# Eficiencia de métodos para resolver ecuaciones lineales

May 16, 2020

## 1 Eficiencia de los métodos de solución de sistemas de ecuaciones lineales

**Óscar Alfonso Gómez Sepúlveda**

**2198577**

### 1.1 Objetivos

- Plantear código para resolver una matriz calculando la invertida
- Desarrollar código para resolver una matriz por el método LU (Lower Upper)
- Determinar los tiempos computacionales para cada método

Se plantea visualizar el tiempo computacional que le toma a dos diferentes métodos de solución de sistemas de ecuaciones, Inv(A)xb y LU decomposition.

- Se importan las librerías necesarias, numpy para generar la matriz aleatoria y time para medir el tiempo computacional

[1]:
```
%matplotlib ipympl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from time import process_time_ns
from sympy import init_session
init_session(use_latex=True)
```

```
IPython console for SymPy 1.5.1 (Python 3.8.2-64-bit) (ground types: python)

These commands were executed:
>>> from __future__ import division
>>> from sympy import *
>>> x, y, z, t = symbols('x y z t')
>>> k, m, n = symbols('k m n', integer=True)
>>> f, g, h = symbols('f g h', cls=Function)
>>> init_printing()
```

```
Documentation can be found at https://docs.sympy.org/1.5.1/
```

- Dimensiones de la matriz aleatoria

[2]:
```
m = 100
n = 100
```

- Generación de matriz y vector aleatorio

[3]:
```
A = (10*np.random.rand(n,m)) # Matriz aleatoria
C = (10*np.random.rand(n)) # Vector aleatoria
A1 = A

df = pd.DataFrame(data=A)
print("La matriz generada aleatoriamente es:")
df
```

```
La matriz generada aleatoriamente es:
```

[3]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | \ |
|---|---|---|---|---|---|---|---|---|
| 0 | 2.910259 | 1.322554 | 1.640273 | 0.072003 | 9.405811 | 5.946100 | 8.446608 | |
| 1 | 1.150163 | 5.741584 | 7.208298 | 6.780837 | 3.438964 | 5.197996 | 9.116755 | |
| 2 | 4.708047 | 7.194662 | 2.435679 | 7.303733 | 4.209647 | 8.210837 | 0.243499 | |
| 3 | 4.903957 | 5.868672 | 5.213807 | 4.569805 | 4.767403 | 9.888242 | 5.267813 | |
| 4 | 1.958806 | 2.198332 | 2.415528 | 3.280604 | 8.311328 | 2.928046 | 0.948006 | |
| .. | ... | ... | ... | ... | ... | ... | ... | |
| 95 | 9.091845 | 1.289879 | 1.458494 | 7.774994 | 6.167048 | 3.788068 | 8.437959 | |
| 96 | 5.852282 | 9.941728 | 7.014411 | 8.231226 | 0.696434 | 2.149230 | 8.464366 | |
| 97 | 6.459895 | 0.938833 | 8.089958 | 5.033157 | 5.539317 | 6.364256 | 1.928239 | |
| 98 | 8.318162 | 6.530241 | 5.747268 | 9.707862 | 8.630749 | 8.030317 | 7.498042 | |
| 99 | 1.430590 | 6.253750 | 8.227369 | 9.415518 | 9.163295 | 4.899882 | 8.117648 | |

| | 7 | 8 | 9 | ... | 90 | 91 | 92 | 93 | \ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 2.043621 | 4.003542 | 2.494842 | ... | 5.491859 | 3.805953 | 4.207447 | 9.554329 | |
| 1 | 5.234933 | 7.508659 | 0.039932 | ... | 1.434773 | 0.429321 | 2.258641 | 1.586116 | |
| 2 | 6.610345 | 5.191012 | 0.696287 | ... | 6.804701 | 3.582814 | 7.222925 | 4.570475 | |
| 3 | 5.529254 | 6.496744 | 4.326106 | ... | 6.679021 | 2.983546 | 7.587068 | 0.211894 | |
| 4 | 9.699616 | 3.515821 | 9.309631 | ... | 1.129509 | 6.831833 | 4.177964 | 7.483105 | |
| .. | ... | ... | ... | ... | ... | ... | ... | ... | |
| 95 | 3.751799 | 0.057161 | 7.855318 | ... | 0.944306 | 8.649799 | 9.995416 | 4.905038 | |
| 96 | 4.575929 | 1.292159 | 8.921850 | ... | 0.754555 | 9.858006 | 3.136981 | 9.962451 | |
| 97 | 6.099950 | 3.697664 | 5.349407 | ... | 4.591077 | 5.303759 | 5.729768 | 7.459454 | |
| 98 | 4.226950 | 9.298035 | 2.030012 | ... | 3.620964 | 0.813223 | 6.042457 | 0.596753 | |
| 99 | 7.290022 | 9.046675 | 6.423626 | ... | 4.949518 | 5.088333 | 8.656156 | 2.608214 | |

| | 94 | 95 | 96 | 97 | 98 | 99 |
|---|---|---|---|---|---|---|
| 0 | 9.116754 | 1.113869 | 4.814218 | 4.352258 | 0.634359 | 9.774294 |
| 1 | 6.190979 | 5.560568 | 1.395141 | 1.504416 | 1.021260 | 7.324607 |

```
2   2.527507  4.177929  4.346277  0.339070  5.943178  3.842668
3   9.944181  4.608147  1.602135  9.351939  1.535943  3.659853
4   4.860617  2.530177  3.037954  2.051675  6.240132  2.880393
..       ...       ...       ...       ...       ...       ...
95  7.634335  8.717734  2.582176  5.371787  2.707795  7.566661
96  7.007559  4.982764  8.756769  2.343500  1.559684  7.083975
97  3.683323  4.446318  2.499490  9.598245  2.206782  5.839720
98  0.972777  9.360098  7.741887  8.795340  9.723241  8.259018
99  2.166502  0.885177  4.185736  6.394575  0.135606  2.588240

[100 rows x 100 columns]
```

### 1.2 Solución de sistema de ecuaciones con la inversa de la matriz

Considere un problema típico de álgebra lineal, como:

$$AX = B, \qquad \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{21} \\ x_{31} \end{bmatrix} = \begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \end{bmatrix} \tag{1}$$

Queremos resolver para X, por lo que obtenemos el inverso de A y hacemos lo siguiente:

$$X = A^{-1}B, \qquad \begin{bmatrix} x_{11} \\ x_{21} \\ x_{31} \end{bmatrix} = \begin{bmatrix} ai_{11} & ai_{12} & ai_{13} \\ ai_{21} & ai_{22} & ai_{23} \\ ai_{31} & ai_{32} & ai_{33} \end{bmatrix} \begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \end{bmatrix} \tag{2}$$

Por lo tanto, tenemos un motivo para encontrar $A^{-1}$ . Entonces, ¿cómo encontramos fácilmente $A^{-1}$ de una manera que esté lista para la codificación?

Con frecuencia hacemos un uso inteligente de " multiplicar por 1 " para facilitar el álgebra. Una forma de " multiplicar por 1 " en álgebra lineal es usar la matriz de identidad. En caso de que haya venido aquí sin saber o estar oxidado en su álgebra lineal, la matriz de identidad es una matriz cuadrada (el número de filas es igual al número de columnas) con 1 en la diagonal y 0 en todas partes, como en la siguiente Matriz de identidad $3 \times 3$.

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

Siguiendo la regla principal del álgebra (hagamos lo que hagamos a un lado del signo igual, haremos al otro lado del signo igual, para "mantenernos fieles" al signo igual), realizaremos operaciones de fila a $A$ para convertirlo metódicamente en una matriz de identidad mientras se aplican esos mismos pasos a lo que es "inicialmente" la matriz de identidad. Cuando lo que era $A$ se convierte en una matriz de identidad, $I$ será $A^{-1}$ . Es importante tener en cuenta que $A$ debe ser una matriz cuadrada para ser invertida. Esto significa que el número de filas de $A$ y el número de columnas de $A$ deben ser iguales.

Si en algún momento, tienes un gran "¡Ah, HA!" momento, intente trabajar por su cuenta y compare con lo que hemos hecho a continuación una vez que haya terminado o eche un vistazo a las cosas a continuación lo menos posible SI se atasca.

Para encontrar $A^{-1}$ fácilmente, premultiplica $B$ por la matriz de identidad y realiza operaciones de fila en $A$ para conducirlo a la matriz de identidad. Desea hacer este elemento a la vez para cada columna de izquierda a derecha.

$$AX = IB, \qquad \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{21} \\ x_{31} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \end{bmatrix} \tag{4}$$

Realice las mismas operaciones de fila en $I$ que está realizando en $A$ , y $I$ convertiré en el inverso de $A$ (es decir, $A^{-1}$).

$$IX = A^{-1}B, \qquad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{21} \\ x_{31} \end{bmatrix} = \begin{bmatrix} ai_{11} & ai_{12} & ai_{13} \\ ai_{21} & ai_{22} & ai_{23} \\ ai_{31} & ai_{32} & ai_{33} \end{bmatrix} \begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \end{bmatrix} \tag{5}$$

Estos son los pasos, $S$ , que seguiríamos para hacer esto para cualquier tamaño de matriz. Esta es solo una descripción general de alto nivel. Haremos una descripción detallada de los números poco después de esto. Piense en el método de inversión como un conjunto de pasos paracada columna de izquierda a derecha y para cada elemento en la columna actual, y cada columna tiene uno de los elementos diagonales, que se representan como los elementos diagonales $S_{k1}$ donde $k = 1$ a $n$. Comenzaremos con la columna más a la izquierda y trabajaremos a la derecha. Y tenga en cuenta que cada $S$ representa un elemento que estamos utilizando para escalar. Cuando estamos en un cierto paso, $S_{ij}$, donde $i$ y $j = 1$ a $n$ independientemente dependiendo de dónde nos encontremos en la matriz, estamos realizando ese paso en toda la fila y usando la fila con la diagonal $S_{k1}$ como parte de esa operación.

$$S = \begin{bmatrix} S_{11} & \dots & \dots & S_{k2} & \dots & \dots & S_{n2} \\ S_{12} & \dots & \dots & S_{k3} & \dots & \dots & S_{n3} \\ \vdots & & & \vdots & & & \vdots \\ S_{1k} & \dots & \dots & S_{k1} & \dots & \dots & S_{nk} \\ \vdots & & & \vdots & & & \vdots \\ S_{1n-1} & \dots & \dots & S_{kn-1} & \dots & \dots & S_{nn-1} \\ S_{1n} & \dots & \dots & S_{kn} & \dots & \dots & S_{n1} \end{bmatrix} \tag{6}$$

[4]:
```
def zeros_matrix(rows, cols):
    """
    Creates a matrix filled with zeros.
        :param rows: the number of rows the matrix should have
        :param cols: the number of columns the matrix should have
        :returns: list of lists that form the matrix.
    """
    M = []
```

```
    while len(M) < rows:
        M.append([])
        while len(M[-1]) < cols:
            M[-1].append(0.0)

    return M

def identity_matrix(n):
    """
    Creates and returns an identity matrix.
        :param n: the square size of the matrix
        :returns: a square identity matrix
    """
    I = zeros_matrix(n, n)
    for i in range(n):
        I[i][i] = 1.0

    return I

def copy_matrix(M):
    """
    Creates and returns a copy of a matrix.
        :param M: The matrix to be copied
        :return: The copy of the given matrix
    """
    rows = len(M)
    cols = len(M[0])

    MC = zeros_matrix(rows, cols)

    for i in range(rows):
        for j in range(rows):
            MC[i][j] = M[i][j]

    return MC

def invert_matrix(A, n):
    """
    Returns the inverse of the passed in matrix.
        :param A: The matrix to be inversed
        :return: The inverse of the matrix A
    """

    # Section 2: Make copies of A & I, AM & IM, to use for row operations
    AM = copy_matrix(A)
    I = identity_matrix(n)
    IM = copy_matrix(I)
```

```
    # Section 3: Perform row operations
    indices = list(range(n)) # to allow flexible row referencing ***
    for fd in range(n): # fd stands for focus diagonal
        fdScaler = 1.0 / AM[fd][fd]
        # FIRST: scale fd row with fd inverse.
        for j in range(n): # Use j to indicate column looping.
            AM[fd][j] *= fdScaler
            IM[fd][j] *= fdScaler
        # SECOND: operate on all rows except fd row as follows:
        for i in indices[0:fd] + indices[fd+1:]: # *** skip row with fd in it.
            crScaler = AM[i][fd] # cr stands for "current row".
            for j in range(n): # cr - crScaler * fdRow, but one element at a␣
 ↪time.
                AM[i][j] = AM[i][j] - crScaler * AM[fd][j]
                IM[i][j] = IM[i][j] - crScaler * IM[fd][j]

    # Section 4: Make sure that IM is an inverse of A within the specified␣
 ↪tolerance
    return IM

start_inv = process_time_ns()

A_I = invert_matrix(A, n)
#A_I
AI = np.array(A_I)
XXX = AI.dot(C)

stop_inv = process_time_ns()
total_time_inv = stop_inv - start_inv
print("El tiempo computacion para determinar la inversa y calcular la matriz es:␣
 ↪{:.2f}".format(total_time_inv) + " [ns]")
df = pd.DataFrame(data=XXX)
print("EL vector solución para la matriz dada es:")
df
```

```
El tiempo computacion para determinar la inversa y calcular la matriz es:
1140625000.00 [ns]
EL vector solución para la matriz dada es:
```

[4]:

| | 0 |
|---|---|
| 0 | -1.265471 |
| 1 | -1.440506 |
| 2 | 0.816436 |
| 3 | -1.087906 |
| 4 | -2.385910 |
| .. | ... |

```
95  0.817862
96  0.283238
97  0.917211
98 -0.546842
99 -0.943914

[100 rows x 1 columns]
```

### 1.3 Factorización LU

La factorización o descomposición LU (del inglés Lower-Upper) es una forma de factorización de una matriz como el producto de una matriz triangular inferior y una superior. Debido a la inestabilidad de este método, deben tenerse en cuenta algunos casos especiales, por ejemplo, si uno o varios elementos de la diagonal principal de la matriz a factorizar es cero, es necesario premultiplicar la matriz por una o varias matrices elementales de permutación. Esta descomposición se usa en el análisis numérico para resolver sistemas de ecuaciones (más eficientemente) o encontrar las matrices inversas.

[5]:
```
U = A
L = np.eye(n)

def LU(U, L, n):
    for k in range(n-1):
        for j in range(n-1):
            t = -(U[j+k+1, k])/(U[k, k])
            L[j+k+1, k] = -t

            for i in range(n):
                if(k == 1 and i == n-1):
                    return (U, L)
                U[j+k+1, i+k] = t * U[k, i+k] + U[j+1+k, i+k]

start_time_LU = process_time_ns()
(U, L) = LU(U, L, n)
stop_time_LU = process_time_ns()

total_time_LU = stop_time_LU - start_time_LU

d = np.linalg.solve(L, C)
x = d = np.linalg.solve(U, d)

print("El tiempo computacion para determinar la descomposión LU y calcular la␣
 ↪matriz es: {:.2f}".format(total_time_LU) + " [ns]")
df = pd.DataFrame(data=x)
print("EL vector solución para la matriz dada es:")
df
```

```
El tiempo computacion para determinar la descomposión LU y calcular la matriz
es: 15625000.00 [ns]
EL vector solución para la matriz dada es:
```

[5]:
```
           0
0  -1.265471
1  -1.440506
2   0.816436
3  -1.087906
4  -2.385910
..       ...
95  0.817862
96  0.283238
97  0.917211
98 -0.546842
99 -0.943914

[100 rows x 1 columns]
```

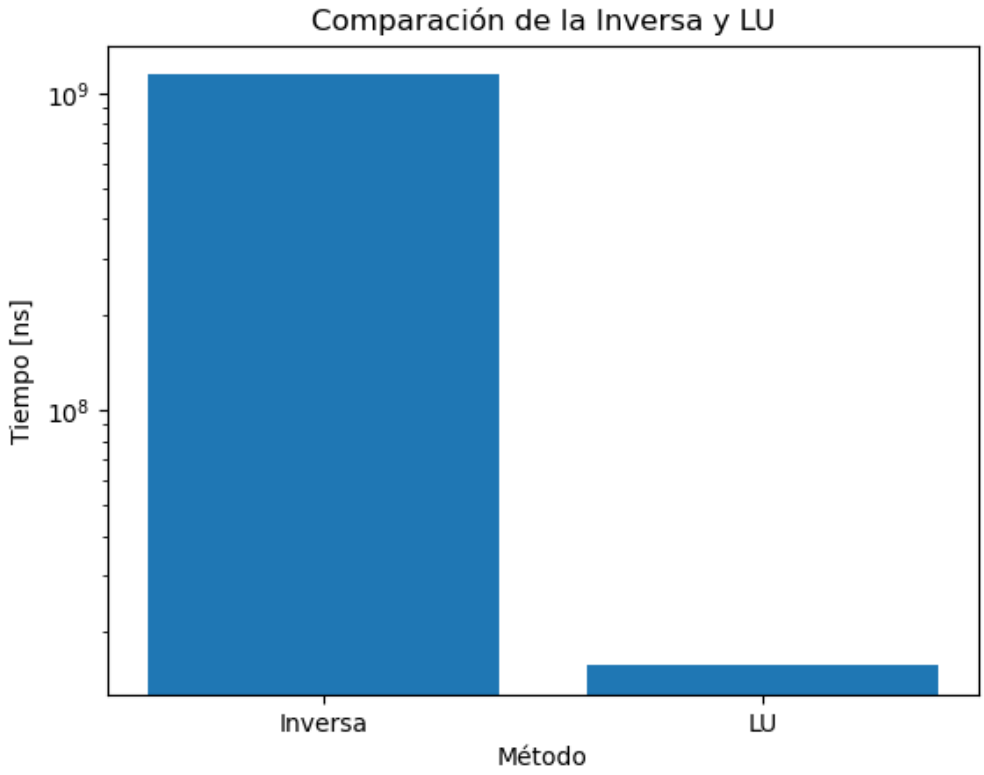
### 1.4 Comparación de tiempos computacionales

[6]:
```
rel = total_time_inv/total_time_LU
print("Relación entre el tiempo de la inversa y el de LU es: {:.0f}".format(rel))

inv_time = pd.Series(total_time_inv)
LU_time = pd.Series(total_time_LU)

frame = { 'Tiempo Inversa': inv_time, 'Tiempo LU': LU_time }

result = pd.DataFrame(frame)
result
```

```
Relación entre el tiempo de la inversa y el de LU es: 73
```

[6]:
```
   Tiempo Inversa  Tiempo LU
0      1140625000   15625000
```

[7]:
```
ejeX = ['Inversa', 'LU']
ejeY = [total_time_inv, total_time_LU]

plt.bar(ejeX, ejeY)
plt.title('Comparación de la Inversa y LU')
plt.xlabel('Método')
plt.ylabel('Tiempo [ns]')
plt.yscale("log")
```




## 1.5 Conclusiones

- Se concluye que el método más eficiente es el de descomposición LU y se puede apreciar que es 73 veces más que el de la inversa.
- Para apreciar un tiempo computacional razonable se debe tener una matriz de dimensiones de 1000 x 1000 debido a la rapidez del cálculo por parte del procesador.
- Se evidencia al realizar varias corridas que el valor de tiempo para la inversa varía, sin embargo, el tiempo computacional para LU se mantiene constante, esto debido a que la matriz es aletaria y en cada corrida es diferente, por tanto para el método de la inversa presenta algunas complicaciones.