Laboratorio 2: Solución de sistemas lineales en MATLAB

Sebastián Valencia Calderón
201111578

1 Introducción

Los sistemas de ecuaciones lineales aparecen en una gran variedad de aplicaciones en ciencia e ingeniería. La naturaleza algorítmica de las soluciones, da lugar a estudiar su eficiencia, su precisión y la representación de los algoritmos y los datos alrededor de ellas. Existen dos métodos para la soluciones de los sistemas de ecuaciones lineales, los iterativos y los no iterativos o directos. Ambos métodos, involucran pasos que pueden ser intensivos en el uso de recursos computacionales mas aun, si se trata de un sistema de grandes dimensiones. Los problemas fundamentales que pueden estudiarse están principalmente relacionados en las siguientes categorías:

1. Cuántas operaciones aritméticas son necesarias para aplicar un método propuesto.
2. Cuál va a ser la precisión o exactitud de la solución hallada por el método propuesto.
3. Cómo puede estimarse la precisión de los cálculos realizados.

Sobre estas tres categorías, se plantean los principales problemas de la computación científica en relación con los problemas de solución de sistemas de ecuaciones lineales. Este tipo de problemas, se presentan de la siguiente manera: dado un sistema de la forma $A \times x = b$, donde A es una matriz en $\mathbb{R}^{n \times m}$, y b es un vector en $\mathbb{R}^n$, se desea hallar un vector en $\mathbb{R}^m$ que satisfaga la igualdad. De manera más formal, se busca responder a la pregunta de si el vector b puede expresarse como un combinación lineal de los vectores columnas de la matriz.

En este caso en particular, se estudia la solución, eficiencia, y diseño de los procedimientos dispuestos en MATLAB para la solución de sistemas lineales. De igual forma, se explora la representación de las matrices y las características particulares que puedan mejorar la eficiencia de los procedimientos. A través de los experimentos realizados, se pretende cumplir los siguientes objetivos.

- Hacer uso y comprender la importancia de algunos métodos iterativos y no iterativos de solución de sistemas lineales disponibles en MATLAB.
- Distinguir algunos sistemas lineales que por su estructura conviene resolverlos mediante métodos de solución de sistemas lineales específicos.
- Identificar algunos casos que implican el mal condicionamiento de una matriz para ser resueltos mediante métodos iterativos o, por el contrario, por métodos no iterativos.

2 Procedimiento

3 Resultados

3.1 Métodos iterativos

1. A continuación, se incluye una descripción de la función de MATLAB `linsolve()`, y unos breves ejemplos de su funcionamiento. La función `linsolve()`, la cual tiene dos parámetros `(A, B)`, tiene como objetivo solucionar el sistema de ecuaciones lineales de la forma $A \times x = b$. Es decir, en caso de que el sistema tenga solución, la función retorna el vector x que satisface la igualdad mencionada. MATLAB, hace uso de optimizaciones dinámicas según las características de la matriz A. Si $A \in \mathbb{R}^{n \times n}$, entonces se usa la factorización LU de la matriz A. De otra forma, la factorizacion QR de la matriz A. Como restricciones, se debe cumplir que en número de filas de A debe ser igual al número de filas de b. Si $A \in \mathbb{R}^{n \times m} \wedge b \in \mathbb{R}^m \Rightarrow x \in \mathbb{R}^n$. En caso de que el problema no esté bien definido, MATLAB, lanza una advertencia. La advertencia, se puede omitir asignando `linsolve(A, B)` a un vector de dos variables, donde la primera posición corresponde a la solución del problema y la segunda al recíproco del número de condición de A de ser cuadrada, de lo contrario, el rango de A (`[X, R] = linsolve(A, b)`).

   A manera de ejemplo, se quiere resolver el sistema lineal mostrado a continuación. De manera algebraica, es posible demostrar que el sistema tiene solución y de hecho es $[2 \ -3 \ 4]$. MATLAB, es capaz de llegar a la misma respuesta haciendo uso de la función estudiada. Cabe resaltar la pertinencia del uso de computadores con sistemas lineales de gran dimensión.

$$\begin{pmatrix} 1 & 2 & 2 \\ 1 & 3 & 3 \\ 2 & 6 & 5 \end{pmatrix} \hat{x} = \begin{pmatrix} 4 \\ 5 \\ 6 \end{pmatrix}$$

   Ahora, se quiere resolver el sistema lineal mostrado a continuación. De manera algebraica, es posible demostrar que el sistema tiene infinitas soluciones. Esto, por la teoría general del álgebra lineal. Para llegar a la misma conclusión a través de MATLAB, es necesario hacer uso de la función actualmente estudiada.

$$\begin{pmatrix} 1 & 2 & 0 & 1 \\ 1 & 1 & 1 & -1 \\ 3 & 1 & 5 & -7 \end{pmatrix} \hat{x} = \begin{pmatrix} 7 \\ 3 \\ 1 \end{pmatrix}$$

   Al intentar calcular este valor, sin hacer uso de la asignación de este a un vector de dos posiciones, MATLAB, indica la advertencia anteriormente discutida. La advertencia, dice lo siguiente: `Warning: Rank deficient, rank = 2, tol = 6.34e-15`. Sin embargo, al realizar la asignación pertinente `[X, R] = linsolve(A, b)`, X toma el valor de $[0 \ 3.33 \ 0 \ 0.33]$, y R, toma el valor de 2, es decir, el rango de la matriz. Si se hacen las verificaciones, se observa que la respuesta del sistema es una aproximación muy buena a la solución numérica más no algebraica del problema. De igual manera ocurre si no se tienen soluciones. En el script 1, se detalla la implementacion en MATLAB de los problemas propuestos para ilustrar el uso básico de

la función `linsolve()`.

2. El parámetro opcional a la función anteriormente estudiada OPTS, proporciona la facilidad de hacerle saber a MATLAB, las características particulares de la matriz A. Es decir, dado que existe un grupo de matrices particulares para las cuales existen algoritmos de factorización o solución directa de los sistemas, es posible optimizar el proceso al contarle a MATLAB estas características.

3. Las posibles características que permiten tales optimizaciones internas, descritas en el literal anterior, se describen a continuación.

   - **UT** Matriz triangular superior. Una matriz es triangular superior si todos los elementos bajo su diagonal son iguales a cero.

   $$A = \begin{pmatrix} \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & 0 & & \times & \times \\ & & & & \times \end{pmatrix} \Rightarrow \forall\ (i,\ j)\ ;\ i > j\ ;\ A_{i,j} = 0$$

     La solución un sistema de ecuaciones de las cuales las ecuaciones del lado izquierdo puedan representarse como matrices de esta forma, es mediante sustitución iterativa. Es decir, primero, se halla x_n, ya que el último pivote (el cual debe ser distinto de cero, por la suposición de que se tiene una solución), lleva de manera sencilla a encontrar este valor. Una vez se cuente con este valor, es sencillo encontrar x_{n-1}, y así de manera sucesiva hacia atrás.
   - **LT** Matriz triangular inferior

   $$A = \begin{pmatrix} \times & & & & \\ \times & \times & & 0 & \\ \times & \times & \times & & \\ \times & \times & \times & \times & \\ \times & \times & \times & \times & \times \end{pmatrix} \Rightarrow \forall\ (i,\ j)\ ;\ i < j\ ;\ A_{i,j} = 0$$

   - **SYM** Matriz simétrica real o compleja conjugada

   $$A \in \mathbb{R}^{n \times m}\ ;\ A^T = A$$

   - **RECT** Matriz general rectangular

   $$A \in \mathbb{R}^{n \times m}\ ;\ n \neq m$$

   - **POSDEF** Matriz positiva definida

   $$A \in \mathbb{R}^{n \times n}\ ;\ \forall\ v \in \mathbb{R}^n\ ;\ v \neq 0\ v^T A v > 0$$

   - **UHESS** Matriz superior de Hessenberg

   $$\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1n} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & x_{n,n-1} & \dots & x_{nn} \end{bmatrix}$$

- **TRANSA** Matriz conjugada transpuesta, se refiere a si la función`linsolve()`, resuelve el sistema para la matriz transpuesta de la dada por argumento.

4. Para generar las matrices aleatorias, se usa el script 2, el cual, recibe las dimensiones de la matriz deseada y el tipo deseado, el último parámetro, puede ser de los siguientes tipos, los cuales representan lo mismo que en el literal anterior. `UT, LT, SYM, RECT, POSDEF, UHESS, TRANSA`. Para generar matrices del tipo `UT`, se hace uso de la función de `MATLAB triu`, al igual que `randi`. Para la generación de matrices tip `LT`, se usa la función `tril`. Para generar matrices del tipo `SYM`, se suman dos matrices haciendo uso de `triu`, las matrices rectangulares se generan directamente con la función `randi`. Para la matriz superior de Hessenberg, se hace uso de la función `hess`.

   Para la matriz de tipo `TRANSA`, se usó la función `complex` y `conj`. Por último, la matriz de tipo `POSDEF`, se genera a partir de una matriz diagonal estrictamente dominante, para las cuales se cumple que $|a_{ii}| > \sum_{j \neq i} |a_{ij}| \; \forall \; i, \; j = 1 \; \ldots n$. Se sabe que estas matrices son positivas definidas. Un procedimiento para generar este tipo de matrices, se encuentra en [1], página 346, se adapta este algoritmo, y se genera de manera consistente cualquiera de estos tipos de algoritmos. Además de devolver la matriz adecuada según el tipo, el script 2 retorna un vector b consistente con las dimensiones. A continuación se incluye un ejemplo de la ejecución de el script 2, para cada tipo de matriz.

   - **UT** `[mat, b] = randmatrix(4, 'UT')`

$$mat = \begin{pmatrix} 7 & 4 & 3 & 2 \\ 0 & 6 & 10 & 3 \\ 0 & 0 & 5 & 1 \\ 0 & 0 & 0 & 7 \end{pmatrix} \; ; \; b = \begin{pmatrix} 4 \\ 8 \\ 5 \\ 1 \end{pmatrix}$$

   - **LT** `[mat, b] = randmatrix(4, 'LT')`

$$mat = \begin{pmatrix} 6 & 0 & 0 & 0 \\ 8 & 6 & 0 & 0 \\ 3 & 5 & 9 & 0 \\ 2 & 7 & 5 & 3 \end{pmatrix} \; ; \; b = \begin{pmatrix} 7 \\ 7 \\ 7 \\ 10 \end{pmatrix}$$

   - **SYM** `[mat, b] = randmatrix(4, 'SYM')`

$$mat = \begin{pmatrix} 5 & 6 & 3 & 5 \\ 6 & 8 & 2 & 5 \\ 3 & 2 & 2 & 4 \\ 5 & 5 & 4 & 8 \end{pmatrix} \; ; \; b = \begin{pmatrix} 4 \\ 2 \\ 10 \\ 7 \end{pmatrix}$$

     Un test para esta matriz, es `issymmetric(mat)`.

- **RECT** `[mat, b] = randmatrix(4, 'RECT')`

$$mat = \begin{pmatrix} 35 & 24 & 50 & 26 & 7 & 4 \\ 96 & 58 & 124 & 63 & 8 & 5 \\ 95 & 70 & 135 & 71 & 5 & 3 \\ 65 & 38 & 85 & 43 & 1 & 2 \end{pmatrix} \; ; \; b = \begin{pmatrix} 7 \\ 8 \\ 10 \\ 10 \end{pmatrix}$$

  El rango de esta matriz, siempre es el tamaño dado por parámetro al script 2. Es decir, la matriz es de rango completo.

- **POSDEF** `[mat, b] = randmatrix(4, 'POSDEF')`

$$mat = \begin{pmatrix} 64 & 4 & 4 & 8 \\ 4 & 52 & 7 & 2 \\ 9 & 8 & 104 & 9 \\ 8 & 10 & 9 & 108 \end{pmatrix} \; ; \; b = \begin{pmatrix} 9 \\ 5 \\ 9 \\ 4 \end{pmatrix}$$

  Un test para esta matriz, es `positivedefinite = all(eig(mat) > 0)`.

- **UHESS** `[mat, b] = randmatrix(4, 'UHESS')`

$$mat = \begin{pmatrix} 2 & -5 & -4 & -4 \\ -13 & 11 & 5 & 0 \\ 0 & 5 & 1 & 4 \\ 0 & 0 & -3 & 2 \end{pmatrix} \; ; \; b = \begin{pmatrix} 6 \\ 9 \\ 6 \\ 2 \end{pmatrix}$$

- **TRANSA** `[mat, b] = randmatrix(4, 'TRANSA')`

$$mat = \begin{pmatrix} 6+8\,\mathrm{i} & 9+6\,\mathrm{i} & 9+3\,\mathrm{i} & 10+7\,\mathrm{i} \\ 1+\mathrm{i} & 9+7\,\mathrm{i} & 7+7\,\mathrm{i} & 10+8\,\mathrm{i} \\ 6+9\,\mathrm{i} & 5+10\,\mathrm{i} & 9+8\,\mathrm{i} & 3+6\,\mathrm{i} \\ 5+10\,\mathrm{i} & 10+6\,\mathrm{i} & 6+\mathrm{i} & 9+2\,\mathrm{i} \end{pmatrix} \; ; \; b = \begin{pmatrix} 9 \\ 6 \\ 10 \\ 7 \end{pmatrix}$$

5. El script para crear las matrices, solucionar los sistemas usando `linsolve()`, y la medición de los tiempos, se incluye en el srcipt 3. Este script, arroja una matriz de medición para los sistemas con cada una de las opciones y una matriz de error con la misma configuración.

   Como ejemplo, correr el comando `[M, E] = linmeasurement(5)`, arroja los siguientes resultados para la matriz de tiempos:

	UT	**LT**	**SYM**	**RECT**	**POSDEF**	**UHESS**	**TRANSA**
UT	0.005370081	0.000013210	0.002794560	0.000086133	-1.000000000	0.000021848	0.000070284
LT	0.000016979	0.000005994	0.000026746	0.000016543	-1.000000000	0.000012827	0.000015403
SYM	0.000007155	0.000005576	0.000015837	0.000012767	-1.000000000	0.000007909	0.000014417
RECT	0.000007576	0.000005856	-1.000000000	0.000016461	-1.000000000	-1.000000000	-1.000000000
POSDEF	-1.000000000	-1.000000000	0.000026553	-1.000000000	0.000111140	-1.000000000	0.000016571
UHESS	0.000007460	0.000005317	0.000015736	0.000013214	-1.000000000	0.000008142	0.000014918
TRANSA	0.000012907	0.000007945	0.000095468	0.000065643	-1.000000000	0.000013785	0.000100514

   Ahora, se corre 10 veces el mismo experimento y se promedian los tiempos, los resultados se muestran a continuación:

	UT	**LT**	**SYM**	**RECT**	**POSDEF**	**UHESS**	**TRANSA**
UT	0.000163	6.68e-05	0.00444	0.0126	-1	0.000371	0.00266
LT	0.000149	7.35e-05	0.00669	0.0121	-1	0.000629	0.00248
SYM	0.000132	6.5e-05	0.00634	0.0123	-1	0.000634	0.00256
RECT	0.0002	7.46e-05	-1	0.0221	-1	-1	-1
POSDEF	-1	-1	0.00467	-1	0.00104	-1	0.00462
UHESS	8.94e-05	6.18e-05	0.00548	0.0133	-1	0.000682	0.00248
TRANSA	0.000357	0.000383	0.0162	0.0413	-1	0.00193	0.00799

Figura 1: Se muestra los tiempos requeridos para cada tipo de matriz y cada tipo de opción posible. Los menos uno, muestran los caso infactibles según las configuraciones de linsolve().

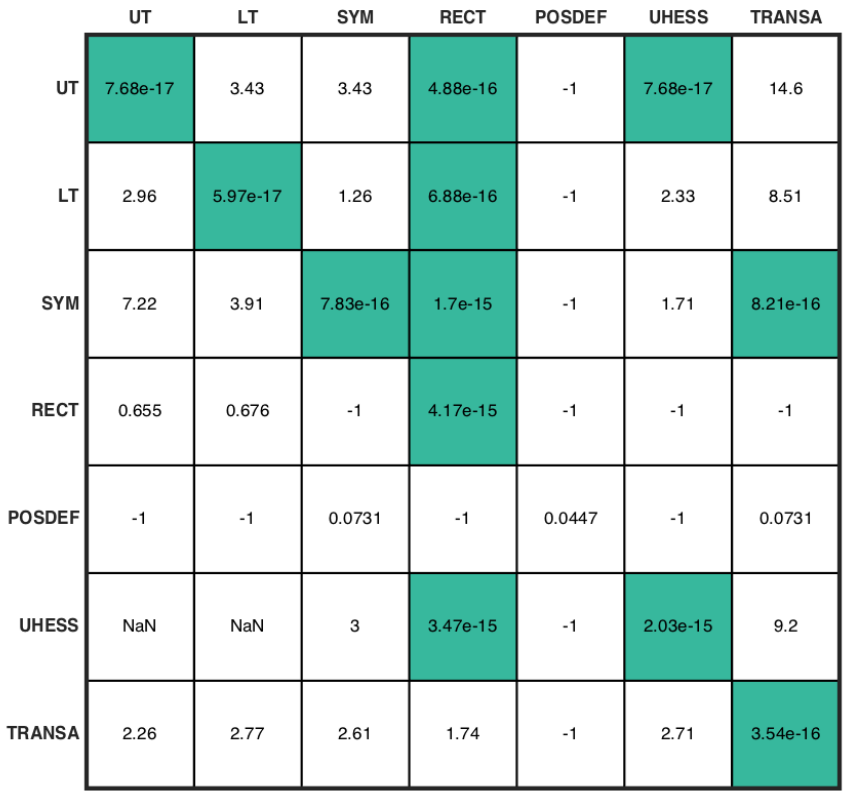
El proceso entero, se ejecuta mediante el script 4.

6. Los tiempos de ejecución del algoritmo, son buenos en cada caso, sin embargo, no son concluyentes para evaluar la ejecución del algoritmo y las pertinencia de cada una de las posibles opciones. Es mejor confiar en el error de cálculo de cada uno de los métodos. Es decir, el error de calcular la solución de un sistema lineal con una configuración de `opts` dada, esto para cada matriz y para cada configuración.

	UT	LT	SYM	RECT	POSDEF	UHESS	TRANSA
UT	1.32e+55	415	415	0.0609	-1	1.32e+55	3.78e+71
LT	409	5.9e+51	20.3	0.0721	-1	1.89e+17	4.01e+16
SYM	1.65e+72	2.99e+71	1.74e-12	4.95e-13	-1	5.66e+39	1.64e-12
RECT	0.643	0.795	-1	5.6e-12	-1	-1	-1
POSDEF	-1	-1	0.00491	-1	0.00498	-1	0.00491
UHESS	NaN	NaN	NaN	5.59e-13	-1	1.55e-13	462
TRANSA	5.66e+50	2.8e+49	23.9	12.1	-1	2.73e+29	1.06e-13

`Figura 2: Se muestra los errores de calculo para cada tipo de matriz y cada tipo de opción posible. Los valores resaltados, muestran los valores menores a una tolerancia dada, en este caso de` 3×10^{-5}.

Idealmente, solucionar una matriz de un tipo τ, con la opción prendida para el mismo tipo τ, debería arrojar los mejores resultados posibles. Es decir, es de fundamental importancia contar con errores bajos, sobre todo en la diagonal de la matriz de la anterior figura. Sin embargo, este no es el caso, por lo que se realiza un experimento para un tamaño menor de la matriz. El resultado, se evidencia en la figura siguiente. Como puede verse, se cumple el caso ideal, excepto para las matrices positivas definidas. Lo que puede resultar de esto, es que existe una perturbación demasiado considerable en las funciones de sustitución en `MATLAB`, este resultado, viene del hecho de que resolver matrices triangulares, trae un error mucho mayor a los otros tipos de matrices. Como puede verse en la figura anterior, resolver una matriz triangular con la opción `RECT`, es mucho mejor en términos de error que las opciones teóricamente ideales. Es fundamental resaltar que el tiempo ni es determinante sino el error de cálculo.



	UT	LT	SYM	RECT	POSDEF	UHESS	TRANSA
UT	7.68e-17	3.43	3.43	4.88e-16	-1	7.68e-17	14.6
LT	2.96	5.97e-17	1.26	6.88e-16	-1	2.33	8.51
SYM	7.22	3.91	7.83e-16	1.7e-15	-1	1.71	8.21e-16
RECT	0.655	0.676	-1	4.17e-15	-1	-1	-1
POSDEF	-1	-1	0.0731	-1	0.0447	-1	0.0731
UHESS	NaN	NaN	3	3.47e-15	-1	2.03e-15	9.2
TRANSA	2.26	2.77	2.61	1.74	-1	2.71	3.54e-16

Figura 3: Se muestra los errores de calculo para cada tipo de matriz y cada tipo de opción posible. Los valores resaltados, muestran los valores menores a una tolerancia dada, en este caso de 3×10^{-5}. El tamaño de la matriz es mucho menor al anterior.

3.2 Métodos no iterativos

1. La función `lsqr`, es una implementación en `MATLAB`, para los gradientes conjugados de ecuaciones normales. Esta función, intenta minimizar $\|b - Ax\|$ para x, si A es consistente, de tal manera que se resuelve el sistema de ecuaciones lineales $Ax = b$, de manera más específica, intenta solucionar mínimos cuadrados para la norma del residuo. Éste método iterativo, funciona mejor si A es una matriz dispersa. La implementación, diseño y análisis de este método, se encuentra en la referencia [2]

   $$x; Ax = b \ ; \ \min \|b - Ax\|$$

   Por otro lado, la función `symmlq`, intenta solucionar el sistema de ecuaciones lineales de la forma $Ax = b$, donde A es una matriz simétrica cuadrada, no necesariamente positiva definida, esta debe ser de gran tamaño y dispersa.

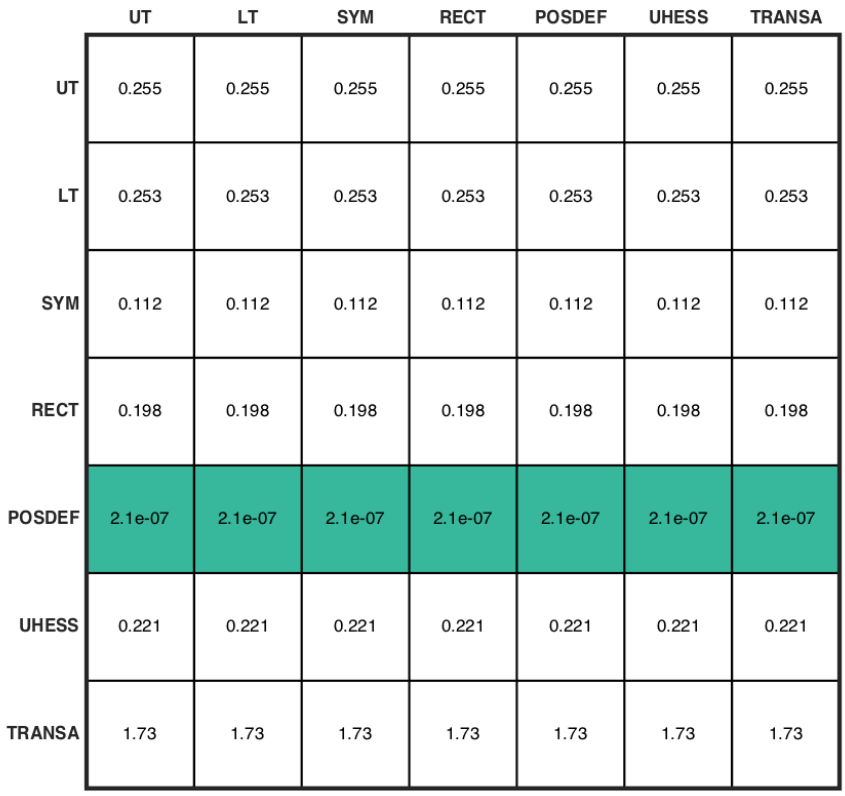
2. Para la función `lsqr`, es deseable pero no requerido, una matriz dispersa. Para la función `symmlq`, es necesario una matriz simétrica.

3. Las matrices en el primer caso, pueden ser las anteriores, para el segundo caso, únicamente simétricas.

4. En le primer caso, se corre el script 5, el cual hace uso del script 6, para solucionar los sistemas de ecuaciones lineales. El único tipo de matriz solucionada por este función es positiva definida, pues se corrió de manera análoga al caso no iterativo, y ninguna matriz converge, excepto las positivas definidas, con un tiempo cercano a 0.00034, con un error de 2.2×10^{-7}. A continuación, se muestran los tiempos.

	UT	LT	SYM	RECT	POSDEF	UHESS	TRANSA
UT	0.0075	0.00885	0.00697	0.00875	0.00697	0.00746	0.00928
LT	0.00913	0.00667	0.0084	0.00735	0.00777	0.00798	0.00885
SYM	0.00841	0.0065	0.0077	0.00631	0.00793	0.00646	0.0073
RECT	0.00435	0.00657	0.00705	0.0103	0.0084	0.0107	0.00788
POSDEF	0.00373	0.00316	0.00313	0.00309	0.00265	0.00356	0.00284
UHESS	0.00379	0.00691	0.00545	0.0115	0.00587	0.00664	0.00915
TRANSA	0.0233	0.0224	0.0217	0.0215	0.0238	0.024	0.0239

Figura 4: Se muestra los tiempos de cálculo para cada tipo de matriz haciendo uso de lsqr, aqui, se debe tener en cuenta que el tiempo incluye los valores independientemente de si la solución converge.



	UT	LT	SYM	RECT	POSDEF	UHESS	TRANSA
UT	0.255	0.255	0.255	0.255	0.255	0.255	0.255
LT	0.253	0.253	0.253	0.253	0.253	0.253	0.253
SYM	0.112	0.112	0.112	0.112	0.112	0.112	0.112
RECT	0.198	0.198	0.198	0.198	0.198	0.198	0.198
POSDEF	2.1e-07	2.1e-07	2.1e-07	2.1e-07	2.1e-07	2.1e-07	2.1e-07
UHESS	0.221	0.221	0.221	0.221	0.221	0.221	0.221
TRANSA	1.73	1.73	1.73	1.73	1.73	1.73	1.73

Figura 5: Se muestra los errores para cada tipo de matriz haciendo uso de lsqr, aqui, la solución sólo converge si la matriz es positiva definida, es decir, como se ve en los valores resaltados. La respuesta de MATLAB, es lsqr converged at iteration 6 to a solution with relative residual 2.2e-07.

En el caso de la función `symmlq`, donde la matriz debe ser simétrica, se realiza un experimento análogo haciendo uso de los scripts 7 y 8. Los resultados, se muestran en las figuras 6 y 7.

Los valores únicamente convergen si las matices son o positivas definidas y en el segundo caso positivas definidas o simétricas, el tiempo de ejecución, es un poco mayor a los hallados con `linsolve`, lo que da lugar a pensar que en algunos casos, es mejor usar el mejor (el que minimice mejor error residual de la solución) método para resolver estos sistemas. De tal manera, se usarían los últimos dos métodos para resolver matrices positivas definidas, y el primero para las matrices simétricas, rectangulares, de Hessenberg y el caso traspuesto. El problema persistente, es el caso de las matrices triangulares, esto ultimo puede deberse a que el error se extiende por las divisiones sucesivas entre resultados anteriores, es decir la perturbación se debe al tamaño de los problemas.

	UT	LT	SYM	RECT	POSDEF	UHESS	TRANSA
UT	0.00893	0.00556	0.00863	0.00747	0.00646	0.00677	0.008
LT	0.00725	0.0068	0.00658	0.00683	0.00773	0.00642	0.00642
SYM	0.00794	0.00684	0.00732	0.00773	0.00648	0.00683	0.00669
RECT	-1	-1	-1	-1	-1	-1	-1
POSDEF	0.00146	0.00197	0.00213	0.00152	0.00146	0.00165	0.00216
UHESS	0.00386	0.0053	0.00364	0.00634	0.00899	0.0061	0.00558
TRANSA	0.018	0.018	0.0183	0.0216	0.0223	0.0217	0.0233

Figura 6: Se muestra los tiempos de cálculo para cada tipo de matriz haciendo uso de symmlq, aqui, se debe tener en cuenta que el tiempo incluye los valores independientemente de si la solución converge. Los valores resultados, son infactibles.

4 Conclusiones

Por medio del laboratorio, se pudo explorar el uso de las herramientas provistas por MATLAB para la solución de sistemas de ecuaciones lineales. Primero, un método que resuelve el sistema hallando la matriz inversa de A, para hallar la solución al sistema, dependiendo del tipo de la matriz, MATLAB realiza optimizaciones pertinentes para resolver el sistema de ecuaciones lineales usando las propiedades intrínsecas de la matriz A. Por otra parte, se hace uso de LSQR para resolver el mismo tipo de ecuaciones sobre matrices positivas definidas o simétricas. A través de los distintos experimentos, se pudo comprobar la perturbación total de la solución de cada incógnita y como esto afecta la solución global del sistema. Ademas, se comprobó que el tiempo no es determinante para soluciones numéricas, es mas importante contemplar el error y no el tiempo de ejecución. El tiempo puede considerarse, una vez se tenga certeza de una solución sin tanto error numérico.

- Tener conciencia de la estructura, forma, características y propiedades de una matriz, es conveniente para determinar el mejor método para solucionar un sistema de ecuaciones lineales. La palabra mejor, se debe entender en el contexto de minimizar el error, de forma que el tiempo de computación y los recursos sean eficientes. Sin

	UT	LT	SYM	RECT	POSDEF	UHESS	TRANSA
UT	0.754	0.754	0.754	0.754	0.754	0.754	0.754
LT	0.761	0.761	0.761	0.761	0.761	0.761	0.761
SYM	0.424	0.424	0.424	0.424	0.424	0.424	0.424
RECT	-1	-1	-1	-1	-1	-1	-1
POSDEF	5.91e-07	5.91e-07	5.91e-07	5.91e-07	5.91e-07	5.91e-07	5.91e-07
UHESS	0.93	0.93	0.93	0.93	0.93	0.93	0.93
TRANSA	2.39	2.39	2.39	2.39	2.39	2.39	2.39

Figura 7: Se muestra los errores para cada tipo de matriz haciendo uso de symmlq, aqui, la solución sólo converge si la matriz es positiva definida o simétrica, es decir, como se ve en los valores resaltados. La respuesta de MATLAB, es lsqr converged at iteration 6 to a solution with relative residual 5.91e-07.

embargo, conviene concentrar el estudio en el error obtenido y no sobre el tiempo computacional invertido en solucionar el sistema.

- En experimentos posteriores, se puede incluir el estudio del acondicionamiento de las matrices y por lo tanto de las soluciones. Es necesario plantear una función general que estudie las propiedades de las matrices, y su condición para resolver de la mejor manera los sistemas lineales.

5 Scripts

Script 1: Uso de la funcion linsolve(), para los tres casos mas estudiados al resolver ecuaciones lineales. (linearsystems.m)

```
% Three equations, one solution
A = [ 1 2 2 ; 1 3 3 ; 2 6 5 ];
b = [ 4 ; 5 ; 6 ];
x1 = linsolve(A, b);

latex_table1 = latex(sym(A));
latex_vector1 = latex(sym(b));

% Three equations, infnitely many solutions
A = [ 1 2 0 1 ; 1 1 1 -1 ; 3 1 5 -7];
b = [ 7 ; 3 ; 1 ];
x2 = linsolve(A, b);
[ans1, R1] = linsolve(A, b);

latex_table2 = latex(sym(A));
latex_vector2 = latex(sym(b));

% Three equations, no solution
A = [ 1 1 1 ; 3 2 1 ; 1 -2 -5 ];
b = [ 3 ; 3 ; 1 ];
x3 = linsolve(A, b);
[ans2, R2] = linsolve(A, b);
```

Script 2: Generación dinámica de matrices aleatorias del tipo triangular, simétrica, rectangular, positiva definida, superior de Hessenberg o una matriz compleja (randmatrix.m)

```
function [matrix, b] = randmatrix(size, kind)

    MIN = 1;
    MAX = 10;

    b = randi([MIN MAX], size, 1);

    switch(kind)
        case 'UT'
            UT = triu(randi([MIN MAX], size),1);
            for i=1:size
                UT(i, i) = randi([MIN, MAX]);
            end;
            matrix = UT;
        case 'LT'
```

```
16              LT = tril(randi([MIN MAX], size),-1);
17              for i=1:size
18                  LT(i, i) = randi([MIN, MAX]);
19              end;
20              matrix = LT;
21          case 'SYM'
22              seed = randi([MIN MAX], size);
23              SYM = triu(seed) + triu(seed, 1)';
24              matrix = SYM;
25          case 'RECT'
26              RECT = zeros(size);
27              for i = 1:size-1
28                  RECT = RECT + randi([MIN MAX], size, 1) * randi([MIN MAX],
                        1, size);
29              end
30              B = randi([MIN MAX], size, ceil(size / 2));
31              RECT = [RECT B];
32              matrix = RECT;
33          case 'POSDEF'
34              POSDEF= randi([MIN MAX], size);
35              for i=1:size
36                  POSDEF(i, i) = 0;
37                  POSDEF(i, i) = sum(abs(POSDEF(i,:))) * 4;
38              end;
39              matrix = POSDEF;
40          case 'UHESS'
41              seed = randi([MIN MAX], size, size);
42              UHESS = ceil(hess(seed));
43              matrix = UHESS;
44          case 'TRANSA'
45              seed = complex(randi([MIN MAX], size, size), randi([MIN MAX],
                    size, size));
46              TRANSA = conj(seed)';
47              matrix = TRANSA;
48      end;
49  end
```

Script 3: Experimento sobre distintos tipos de matrices y condiciones para el metodo linsolve (linmeasurement.m)

```
function [measurements, err] = linmeasurement(dimension)

    MATRICES = {'UT', 'LT', 'SYM', 'RECT', 'POSDEF', 'UHESS', 'TRANSA'};
    n = size(MATRICES);
    n = n(2);

    measurements = zeros(n);
    err = zeros(n);

    for j=1:n
        opts.(MATRICES{j}) = false;
    end;

    macrocounter = 1;
    for k=1:n
        kind = MATRICES{k};
        [matrix, b] = randmatrix(dimension, kind);

        ms = zeros(1, 7);
        es = zeros(1, 7);

        counter = 1;
        for i=1:n
            current = MATRICES{i};
            for j=1:n
                if(strcmp(MATRICES{j}, current) == 0)
                    opts.(MATRICES{j}) = false;
                else
                    opts.(MATRICES{j}) = true;
                end;
            end;

            if(strcmp('POSDEF', kind) == 1)
                opts.('SYM') = true;
            end;

            try
                start = tic;
                [x, r] = linsolve(matrix, b, opts);
                elapsed = toc(start);

                if(strcmp('TRANSA', kind) == 1)
                    residual = b - matrix'*x;
                else
                    residual = b - matrix*x;
```

```
                end;

                es(counter) = norm(residual) / norm(b);
                ms(counter) = elapsed;
            catch
                es(counter) = -1;
                ms(counter) = -1;
            end;

            counter = counter + 1;
        end;

        measurements(macrocounter, :) = ms;
        err(macrocounter, :) = es;

        macrocounter = macrocounter + 1;

    end;
end
```

Script 4: Experimento completo para el caso de linsolve (runlinsolve.m)

```
TIMES = 10;
dimension = 500;

format long;

m = zeros(7);
e = zeros(7);

for i=1:TIMES
    [measurements, errors] = linmeasurement(dimension);
    m = m + measurements;
    e = e + errors;
end;

m = m ./ TIMES;
e = e ./ TIMES;

plotter(e, parula, 'E');
set(gcf, 'Position', [400 400 700 700]);
saveas(gcf, '../img/plotlinsolve_error.png');

plotter(m, winter, 'M');
set(gcf, 'Position', [400 400 700 700]);
saveas(gcf, '../img/plotlinsolve_times.png');
```

Script 5: Experimentación haciendo uso de LSQR (runlsqr.m)

```
TIMES = 10;
dimension = 500;

format long;

m = zeros(7);
e = zeros(7);

for i=1:TIMES
    [measurements, errors] = lsqrmeasurement(dimension);
    m = m + measurements;
    e = e + errors;
end;

m = m ./ TIMES;
e = e ./ TIMES;

plotter(e, parula, 'E');
set(gcf, 'Position', [400 400 700 700]);
saveas(gcf, '../img/plotlsqr_error.png');

plotter(m, winter, 'M');
set(gcf, 'Position', [400 400 700 700]);
saveas(gcf, '../img/plotlsqr_times.png');
```

Script 6: Experimentación haciendo uso de LSQR (lsqrmeasurement.m)

```
function [measurements, err] = lsqrmeasurement(dimension)

    MATRICES = {'UT', 'LT', 'SYM', 'RECT', 'POSDEF', 'UHESS', 'TRANSA'};
    n = size(MATRICES);
    n = n(2);

    measurements = zeros(n);
    err = zeros(n);

    for j=1:n
        opts.(MATRICES{j}) = false;
    end;

    macrocounter = 1;
    for k=1:n
        kind = MATRICES{k};
        [matrix, b] = randmatrix(dimension, kind);

```

```
19          ms = zeros(1, 7);
20          es = zeros(1, 7);
21
22          counter = 1;
23          for i=1:n
24              current = MATRICES{i};
25              for j=1:n
26                  if(strcmp(MATRICES{j}, current) == 0)
27                      opts.(MATRICES{j}) = false;
28                  else
29                      opts.(MATRICES{j}) = true;
30                  end;
31              end;
32
33              if(strcmp('POSDEF', kind) == 1)
34                  opts.('SYM') = true;
35              end;
36
37              try
38                  start = tic;
39                  x = lsqr(matrix, b);
40                  elapsed = toc(start);
41
42                  if(strcmp('TRANSA', kind) == 1)
43                      residual = b - matrix'*x;
44                  else
45                      residual = b - matrix*x;
46                  end;
47
48                  es(counter) = norm(residual) / norm(b);
49                  ms(counter) = elapsed;
50              catch
51                  es(counter) = -1;
52                  ms(counter) = -1;
53              end;
54
55              counter = counter + 1;
56          end;
57
58          measurements(macrocounter, :) = ms;
59          err(macrocounter, :) = es;
60
61          macrocounter = macrocounter + 1;
62
63      end;
64  end
```

Script 7: Experimentación haciendo uso de LSQR simétrico (runsymmlq.m)

```
TIMES = 10;
dimension = 500;

format long;

m = zeros(7);
e = zeros(7);

for i=1:TIMES
    [measurements, errors] = symmeasurement(dimension);
    m = m + measurements;
    e = e + errors;
end;

m = m ./ TIMES;
e = e ./ TIMES;

plotter(e, parula, 'E');
set(gcf, 'Position', [400 400 700 700]);
saveas(gcf, '../img/plotsymmlq_error.png');

plotter(m, winter, 'M');
set(gcf, 'Position', [400 400 700 700]);
saveas(gcf, '../img/plotsymmlq_times.png');
```

Script 8: Experimentación haciendo uso de LSQR simétrico. (symmeasurement.m)

```
function [measurements, err] = symmeasurement(dimension)

    MATRICES = {'UT', 'LT', 'SYM', 'RECT', 'POSDEF', 'UHESS', 'TRANSA'};
    n = size(MATRICES);
    n = n(2);

    measurements = zeros(n);
    err = zeros(n);

    for j=1:n
        opts.(MATRICES{j}) = false;
    end;

    macrocounter = 1;
    for k=1:n
        kind = MATRICES{k};
        [matrix, b] = randmatrix(dimension, kind);

```

```
19          ms = zeros(1, 7);
20          es = zeros(1, 7);
21
22          counter = 1;
23          for i=1:n
24              current = MATRICES{i};
25              for j=1:n
26                  if(strcmp(MATRICES{j}, current) == 0)
27                      opts.(MATRICES{j}) = false;
28                  else
29                      opts.(MATRICES{j}) = true;
30                  end;
31              end;
32
33              if(strcmp('POSDEF', kind) == 1)
34                  opts.('SYM') = true;
35              end;
36
37              try
38                  start = tic;
39                  x = symmlq(matrix, b);
40                  elapsed = toc(start);
41
42                  if(strcmp('TRANSA', kind) == 1)
43                      residual = b - matrix'*x;
44                  else
45                      residual = b - matrix*x;
46                  end;
47
48                  es(counter) = norm(residual) / norm(b);
49                  ms(counter) = elapsed;
50              catch
51                  es(counter) = -1;
52                  ms(counter) = -1;
53              end;
54
55              counter = counter + 1;
56          end;
57
58          measurements(macrocounter, :) = ms;
59          err(macrocounter, :) = es;
60
61          macrocounter = macrocounter + 1;
62
63      end;
64  end
```

Listings

6 Bibliografía

[1] Greenbaum, A. and Chartier, T.P. *Numerical Methods: Design, Analysis, and Computer Implementation of Algorithms* 2012, Princeton University Press.

[2] Paige, Christopher C. and Saunders, Michael A. *LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares* March 1982, ACM Trans. Math. Softw..