multiplications	MN^2	$\mathcal{O}\left(MN^2\right)$
Additions	$MN^2 - \frac{N^2}{2} - \frac{N}{2}$	$\mathcal{O}\left(MN^2\right)$
Divisions	MN	$\mathcal{O}\left(MN\right)$
Square roots	N	$\mathcal{O}\left(N\right)$
Flops	$2MN^2 - \frac{N^2}{2} - MN + \frac{N}{2}$	$\mathcal{O}\left(2MN^2\right)$

Tabla B.4: Complejidad total de la ortogonalización de Gram-Schmidt

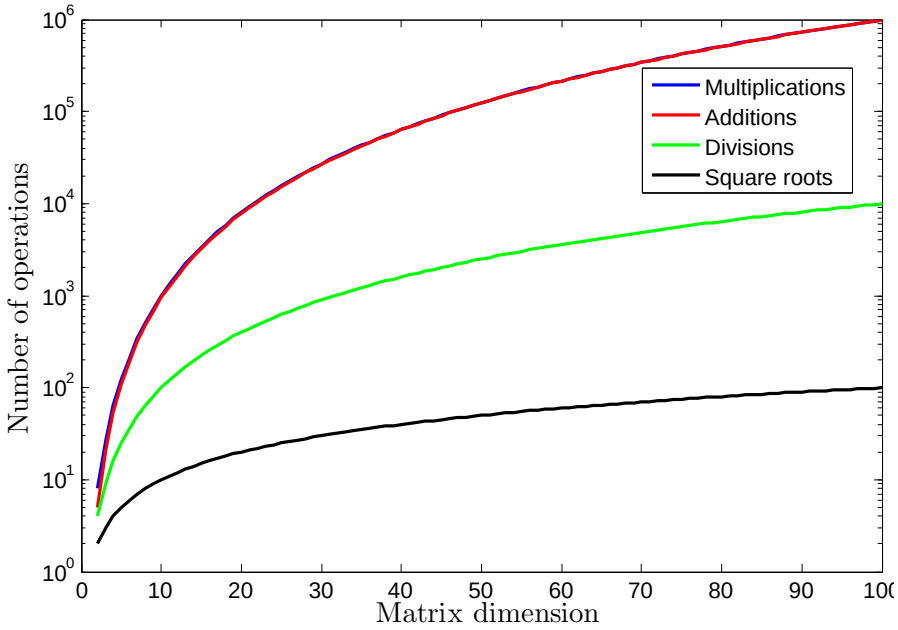

Figura B.1: Complejidad de la ortogonalización de Gram-Schmidt frente al tamaño de la matriz

Order of Operations	$M = N$
Multiplications	M^3
Additions	M^3
Divisions	M^2
Square roots	M
Total	$2M^3$

Tabla B.5: Orden de complejidad total de la ortogonalización de Gram-Schmidt

B.2 La transformación de Householder

El segundo método de realizar la descomposición QR es la transformación de Householder. Se basa en generar una matriz auxiliar **H** que multiplicada por la original, crea otra matriz con el primer vector columna formado por ceros excepto el primer elemento. Siendo **A** la matriz $M \times N$ original, la metodología puede explicarse mediante la expresión B.13:

$$\mathbf{H}_1\mathbf{A} = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1M} \\ 0 & & & \\ \vdots & & A' & \\ 0 & & & \end{bmatrix} \tag{B.13}$$

Como puede observarse, se obtiene una matriz $\mathbf{A}'$ más pequeña. Repitiendo este proceso, tomando como matriz a transformar las sucesivas $\mathbf{A}'$ obtenidas en cada paso, se obtiene una matriz triangular **R**. El conjunto de ecuaciones necesarias para generar esta matriz **H** es:

$$\mathbf{u} = \mathbf{a} + \alpha\mathbf{e} \tag{B.14}$$

$$\mathbf{v} = \frac{\mathbf{u}}{\|\mathbf{u}\|} \tag{B.15}$$

$$\mathbf{H} = \mathbf{I} - 2\mathbf{v}\mathbf{v}^T \tag{B.16}$$

Donde **a** es el vector en la matriz original a ser transformado en un vector de ceros, α es la norma del vector **a**, $\mathbf{e} = [1, 0, 0, \ldots]^T$ y **I** es la matriz identidad del tamaño adecuado.

Una vez se calcula **R** ha de obtenerse **Q**. Como cada vez que se crea una nueva matriz **H**, el tamaño de la misma disminuye, la multiplicación de matrices estándar no puede realizarse. La solución es insertar cada una de estas matrices dentro de la matriz identidad del tamaño de la matriz **A**. La ecuación B.17 muestra esta técnica.

$$\mathbf{H}_k = \begin{bmatrix} \mathbf{I}_{k-1} & 0 \\ 0 & \mathbf{H}_k' \end{bmatrix} \tag{B.17}$$

Ahora, la multiplicación de matrices puede realizarse y la matriz **Q** puede calcularse mediante la ecuación B.18:

$$\mathbf{Q} = \mathbf{H}_1^T\mathbf{H}_2^T\mathbf{H}_3^T \cdots \mathbf{H}_{M-1}^T \tag{B.18}$$

Y

$$\mathbf{R} = \mathbf{Q}^T\mathbf{A} \tag{B.19}$$

B.2.1 Complejidad computacional de la transformación de Householder

Como en el método anterior, la complejidad se estudia paso a paso. Lo primero, es conveniente reagrupar todas las ecuaciones del algoritmo para analizar de una forma más clara el número total de operaciones.

$\mathbf{R} = \mathbf{A}$

$\mathbf{Q} = \mathbf{I}_{M\times M}$

for $i = 1 : min(M-1, N)$

1. $\mathbf{r} = \mathbf{R}(k : M, k)$
2. $\mathbf{v} = \mathbf{r}$
3. $\mathbf{v}(1) = \mathbf{v}(1) + sign(\mathbf{r}(1))||\mathbf{r}||$
4. $\mathbf{v} = \frac{\mathbf{v}}{||\mathbf{v}||}$
5. $\mathbf{R}(k : M, k : N) = \mathbf{R}(k : M, k : N) - 2\mathbf{v}\left(\mathbf{v}^T\mathbf{R}(k : M, k : N)\right)$
6. $\mathbf{Q}(k : M, k : N) = \mathbf{Q}(k : M, k : N) - 2\mathbf{v}\left(\mathbf{v}^T\mathbf{Q}(k : M, k : N)\right)$

Mirando el algoritmo, puede verse que las líneas 1 y 2 no requieren operación alguna por lo tanto el análisis estará centrado en las líneas 3 - 6. Para contar las operaciones, se definen dos variables, $l = M - k + 1$ y $p = N - k + 1$, las cuales representan los tamaños de los vectores y matrices en cada paso k. La tabla B.6 muestra la complejidad de cada ecuación, y el total para una iteración.

	Multiplications	**Additions**	**Divisions**	**Square roots**
3.	l	l	0	1
4.	l	$l-1$	l	1
5.	$2lp + l$	$2lp - p$	0	0
6.	$2lp + l$	$2lp - p$	0	0
Total	$4lp + 4l$	$4lp + 2l - 2p - 1$	l	2

Tabla B.6: Complejidad de cada paso en la transformación de Householder

Una vez se tienen las expresiones para cada operación, es posible calcular el número total mediante la expresión B.20:

$$N_{op} = \sum_{k=1}^{\text{mín}(M-1,N)} N_{op_k} \tag{B.20}$$

Donde N_{op_k} es la expresión de cada una de las operaciones en cada paso del algoritmo.

Cálculos detallados: M-1 $\leq$ N

Mirando la tabla B.6:

Multiplicaciones

$$\begin{aligned} N_{mults} &= \sum_{k=1}^{M-1} (4lp + 4l) \\ &= \sum_{k=1}^{M-1} 4(M-k+1)(N-k+1) + 4(M-k+1) \\ &= \frac{-2}{3}M^3 + (2N+2)\,M^2 + \left(2N + \frac{20}{3}\right) M \\ &- 4N - 8 \end{aligned} \tag{B.21}$$

Sumas

$$\begin{aligned} N_{adds} &= \sum_{k=1}^{M-1} 4lp + 2l - 2p - 1 \\ &= \sum_{k=1}^{M-1} 4(M-k+1)(N-k+1) + 2(M-k+1) - 2(N-k+1) - 1 \\ &= \frac{-2}{3}M^3 + (2N+2)\,M^2 - \frac{5}{3}M - 2N - 3 \end{aligned} \tag{B.22}$$

Divisiones

$$\begin{aligned} N_{divs} &= \sum_{k=1}^{M-1} l \\ &= \sum_{k=1}^{M-1} [M-k+1)] \\ &= \frac{1}{2}M^2 + \frac{1}{2}M - 1 \end{aligned} \tag{B.23}$$

Raíces cuadradas

$$\begin{aligned} N_{sqrts} &= \sum_{k=1}^{M-1} 2 \\ &= 2M-2 \end{aligned} \tag{B.24}$$

Cálculos detallados: M-1 > N

Mirando la tabla B.6:

Multiplicaciones

$$\begin{aligned} N_{mults} &= \sum_{k=1}^{N} (4lp+4l) \\ &= \sum_{k=1}^{N} 4(M-k+1)(N-k+1)+4(M-k+1) \\ &= \frac{-2}{3}N^3+\left(2M-\frac{5}{2}\right)N^2+\left(6M+\frac{8}{3}\right)N \end{aligned} \tag{B.25}$$

Sumas

$$\begin{aligned} N_{adds} &= \sum_{k=1}^{N} 4lp+2l-2p-1 \\ &= \sum_{k=1}^{N} 4(M-k+1)(N-k+1)+2(M-k+1)-2(N-k+1)-1 \\ &= \frac{-2}{3}M^3+(2M-2)M^2+\left(4M-\frac{1}{3}\right)N \end{aligned} \tag{B.26}$$

Divisiones

$$\begin{aligned} N_{divs} &= \sum_{k=1}^{N} l \\ &= \sum_{k=1}^{N} [M-k+1)] \end{aligned}$$

$$= \frac{1}{2}N^2 + \left(M - \frac{1}{2}\right)N \tag{B.27}$$

Raíces cuadradas

$$\begin{aligned} N_{sqrts} &= \sum_{k=1}^{N} 2 \\ &= 2N \end{aligned} \tag{B.28}$$

$M-1 \leq N$	**Operations**
Multiplications	$\frac{-2}{3}M^3 + (2N-2)M^2 + \left(2N + \frac{20}{3}\right)M - 4N - 8$
Additions	$\frac{-2}{3}M^3 + (2N+2)M^2 - \frac{5}{3}M - 2N - 3$
Divisions	$\frac{1}{2}M^2 + \frac{1}{2}M - 1$
Square roots	$2M - 2$
$M-1 > N$	**Operations**
Multiplications	$\frac{-2}{3}N^3 + (2M-2)N^2 + \left(6M + \frac{8}{3}\right)N$
Additions	$\frac{-2}{3}N^3 + (2M-2)N^2 + \left(4M - \frac{1}{3}\right)N$
Divisions	$-\frac{1}{2}N^2 + \left(M - \frac{1}{2}\right)N$
Square roots	$2N$

Tabla B.7: Complejidad total de la transformación de Householder

La tabla B.7 muestra una compilación de todas las expresiones de este apartado.

Mirando los cálculos anteriores, se puede decucir el orden de complejidad del algoritmo. Los resultados se muestran en la tabla B.8

Order of Operations	$M-1 \leq N$	$M-1 > N$
Multiplications	$2NM^2 - \frac{2}{3}M^3$	$2N^2M - \frac{2}{3}N^3$
Additions	$2NM^2 - \frac{2}{3}M^3$	$2N^2M - \frac{2}{3}N^3$
Divisions	$\frac{1}{2}M^2$	$MN - \frac{1}{2}N^2$
Square roots	$2M$	$2N$

Tabla B.8: Orden de complejidad de la transformación de Householder para cada operación

Contando cada operación como un flop, el orden total en términos de número de flops puede obtenerse. La tabla B.9 muestra el orden dependiendo del tamaño de la matriz original.

Order of Householder	$M-1 \leq N$	$M-1 > N$
FLOPS	$4NM^2 - \frac{4}{3}M^3$	$4N^2M - \frac{4}{3}N^3$

Tabla B.9: Orden de complejidad en términos de flops de la transformación de Householder para cada operación

Para realizar una mejor comparación entre métodos, se calculan los órdenes de

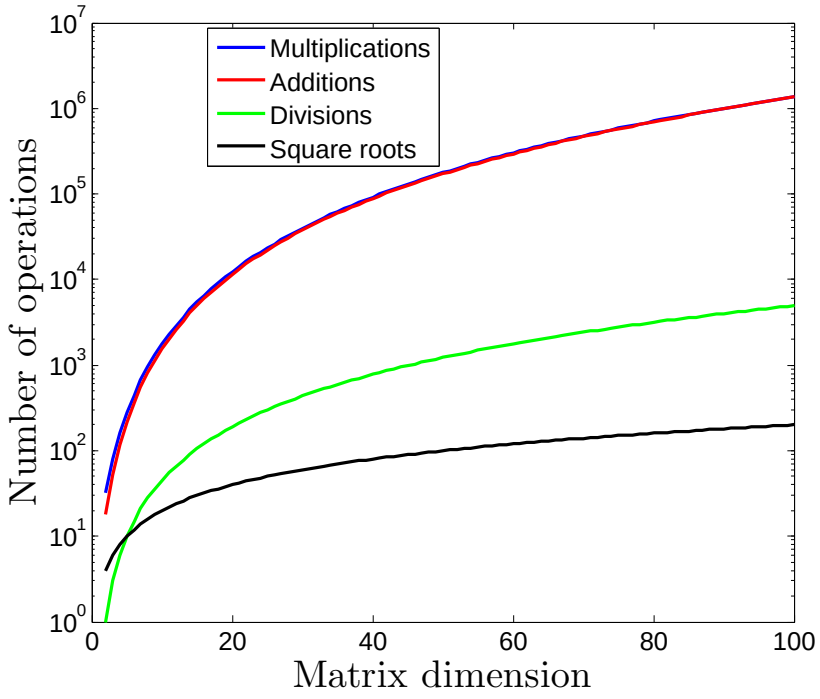

Figura B.2: Complejidad de la transformación de Householder frente al tamaño de la matriz $M \times M$

complejidad para el caso $M = N$. La tabla B.10 muestra el orden de complejidad en este caso.

Order of Operations	$M = N$
Multiplications	$\frac{4}{3}M^3$
Additions	$\frac{4}{3}M^3$
Divisions	$\frac{1}{2}M^2$
Square roots	$2M$
Total	$\frac{8}{3}M^3$

Tabla B.10: Orden de complejidad de la transformación de Householder en el caso $M = N$

B.3 Las rotaciones de Givens

B.3.1 El concepto

Las rotaciones de Givens son una técnica empleada para generar ceros en matrices. Se basa en rotaciones de vectores en planos. Estas rotaciones se representan en una matriz $M \times M$, $\mathbf{G}$, conocida como *matriz de rotación*. La matriz $\mathbf{G}$ se define como:

$$G(i,j) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & -s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \tag{B.29}$$

Donde c y s aparecen en la fila i y columna j correspondientes a la posición del elemento a_{ij} en la matriz original $\mathbf{A}$, el cual va a ser transformado para ser un cero. Esto es $\mathbf{G}$ es una matriz identidad donde:

$$g_{ii} = g_{jj} = c$$

$$g_{ji} = -s$$

$$g_{ij} = s$$

Una vez se conoce esta matriz $\mathbf{G}$, un cero en la posición deseada puede ser generado mediante la multiplicación de estas dos matrices, la matriz de rotación y la matriz original. El problema ahora es cómo calcular estos valores c y s. Tan solo dos elementos de la matriz $\mathbf{A}$ son necesarios, el que va a ser transformado a cero, y otro auxiliar, perteneciente al mismo vector columna, el cual es deseable que esté situado en la diagonal de la matriz. Una vez seleccionados estos dos elementos, el problema puede ser reducido a:

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} a_{ii} \\ a_{ij} \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix} \tag{B.30}$$

Definido esto, tres valores desconocidos se obtienen, c, s y r. Estos valores se calculan mediante las ecuaciones B.31, B.32 y B.33

$$r = \sqrt{a_{ii}^2 + a_{ij}^2} \tag{B.31}$$

$$c = \frac{a_{ii}}{r} \tag{B.32}$$

$$s = -\frac{a_{ij}}{r} \tag{B.33}$$

$$\tag{B.34}$$

Conocidos estos valores, el cero puede ser generado en la posición deseada. Generando ceros en todas las posiciones por debajo de la diagonal de $\mathbf{A}$ mediante varias multiplicacionesde matrices, la matriz triangular $\mathbf{R}$ puede ser calculada. En este punto,

R es conocida, pero **Q** es aun desconocida. La matriz **Q** puede obtenerse multiplicando las sucesivas matrices de rotación **G** empleadas para generar **R**. Las matrices **G** son unitarias, por lo tanto, la matriz resultante de estas multiplicaciones será también unitaria. La ecuación B.35 muestra cómo calcular **Q**.

$$\mathbf{Q} = \mathbf{G}_1^T \mathbf{G}_2^T \mathbf{G}_3^T \mathbf{G}_4^T \dots \tag{B.35}$$

Donde $\mathbf{G}_i$ es cada una de las matrices de rotación de cada cero generado. Al final:

$$\mathbf{A} = [\mathbf{G}_1^T \mathbf{G}_2^T \mathbf{G}_3^T \mathbf{G}_4^T \dots]\mathbf{R} \tag{B.36}$$

Debido a la alta complejidad computacional del método, del orden de M^5 para $M = N$, ha sido descartado como opción, por eso los cálculos no se detallan.

B.4 Comparación entre métodos

La figura B.3 muestra una comparación entre los métodos de la ortogonalización de Gram-Schmidt y la transformación de Householder en matrices cuadradas $M = N$. No se incluye el método de las rotaciones de Givens debido a su alta complejidad computacional, y así de esta forma se visualiza más claramente la comparación. La última figura, la figura B.4, muestra el número total de flops en cada método. Puede verse la diferencia entre las rotaciones de Givens y los otros dos métodos, dos órdenes de magnitud. Para observar mejor las diferencias entre los dos métodos menos complejos, la figura B.3 muestra las flops sin las rotaciones de Givens. La tabla B.11 resume el total de las complejidades de los tres métodos, en el caso $M = N$.

Order of Operations	**Gram-Schmidt**	**Householder**	**Givens Rotations**
Multiplications	M^3	$\frac{4}{3}M^3$	M^5
Additions	M^3	$\frac{4}{3}M^3$	M^5
Divisions	M^2	$\frac{1}{2}M^2$	M^2
Square roots	M	$2M$	$\frac{1}{2}M^2$
Total	$2M^3$	$\frac{8}{3}M^3$	$2M^5$

Tabla B.11: Órdenes de complejidad, $M = N$

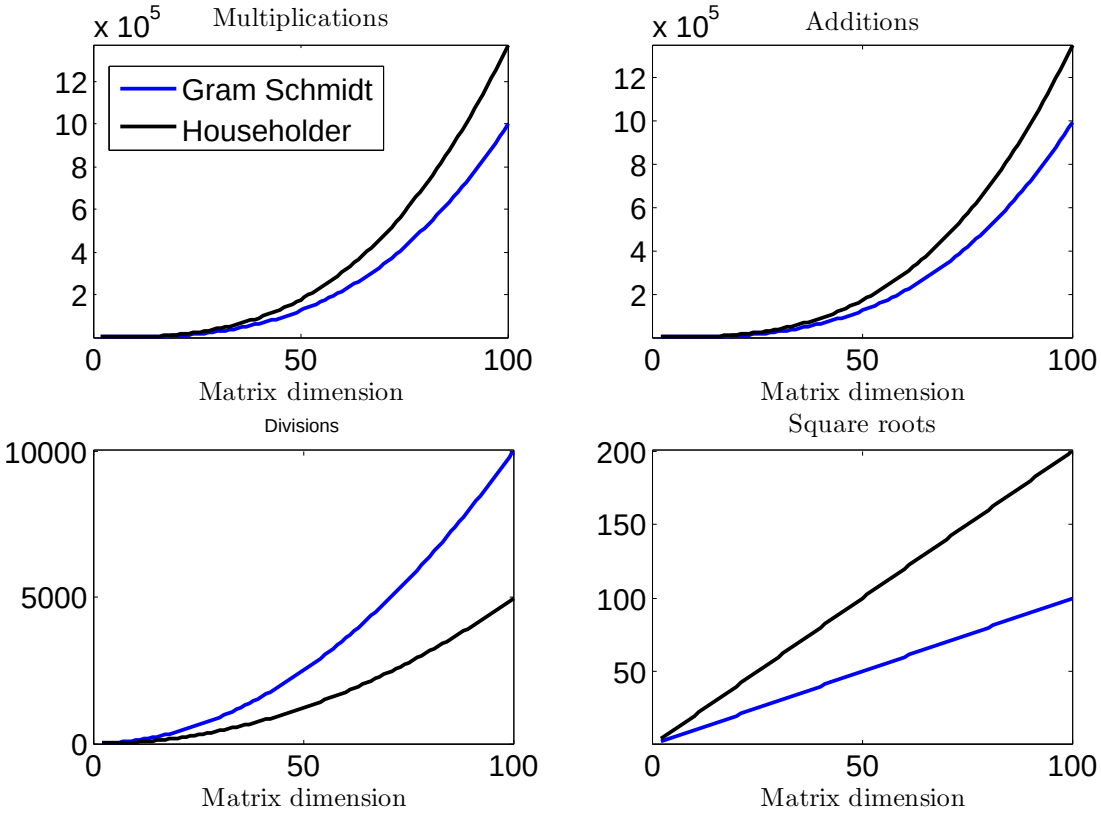

Figura B.3: Comparación de operaciones en diferentes métodos

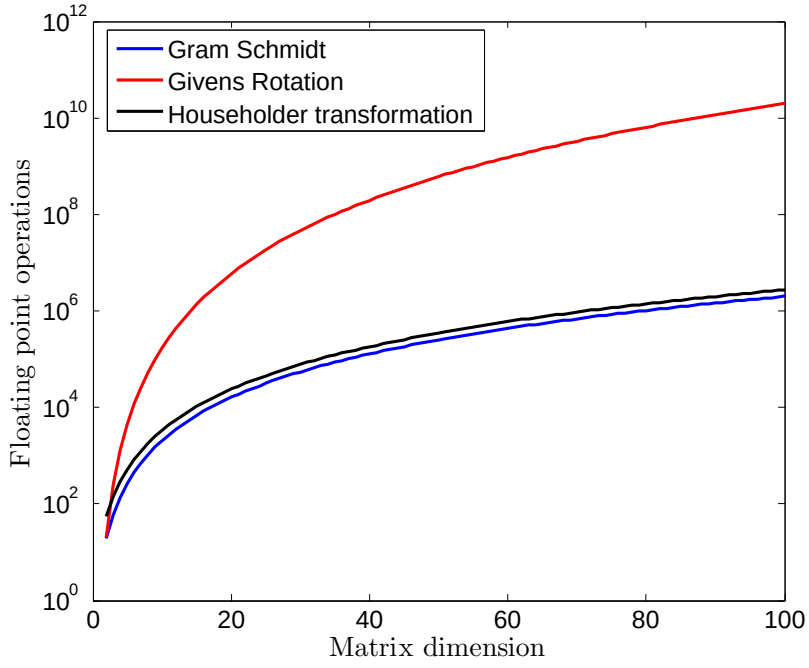

Figura B.4: Comparación de operaciones en diferentes métodos

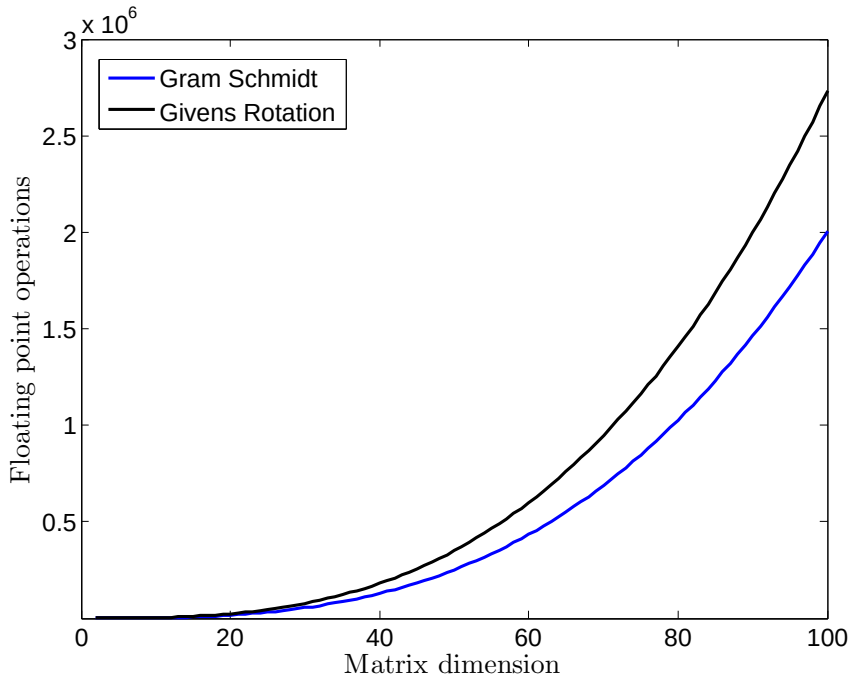

Figura B.5: Comparación de flops en la ortogonalización de Gram-Schmidt y en la transformación de Householder

Anexo C

Códigos de secuencias

C.1 Generar una secuencia

```
1  %% Number of interconnections
2  n_nodes = 3:20;
3  n_connections = n_nodes.*(n_nodes-1)./2;
4  %% Minimum number of transmissions
5  n_transmissions = n_connections + 1 + ceil(n_nodes./2);
6  %% Scheduling
7  n = 7;
8  transmissions = n*(n-1)/2 + 1 + ceil(n/2);
9  checkmatrix = eye(n);
10 sequence = zeros(1,transmissions);
11 last_node = 1;
12 sequence(1) = 1;
13 for i=2:transmissions
14         possible_nodes = find(checkmatrix(last_node,:)==0);
15         checksubmatrix = checkmatrix(:,possible_nodes);
16         index_node_tx = morezeros(checksubmatrix);
17         node_tx = possible_nodes(index_node_tx);
18         sequence(i) = node_tx;
19         checkmatrix(last_node,node_tx) = 1;
20         last_node = node_tx;
21 end
```

C.2 Generar todas las secuencias

C.2.1 Principal

```
n = number of nodes;
schedule = 1;
checkmatrix = eye(n);
n_tx = 1;
fid = fopen('sch_5.rtf','w');
add_node(n,schedule,checkmatrix,n_tx,fid)
fclose(fid);
```

C.2.2 Función recursiva

```
function add_node(n,schedule,checkmatrix,n_tx,fid)
nodes = find(checkmatrix(schedule(end),:) == 0);
if length(nodes) == 0
    return
else
    for i = 1:length(nodes)
        nodes;
        new_schedule = [schedule nodes(i)];
        n_tx = n_tx + 1;
        checkmatrix(schedule(end),nodes(i)) = 1;
        if n_tx == n*(n-1)/2 + 1 + ceil(n/2)
            for k = 1:n
                n_tx_each_node(k) = length(find(new_schedule == k));
            end
            if (max(n_tx_each_node) - min(n_tx_each_node)) < 2 &&
             max(n_tx_each_node) == n_tx_each_node(1)
                formatSpec = '%s\n';
                fprintf(1,formatSpec, int2str(new_schedule));
            end
        end
        add_node(n,new_schedule,checkmatrix,n_tx,fid);
        n_tx = n_tx - 1;
        checkmatrix(schedule(end),nodes(i)) = 0;
    end
end
```

Anexo D

Códigos de la descomposición de matrices

D.1 Ortogonalización de Gram-Schmidt

```
1  function [Q,R,mults,adds,divs,sqrts] = gram_schmidt(A)
2      M = size(A,1);
3      Q = A;
4      R = zeros(M,M);
5      mults = 0;
6      adds = 0;
7      divs = 0;
8      sqrts = 0;
9  for k = 1:M
10     for i = 1:k-1
11         s = 0;
12         for j = 1:M
13             s = s + Q(j,i)*Q(j,k);
14             mults = mults + 1;
15             adds = adds + 1;
16         end
17         adds = adds - 1;
18         R(i,k) = s;
19     end
20     for i = 1:k-1
21         for j = 1:M
22             Q(j,k) = Q(j,k) - Q(j,i)*R(i,k);
23             mults = mults + 1;
24             adds = adds + 1;
```

```
25          end
26      end
27      s = 0;
28      for j = 1:M
29          s = s + Q(j,k)*Q(j,k);
30          mults = mults + 1;
31          adds = adds + 1;
32      end
33      adds = adds - 1;
34      R(k,k) = sqrt(s);
35      sqrts = sqrts + 1;
36      for j = 1:M
37          Q(j,k) = Q(j,k)/R(k,k);
38          divs = divs + 1;
39      end
40  end
```

D.2 Transformación de Householder

```
1   function [Q,R,mults,adds,divs,sqrts] = householder(A)
2       mults = 0;
3       adds = 0;
4       divs = 0;
5       sqrts = 0;
6       [m,n] = size(A);
7       Q = eye(m);
8       R = A;
9       QQ = eye(m);
10  for j = 1:n
11      l=m-j+1;
12      [normx,mults_n,adds_n,divs_n,sqrts_n] = normalize(R(j:end,j));
13      s = -sign(R(j,j));
14      u1 = sign(R(j,j))*normx*QQ(j:end,j) + R(j:end,j);
15      mults = mults + l;
16      adds = adds + l;
17      sqrts = sqrts + 1;
18      [normu,mults_u,adds_u,divs_u,sqrts_u] = normalize(u1);
19      w = u1/normu;
20      mults = mults + l;
21      adds = adds + l - 1;
22      divs = divs + l-1;
```

```
23      sqrts = sqrts + 1;
24      tau = -s*u1/normx;
25      R(j:end,:) = R(j:end,:)-2*w*(w'*R(j:end,:));
26      mults = mults + 2*l^2+l;
27      adds = adds + 2*l^2-l;
28      Q(j:end,:) = Q(j:end,:)-2*w*(w'*Q(j:end,:));
29      mults = mults + 2*l^2+l;
30      adds = adds + 2*l^2-l;
31  end
32  Q = Q';
```

D.3 Las rotaciones de Givens

```
1   function [Q,R,mults,adds,divs,sqrts] = givens_rotation(A)
2       mults = 0;
3       adds = 0;
4       divs = 0;
5       sqrts = 0;
6
7       M = size(A,1);
8       matrix = A;
9       Q = eye(M);
10
11  for column = 1:M
12      for row = column:M-1
13          i = column;
14          j = row + 1;
15          a = matrix(i,i);
16          b = matrix(j,i);
17          ab =[a;b];
18
19          r = sqrt(a^2+b^2);
20          mults = mults + 2;
21          adds = adds + 1;
22          sqrts = sqrts + 1;
23          c = a/r;
24          s =-b/r;
25          divs = divs + 2;
26          adds = adds + 1;
27
28          Qi = eye(M);
```

```
29          Qi(i,i) = c;
30          Qi(i,j) = -s;
31          Qi(j,i) = s;
32          Qi(j,j) = c;
33
34          adds = adds + 1;
35
36          matrix = Qi*matrix;
37          mults = mults + M^3;
38          adds = adds + M^2*(M-1);
39
40          Q = Q*Qi.';
41          mults = mults + M^3;
42          adds = adds + M^2*(M-1);
43      end
44  end
45  mults = mults - M^3;
46  adds = adds - M^2*(M-1);
47
48  R = matrix;
49  Q = Q;
```

D.4 La descomposición de Cholesky

```
1   function [L,D,mults,adds,divs,sqrts] = cholesky(A)
2   M = size(A);
3   mults = 0;
4   adds = 0;
5   divs = 0;
6   sqrts = 0;
7   L = eye(M);
8   D = zeros(M);
9   D(1,1) = A(1,1);
10  for j = 1:M
11      s = 0;
12      for k = 1:j-1
13          s = s + L(j,k)*L(j,k)*D(k,k);
14          mults = mults + 2;
15          adds = adds + 1;
16      end
17      D(j,j) = A(j,j) - s;
```

```
18      adds = adds + 1;
19      for i = 1:M
20          if i > j
21              s = 0;
22              for k = 1:j-1
23                  s = s + L(i,k)*L(j,k)*D(k,k);
24                  mults = mults + 2;
25                  adds = adds + 1;
26              end
27              L(i,j) = 1/D(j,j)*(A(i,j) - s);
28              mults = mults + 1;
29              adds = adds + 1;
30              divs = divs + 1;
31          end
32      end
33  end
34  end
```

Anexo E

Otros códigos

E.1 Funciones

E.1.1 Generar escenario

```
1  function [pos,scenario] = generate_scenario(size_x,size_y,n_nodes)
2  scenario = ones(size_x,size_y);
3  x_pos = ceil(size_x/2*rand(1,n_nodes)+size_x/4);
4  y_pos = ceil(size_y/2*rand(1,n_nodes)+size_x/4);
5  pos = [x_pos;y_pos].';
6  if n_nodes == 2
7      pos = [1 1 ; 1 1];
8  end
9  for i=1:n_nodes
10     scenario(x_pos(i),y_pos(i)) = 0;
11 end
```

E.1.2 Generar movimiento

```
1  function [pos_x,pos_y,v_x,v_y,a_x,a_y] = generate_movement
2  (x_init,y_init,vx_init,vy_init,a_strength,T,t,time_stop)
3
4  a_x = a_strength*randn(1,length(t));
5  a_y = a_strength*randn(1,length(t));
6
7  a_x(1:round(time_stop/T)) = zeros(1,round(time_stop/T));
8  a_y(1:round(time_stop/T)) = zeros(1,round(time_stop/T));
9
10 v_x(1) = vx_init;
11 v_y(1) = vy_init;
```

```
12 pos_x(1) = x_init;
13 pos_y(1) = y_init;
14 for i=2:length(t)
15     v_x(i) = v_x(i-1) + a_x(i-1)*T;
16     v_y(i) = v_y(i-1) + a_y(i-1)*T;
17     pos_x(i) = pos_x(i-1) + v_x(i-1)*T + 0.5*a_x(i-1)*T^2;
18     pos_y(i) = pos_y(i-1) + v_y(i-1)*T + 0.5*a_y(i-1)*T^2;
19 end
```

E.1.3 Invertir una matriz triangular superior

```
1  function [W,mults,adds,divs] = inverse_upper_triangle(R)
2  M = size(R,1);
3  W = zeros(M,M);
4  mults = 0;
5  adds = 0;
6  divs = 0;
7
8  for i = 1:M
9      W(i,i) = 1./R(i,i);
10     divs = divs + 1;
11 end
12 for diff = 1:M-1
13     for i = 1:M-1
14         j = i + diff;
15         if j<=M
16             fact = 0;
17             for k = i+1:j
18                 fact = fact + R(i,k)*W(k,j);
19                 mults = mults + 1;
20                 adds = adds + 1;
21             end
22             adds = adds - 1;
23             W(i,j) = -fact./R(i,i) ;
24             adds = adds + 1;
25             divs = divs + 1;
26             mults;
27         end
28     end
29 end
```

E.2 Código general de un escenario

```
1
2  %% Generate scenario
3  size_x = 1000;
4  size_y = 1000;
5  n_nodes = 3;
6  [pos,scenario] = generate_scenario(size_x,size_y,n_nodes);
7
8  %% Generate movement
9  T = 1;
10 total_time = 250;
11 t = 0:T:total_time-T;
12 time_stop = 0;
13
14 %Node 1
15 vx1_init = 0;
16 vy1_init = 0;
17 acc1 = 0.1;
18 [x11,y11,vx11,vy11,ax11,ay11] = generate_movement(pos(1,1),pos(1,2),
19 vx1_init,vy1_init,acc1,T,t,time_stop);
20
21
22 %Node 2
23 vx2_init = 0;
24 vy2_init = 0;
25 acc2 = 0.1;          % Static node 2
26 [x22,y22,vx22,vy22,ax22,ay22] = generate_movement(pos(2,1),pos(2,2),
27 vx2_init,vy2_init,acc2,T,t,time_stop);
28
29
30 %Node 3
31 vx3_init = 0;
32 vy3_init = 0;
33 acc3 = 0.1;
34 [x33,y33,vx33,vy33,ax33,ay33] = generate_movement(pos(3,1),pos(3,2),
35 vx3_init,vy3_init,acc3,T,t,time_stop);
36
37 %% Measures
38 c = 3e8;
39 t12 = (1/c)*sqrt((x11 - x22).^2 + (y11 - y22).^2);
```

```
40 t13 = (1/c)*sqrt((x11 - x33).^2 + (y11 - y33).^2);
41 t23 = (1/c)*sqrt((x22 - x33).^2 + (y22 - y33).^2);
42 t_true = [t12;t13;t23];
43
44 % Distances
45 d12 = t12*c;
46 d13 = t13*c;
47 d23 = t23*c;
48 d = [d12;d13;d23];
49
50 % Scheduling matrices each node
51 H1 = [2 0 0; -1 1 1; 1 -1 1];
52 H2 = [0 0 2; 2 0 0; -1 1 1];
53 H3 = [1 -1 1; 0 0 1; 1 1 -1];
54
55 % Delay vectors at each node
56 processing_time = 1e-6;
57 D1 = [1; 1; 1];
58 D2 = [2; 2; 1];
59 D3 = [1; 2; 1];
60 processing_delay1 = processing_time.*D1;
61 processing_delay2 = processing_time.*D2;
62 processing_delay3 = processing_time.*D3;
63
64 % Clean measurements
65 z1 = H1*[t12;t13;t23] + repmat(processing_delay1,1,length(t));
66 z2 = H2*[t12;t13;t23] + repmat(processing_delay2,1,length(t));
67 z3 = H3*[t12;t13;t23] + repmat(processing_delay3,1,length(t));
68
69 %% Filter parameters
70
71 % Parameters
72 M = 3;   %State size [d12; d13; d23]
73 N = 3;   %Measurements [T1; T2; T3]
74 L = 3;   %Noise
75
76 % Measurement matrices
77 H1 = [2 0 0; -1 1 1; 1 -1 1];
78 H2 = [0 0 2; 2 0 0; -1 1 1];
79 H3 = [1 -1 1; 0 0 1; 1 1 -1];
80
```

```
81  % State transition matrix
82  F = eye(M);
83
84  % Process noise generation
85  std_m = 1;
86  m = std_m*randn(L,length(t));
87  cov_m = generate_covariance_matrix(m(1,:),L);
88
89  %% Noisy measures
90  sigma_0 = 1e-7;
91  k_sigma = 1;
92  for k=1:length(z1)
93      r(:,k) = (sigma_0 .* exp((k_sigma .* t_true(:,k))/2)
94      .* 10^-9).*3e8 ;
95      zn1(:,k) = z1(:,k) + r(:,k) .* randn(N,1);
96      zn2(:,k) = z2(:,k) + r(:,k) .* randn(N,1);
97      zn3(:,k) = z3(:,k) + r(:,k) .* randn(N,1);
98  end
99
100 % Noisy measures in a row vector
101 zn1_v = [];
102 zn2_v = [];
103 zn3_v = [];
104 for i = 1:size(zn1,2)
105     zn1_v = [zn1_v, zn1(:,i)'];
106     zn2_v = [zn2_v, zn2(:,i)'];
107     zn3_v = [zn3_v, zn3(:,i)'];
108 end
109
110 %% Filter initialization
111
112 % State vector
113 x1(:,1) = [0;0;0];
114 P1 = eye(M);
115 x2(:,1) = [0;0;0];
116 P2 = eye(M);
117 x3(:,1) = [0;0;0];
118 P3 = eye(M);
119
120 %% Filter: Node 1
121   t_v = 0:T/M:total_time-T/M;
```

```
122    x1_hat = zeros(M,2);
123    actual_pos = [x11(1) y11(1); x22(1) y22(1); x33(1) y33(1)];
124    for i = 1:length(zn1_v)-1
125        i;
126        %Corresponding vector in the measurement matrix
127        H1h = H1(mod(i,3)+1,:);
128
129        %Measurement noise covariance matrix
130        t_approx = H1*x1(:,i);
131        sigma_w_r_approx(:,i) =
132        = (sigma_0 .* exp((k_sigma .* t_approx)/2) .* 10^-9).*3e8;
133        cov_n = diag(sigma_w_r_approx(:,i).^2);
134
135        % Filtering  %
136
137        %Time update
138        x1_plus(:,i) = F*x1(:,i);
139        P1_plus = F*P1*F' + cov_m;
140        %Measurement update
141        K = F*P1_plus*H1'/(H1*P1_plus*H1' + cov_n);
142        inn1(i) = zn1_v(i+1) - H1h*x1(:,i) - processing_delay1(mod(i,3)+1);
143        x1(:,i+1) = x1_plus(:,i) + K(:,mod(i,3)+1)*inn1(i);
144        P1 = (eye(M) - K*H1)*P1_plus + cov_m;
145
146        % %
147        % Position from estimated ranges
148        index = ceil(i/M);
149        actual_pos = [x11(index) y11(index); x22(index)
150        y22(index); x33(index) y33(index)];
151
152        ranges = squareform(abs(c.*x1(:,end)));
153        x1_hat = cmdscale(ranges);
154        [dp,x1_transf,tr] = procrustes(actual_pos,x1_hat);
155        error = abs(x1_transf-actual_pos);
156        error_x(:,i) = error(:,1);
157        x(:,i) = x1_transf(:,1);
158        y(:,i) = x1_transf(:,2);
159        actual_pos = [x(:,i) y(:,i)];
160    end
161    d1 = x1.*c;
162 end
```