Implementación de Algoritmos para Solución de Sistemas de Ecuaciones Lineales

Descripción

Existen diversos métodos para resolver Sistema de Ecuaciones Lineales y su uso y eficiencia depende en gran medida de las propiedades y característica que tenga su representación matricial correspondiente.

Durante esta primera entrega se implementaron diversos algoritmos de solución para ciertas matrices en especial.

1. Solución de una matriz diagonal
2. Solución de una matriz triangular superior mediante sustitución hacia atrás
3. Solución de una matriz triangular inferior mediante sustitución hacia atrás
4. Solución mediante Eliminación Gaussiana
5. Solución mediante Eliminación Gaussiana con Pivoteo
6. Solución mediante el Método de Doolittle
7. Solución mediante el Método de Cholesky Modificado
8. Generación de la matriz Inversa mediante el Método de Doolittle

Notemos que para todos los casos anteriores las matrices usadas son cuadradas.

Solución de una Matriz Diagonal

Dada un sistema de la forma $Ax = b$ con una Matriz Diagonal A:

$$\begin{pmatrix} a_{0,0} & 0 & 0 & \dots & 0 \\ 0 & a_{1,1} & 0 & \dots & 0 \\ 0 & 0 & a_{2,2} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{1}$$

La solución para cada x_i está dada por:

$$x_i = \frac{b_i}{a_{i,i}}, i \in [0, 1, \dots, n] \tag{2}$$

Algoritmo

```
n -> Tamaño de la matriz
a[n][n] -> Matriz A
b[n] -> Vector b
x[n] -> Vector solución de tamaño n

Desde i=0 hasta n:
    x[i] = b[i] / a[i][i]
```

Ejemplo de prueba

Entrada

Matriz *A*:

```
4 4

1.0 0.0 0.0 0.0
0.0 2.0 0.0 0.0
0.0 0.0 3.0 0.0
0.0 0.0 0.0 4.0
```

Matriz *b*:

```
4 1

1.0
2.0
3.0
4.0
```

Salida

```
1.000000 0.000000 0.000000 0.000000 1.000000
0.000000 2.000000 0.000000 0.000000 2.000000
0.000000 0.000000 3.000000 0.000000 3.000000
0.000000 0.000000 0.000000 4.000000 4.000000

X_1 = 1.000000
X_2 = 1.000000
X_3 = 1.000000
X_4 = 1.000000

Determinant: 24.000000
```

Observaciones y mejoras

- Dado que la propiedad de la matriz en este caso es ser diagonal la implementación de la solución es sumamente sencilla.
- Se se sabe de antemano que la matriz es una matriz diagonal puede ser guardado en un arreglo de tamaño *n* sin el caso así lo permite.

Solución de una matriz triangular superior mediante sustitución hacia atrás

Dada el sistema $Ax = b$ y una matriz triangular superior A de la forma:

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \dots & a_{0,n} \\ 0 & a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ 0 & 0 & a_{2,2} & \dots & a_{1,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{3}$$

La solución para cada x_i está dada por:

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{i,j} x_j}{a_{i,i}}, i \in [n, n-1, \dots, 0] \tag{4}$$

Algoritmo

```
n -> Tamaño de la matriz
a[n][n] -> Matriz A
b[n] -> Vector b
x[n] -> Vector solución de tamaño n

Desde i=n hasta 0:
    sol[i] = b[i]
    Desde j = i+i hasta n:
        sol[i] = sol[i] - (a[i][j] * x[j])
    sol[i] = sol[i] / a[i][i]
```

Ejemplo de prueba

Entrada

Matriz *A*:

```
4 4

1.0 2.0 3.0  4.0
0.0 5.0 6.0  7.0
0.0 0.0 8.0  9.0
0.0 0.0 0.0 10.0
```

Matriz *b*:

```
4 1

1.0
2.0
3.0
4.0
```

Salida

```
1.000000 2.000000 3.000000 4.000000 1.000000
0.000000 5.000000 6.000000 7.000000 2.000000
0.000000 0.000000 8.000000 9.000000 3.000000
0.000000 0.000000 0.000000 10.000000 4.000000

X_1 = -0.235000
X_2 = -0.070000
X_3 = -0.075000
X_4 = 0.400000

Determinant: 400.000000
```

Observaciones y mejoras

- Dado que solo se usa la mitad de la matriz, en vez de usar un una matriz de tamaño nxn podemos usar un arreglo de tamaño $\frac{n+1*(n+2)}{2}$, cuyo i-ésimo renglón tendría tamaño $n-i+1, i \in [0,1,\ldots,n]$.

Solución de una matriz triangular inferior mediante sustitución hacia atrás

Dada una matriz triangular inferior de la forma $Ax = b$:

$$\begin{pmatrix} a_{0,0} & 0 & 0 & \ldots & 0 \\ a_{1,0} & a_{1,1} & 0 & \ldots & 0 \\ a_{2,0} & a_{2,1} & a_{2,2} & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n,0} & a_{n,1} & a_{n,2} & \ldots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{5}$$

La solución para cada x_i está dada por:

$$x_i = \frac{b_i - \sum_{j=0}^{i-1} a_{i,j} x_j}{a_{i,i}}, i \in [0,1,\ldots,n] \tag{6}$$

Algoritmo

```
n -> Tamaño de la matriz
a[n][n] -> Matriz A
b[n] -> Vector b
x[n] -> Vector solución de tamaño n

Desde i=0 hasta n:
    sol[i] = b[i]
    Desde j = 0 hasta i-1:
        sol[i] = sol[i] - (a[i][j] * x[j])
    sol[i] = sol[i] / a[i][i]
```

Ejemplo de prueba

Entrada

Matriz *A*:

```
4 4

1.0 0.0 0.0  0.0
2.0 3.0 0.0  0.0
4.0 5.0 6.0  0.0
7.0 8.0 9.0 10.0
```

Matriz *b*:

```
4 1

1.0
2.0
3.0
4.0
```

Salida

```
1.000000 0.000000 0.000000 0.000000 1.000000
2.000000 3.000000 0.000000 0.000000 2.000000
4.000000 5.000000 6.000000 0.000000 3.000000
7.000000 8.000000 9.000000 10.000000 4.000000

X_1 = 1.000000
X_2 = 0.000000
X_3 = -0.166667
X_4 = -0.150000

Determinant: 180.000000
```

Observaciones y mejoras

Dado que solo se usa la mitad de la matriz, en vez de usar un una matriz de tamaño nxn podemos usar un arreglo de tamaño $\frac{n+1*(n+2)}{2}$, cuyo i-ésimo renglón tendría tamaño $i+1, i \in [0, 1, \ldots, n]$.

Solución mediante Eliminación Gaussiana

Dada una matriz triangular superior A y sistema de la forma $Ax = b$:

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \ldots & a_{0,n} \\ a_{1,0} & a_{1,1} & a_{1,2} & \ldots & a_{1,n} \\ a_{2,0} & a_{2,1} & a_{2,2} & \ldots & a_{1,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n,0} & a_{n,1} & a_{n,2} & \ldots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{7}$$

Podemos encontrar la solución expresándola en una matriz triangular superior y resolviendo dicha matriz mediante el proceso de Eliminación Gaussiana.

Algoritmo

```
n -> Tamaño de la matriz
a[n][n] -> Matriz A
b[n] -> Vector b
x[n] -> Vector solución de tamaño n

- Desde pivote=0 hasta n:
    - Si a[pivote][pivote] == 0:
        - Intercambiar fila pivote con alguna fila i, con i > pivote y
        donde a[i][pivote] != 0
        - Si no se encontró un cambio no hay solución única, terminar.
    - Desde i = pivote + i hasta n:
        - Desde j=0 hasta n:
            a[i][j] -= a[i][j] * a[pivote][j] / a[pivote][pivote]

- Resolver Matriz triangular superior resultante con sustitución hacia
atrás
```

Ejemplo de prueba

Entrada

Matriz *A*:

```
4 4

2.402822 4.425232 1.929374 1.370355
1.201411 2.212616 0.964687 0.685178
1.119958 0.964687 2.053172 0.566574
0.742142 0.685178 0.566574 1.696828
```

Matriz *b*:

```
4 1

0.060000
0.542716
0.857204
0.761270
```

Salida

```
2.402822 4.425232 1.929374 1.370355 0.060000
1.201411 2.212616 0.964687 0.685178 0.542716
1.119958 0.964687 2.053172 0.566574 0.857204
0.742142 0.685178 0.566574 1.696828 0.761270

X_1 = -5425480.766483
X_2 = 1837967.142749
X_3 = 1812934.000000
X_4 = 1025431.999682
```

Observaciones y mejoras

- Este algoritmo realiza un pivoteo básico de solo renglones al detectar que se ha formado un cero en la diagonal. Pero no minimiza el overflow causado por divisiones entre número muy pequeños o muy grandes.

Solución mediante Eliminación Gaussiana con Pivoteo

Dada una matriz triangular superior A y un sistema de la forma $Ax = b$:

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \dots & a_{0,n} \\ a_{1,0} & a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,0} & a_{2,1} & a_{2,2} & \dots & a_{1,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n,0} & a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{8}$$

Podemos encontrar la solución expresándola en una matriz triangular superior y resolviendo dicha matriz mediante el proceso de Eliminación Gaussiana.

Algoritmo

```
n -> Tamaño de la matriz
a[n][n] -> Matriz A
b[n] -> Vector b
x[n] -> Vector solución de tamaño n

- Desde pivote=0 hasta n:
    - Encontrar fila i y columna j, con i,j > pivote tal que ABS(a[i][j])
    sea el máximo distinto de cero.
```

```
    - Si no se encontró un máximo no hay solución única, terminar.
    - Intercambiar fila pivote con fila i
    - Intercambiar columna pivote por columna j
    - Desde i = pivote + i hasta n:
        - Desde j=0 hasta n:
            a[i][j] -= a[i][j] * a[pivote][j] / a[pivote][pivote]

- Resolver Matriz triangular superior resultante con sustitución hacia atrás
```

Ejemplo de prueba

Entrada

Matriz *A*:

```
4 4

2.402822 4.425232 1.929374 1.370355
1.201411 2.212616 0.964687 0.685178
1.119958 0.964687 2.053172 0.566574
0.742142 0.685178 0.566574 1.696828
```

Matriz *b*:

```
4 1

0.060000
0.542716
0.857204
0.761270
```

Salida

```
2.402822 4.425232 1.929374 1.370355 0.060000
1.201411 2.212616 0.964687 0.685178 0.542716
1.119958 0.964687 2.053172 0.566574 0.857204
0.742142 0.685178 0.566574 1.696828 0.761270

X_1 = -5425480.766483
X_2 = 1837967.142749
X_3 = 1812934.000000
X_4 = 1025431.999682
```

Observaciones y mejoras

- Aunque hace un pivoteo buscando un el valor más grande para no dividir un número muy pequeño aun puede provocar un overflow por dividir.

Solución mediante el Método de Doolittle

Dada una matriz triangular superior A y un sistema de la forma $Ax = b$:

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \dots & a_{0,n} \\ a_{1,0} & a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,0} & a_{2,1} & a_{2,2} & \dots & a_{1,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n,0} & a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{9}$$

La matriz A puede ser escrita como producto de dos matrices triangulares inferior y superior L y U respectivamente y L con una diagonal de unos.

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \dots & a_{0,n} \\ a_{1,0} & a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,0} & a_{2,1} & a_{2,2} & \dots & a_{1,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n,0} & a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ L_{1,0} & 1 & 0 & \dots & 0 \\ L_{2,0} & L_{2,1} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ L_{n,0} & L_{n,1} & L_{n,2} & \dots & 1 \end{pmatrix} \begin{pmatrix} U_{0,0} & U_{0,1} & U_{0,2} & \dots & U_{0,n} \\ 0 & U_{1,1} & U_{1,2} & \dots & U_{1,n} \\ 0 & 0 & U_{2,2} & \dots & U_{2,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & U_{n,n} \end{pmatrix} \tag{10}$$

Donde cada valor $L_{i,j}$ con $i > j$ y $U_{i,j}$ con $i < j$ están dado por:

$$U_{i,j} = a_{i,j} - \sum_{k=0}^{i-i} L_{i,k} U_{k,j}, i \leq j \tag{11}$$

$$L_{i,j} = \frac{a_{i,j} - \sum_{k=0}^{j-i} L_{i,k} U_{k,j}}{U_{j,j}}, i > j \tag{12}$$

Así podemos reescribir $Ax = b$ como $LUx = b$. Notemos que la multiplicación de $Ux = y$ es una matriz de tamaño $nx1$, por tanto resolvemos $Ly = b$ con y como nuestro vector de incógnitas mediante sustitución hacia atrás.

Una vez obtenido y podemos resolver $Ux = y$ nuevamente mediante sustitución hacia atrás para obtener los valores del vector x.

Algoritmo

El algoritmo siguiente guarda la Factorización LU en otra matriz, puede omitirse y guardarse sobre la matriz A si así se desea, con una ligera modificación en el proceso si se detectan ceros en la diagonal.

```
n -> Tamaño de la matriz
a[n][n] -> Matriz A
lu[n][n] -> Matriz LU, asumimos que para L la diagonal es de unos
b[n] -> Vector b
x[n] -> Vector solución de tamaño n

- Desde i=0 hasta n:
    - Desde j=0 hasta n:
        lu[i][j] = a[i][j]
        limite = i-1 Si i <= j de lo contrario j-1
        - Desde k=0 hasta limite:
            lu[i][j] -= lu[i][k] * lu[k][j]
        - Si i > j:
            lu[i][j] /= LU[j][j]
    - Si lu[i][i] == 0:
        - Cambiar renglon a[i][:] con algún renglón a[r][:] tal que r > i
        i = i - 1
```

```
        - Si ya se cambió con todos los renglones posibles,
        no existe solución única, terminar.

- Resolver Ly = b
- Resolver Ux = y
```

Ejemplo de prueba

Entrada

Matriz *A*:

```
4 4

2.402822 4.425232 1.929374 1.370355
1.201411 2.212616 0.964687 0.685178
1.119958 0.964687 2.053172 0.566574
0.742142 0.685178 0.566574 1.696828
```

Matriz *b*:

```
4 1

0.060000
0.542716
0.857204
0.761270
```

Salida

```
2.402822 4.425232 1.929374 1.370355 0.060000
1.201411 2.212616 0.964687 0.685178 0.542716
1.119958 0.964687 2.053172 0.566574 0.857204
0.742142 0.685178 0.566574 1.696828 0.761270

X_1 = -5425479.813547
X_2 = 1837966.776940
X_3 = 1812933.651960
X_4 = 1025432.000084

Determinant: 0.000001
```

Observaciones y mejoras

- Durante el proceso de construcción de las matrices L y U puede haber casos en los que se forman ceros en la diagonal de U. En este caso se optó por hacer un pivoteo de renglones con algún renglón aún no modificado durante el proceso. Si se ha probado con todos los renglones sin éxito significa que el sistema de ecuaciones es inconsistente y por tanto no hay solución única.

Solución mediante el Método de Cholesky Modificado

Dada una matriz triangular superior A un sistema de la forma $Ax = b$:

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \dots & a_{0,n} \\ a_{1,0} & a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,0} & a_{2,1} & a_{2,2} & \dots & a_{1,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n,0} & a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{13}$$

La matriz A puede ser escrita como producto de tres matrices LDL^t triangular inferior, diagonal y triangular superior respectivamente, y L con unos en la diagonal.

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \dots & a_{0,n} \\ a_{1,0} & a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,0} & a_{2,1} & a_{2,2} & \dots & a_{1,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n,0} & a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{pmatrix} \tag{14}$$

$$= \tag{15}$$

$$\begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ L_{1,0} & 1 & 0 & \dots & 0 \\ L_{2,0} & L_{2,1} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ L_{n,0} & L_{n,1} & L_{n,2} & \dots & 1 \end{pmatrix} \begin{pmatrix} D_{0,0} & 0 & 0 & \dots & 0 \\ 0 & D_{1,1} & 0 & \dots & 0 \\ 0 & 0 & D_{2,2} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & D_{n,n} \end{pmatrix} \begin{pmatrix} 1 & L_{1,0} & L_{2,0} & \dots & U_{n,0} \\ 0 & 1 & U_{2,1} & \dots & U_{n,1} \\ 0 & 0 & 1 & \dots & U_{n,1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix} \tag{16}$$

Donde cada valor $L_{i,j}$ con $i > j$ y $U_{i,j}$ con $i < j$ están dado por:

$$L_{i,j} = \frac{a_{i,j} - \sum_{k=0}^{j-i} L_{i,k} D_{k,k} L_{j,k}}{d_{j,j}}, i \neq j \tag{17}$$

$$L_{i,i} = a_{i,i} - \sum_{k=0}^{i-i} L_{i,k}^2 D_{k,k} \tag{18}$$

Así podemos reescribir $Ax = b$ como $LDL^{-1}x = b$.

Notemos que la multiplicación de LD también es una matriz triangular inferior, y la multiplicación de $L^t x = y$ es una matriz de $nx1$.

Por tanto resolvemos $LDy = b$ con y como nuestro vector de incógnitas mediante sustitución hacia atrás.

Una vez obtenido y podemos resolver $L^t x = y$ nuevamente mediante sustitución hacia atrás para obtener los valores del vector x.

Algoritmo

El algoritmo siguiente guarda la Factorización LU en otra matriz, puede omitirse y guardarse sobre la matriz A si así se desea, con una ligera modificación en el proceso si se detectan ceros en la diagonal.

```
1  n -> Tamaño de la matriz
2  a[n][n] -> Matriz A
3  ldlt[n][n] -> Matriz LU, asumimos que para L la diagonal es de unos
4  b[n] -> Vector b
5  x[n] -> Vector solución de tamaño n
6
```

```
- Desde i=0 hasta n:
    - Desde j=0 hasta n:
        ldlt[i][j] = a[i][j]
        limite = i-1 Si i == j de lo contrario j-1
        - Desde k=0 hasta limite:
            ldlt[i][j] -= ldlt[i][k] * ldlt[k][j]
        - Si i != j:
            ldlt[i][j] /= ldlt[j][j]
    - Si ldlt[i][i] == 0:
        - Cambiar renglon a[i][:] con algún renglón a[r][:] tal que r > i
        i = i - 1
        - Si ya se cambió con todos los renglones posibles,
        no existe solución única, terminar.

- Resolver LDy = b
- Resolver L^-1x = y
```

Ejemplo de prueba

Entrada

Matriz *A*:

```
10 10

3192.302 2698.635 1978.379 2126.596 2790.948 3222.904 2115.401 2977.436 2323.447 2064.871
2698.635 3648.584 2353.185 2553.399 2775.733 2717.937 2426.996 3440.832 2690.923 1824.800
1978.379 2353.185 2832.480 1721.412 2479.264 2633.894 2114.959 2189.508 2124.635 1680.251
2126.596 2553.399 1721.412 2457.944 1722.838 2208.851 1909.113 2425.035 1814.554 1303.733
2790.948 2775.733 2479.264 1722.838 3632.273 3450.131 2061.848 2784.201 2362.622 2397.096
3222.904 2717.937 2633.894 2208.851 3450.131 4206.092 2283.862 2789.959 2211.154 2503.426
2115.401 2426.996 2114.959 1909.113 2061.848 2283.862 2666.696 2482.297 2241.853 1505.690
2977.436 3440.832 2189.508 2425.035 2784.201 2789.959 2482.297 3883.093 2756.404 1785.132
2323.447 2690.923 2124.635 1814.554 2362.622 2211.154 2241.853 2756.404 2866.823 1814.375
2064.871 1824.800 1680.251 1303.733 2397.096 2503.426 1505.690 1785.132 1814.375 2255.474
```

Matriz *b*:

```
10 1

1
2
3
4
5
6
7
8
9
10
```

Salida

```
3192.302000 2698.635000 1978.379000 2126.596000 2790.948000 3222.904000 2115.401000 2977.436000 2323.447000 2064.871000 1.000000
2698.635000 3648.584000 2353.185000 2553.399000 2775.733000 2717.937000 2426.996000 3440.832000 2690.923000 1824.800000 2.000000
1978.379000 2353.185000 2832.480000 1721.412000 2479.264000 2633.894000 2114.959000 2189.508000 2124.635000 1680.251000 3.000000
2126.596000 2553.399000 1721.412000 2457.944000 1722.838000 2208.851000 1909.113000 2425.035000 1814.554000 1303.733000 4.000000
2790.948000 2775.733000 2479.264000 1722.838000 3632.273000 3450.131000 2061.848000 2784.201000 2362.622000 2397.096000 5.000000
3222.904000 2717.937000 2633.894000 2208.851000 3450.131000 4206.092000 2283.862000 2789.959000 2211.154000 2503.426000 6.000000
2115.401000 2426.996000 2114.959000 1909.113000 2061.848000 2283.862000 2666.696000 2482.297000 2241.853000 1505.690000 7.000000
2977.436000 3440.832000 2189.508000 2425.035000 2784.201000 2789.959000 2482.297000 3883.093000 2756.404000 1785.132000 8.000000
2323.447000 2690.923000 2124.635000 1814.554000 2362.622000 2211.154000 2241.853000 2756.404000 2866.823000 1814.375000 9.000000
2064.871000 1824.800000 1680.251000 1303.733000 2397.096000 2503.426000 1505.690000 1785.132000 1814.375000 2255.474000 10.000000

X_1 = -0.028663
X_2 = -0.019074
X_3 = -0.008506
X_4 = 0.009017
X_5 = -0.001359
X_6 = 0.012221
X_7 = 0.000322
X_8 = 0.019538
X_9 = 0.010652
X_10 = 0.010865

Determinant: 48749394839768796131856744448.000000
```

Observaciones y mejoras

- Durante el proceso de construcción de las matrices L, D puede haber casos en los que se forman ceros en la diagonal de D. En este caso se optó por hacer un pivoteo de renglones con algún renglón aún no modificado durante el proceso. Si se ha probado con todos los renglones sin éxito significa que el sistema de ecuaciones es inconsistente y por tanto no hay solución única.

Generación de la matriz Inversa mediante el Método de Doolittle

Dada una matriz cuadrada A de la forma:

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \dots & a_{0,n} \\ a_{1,0} & a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,0} & a_{2,1} & a_{2,2} & \dots & a_{1,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n,0} & a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{pmatrix} \tag{19}$$

Si A es invertible podemos encontrar una matriz A' talque $AA' = I$, dónde I es la matriz identidad.

Factorizamos la matriz A en las sus dos matrices LU donde L tiene unos en la diagonal.

La i-ésima columna de la matriz A' estará dado por la solución de:

$$A'[:][i] = LUI[:][i], i \in [1, n] \tag{22}$$

Algoritmo

```
n -> Tamaño de la matriz
A[n][n] -> Matriz A
A'[n][n] -> Inversa de A
I'[n][n] -> Matriz identidad
LU[n][n] -> Factorización LU de A, asumimos que al usar L debe tener unos en la diagonal
y[n] -> Vector solución de tamaño n
x[n] -> Vector solución de tamaño n

- Factorizar A en LU
- Desde i=0 hasta n:
    y = Resolver Ly=I[:][i]
    x = Resolver Ux=y
    A'[:][i] = x
```

Ejemplo de prueba

Entrada

Matriz *A*:

```
4 4

2.402822 4.425232 1.929374 1.370355
1.201411 2.212616 0.964687 0.685178
1.119958 0.964687 2.053172 0.566574
0.742142 0.685178 0.566574 1.696828
```

Salida

```
2.402822 4.425232 1.929374 1.370355
1.201411 2.212616 0.964687 0.685178
1.119958 0.964687 2.053172 0.566574
0.742142 0.685178 0.566574 1.696828

5290922.867794 -10581846.610143 -0.602503 3.672447
-1792383.492965 3584767.889953 -0.035834 -1.409393
-1767970.895007 3535941.842308 0.832539 -1.341029
-1000000.000082 2000000.000165 0.000000 0.000000

1.000000 -0.000000 0.000000 0.000000
0.000000 1.000000 0.000000 0.000000
0.000000 -0.000000 1.000000 0.000000
0.000000 -0.000000 0.000000 1.000000
```

Observaciones y mejoras

- En la implementación en código dado que la Factorización LU se realiza sobre la misma matriz A se necesita hacer una copia de la matriz para poder imprimir el producto de AxA'. En casos dónde no se necesite se puede obviar la creación de un duplicado de la matriz A.