Informe Laboratorio: Análisis Numérico
Práctica No. 5

Daniel Delgado
Código: 2182066
Grupo: B2
Escuela de Ingeniería de Sistemas e Informática
Universidad Industrial de Santander

5 de febrero de 2021

1. Introducción

Los sistemas de ecuaciones hacen parte importante dentro de los conceptos matemáticos trabajados debido a que estos permiten la determinación de los valores de una n cantidad de variables a partir de unas condiciones dadas. Una de las maneras de dar solución a estos sistemas, al igual que ser una de las soluciones empleadas en términos computacionales, se encuentra la descomposición LU de sistemas matriciales.

Este método, basado en la descomposición de una matriz en dos términos L y U, se utilizan estos 2 términos para poder dar solución a los sistemas de ecuaciones dados a partir de la aplicación de despejes de las ecuaciones resultantes en 2 pasos.

La compresión de esta manera de solucionar sistemas de ecuaciones, al igual que el desarrollo de la algoritmia relacionada, son los principales temas a a tratar durante el desarrollo del presente informe, así como la resolución de los problemas propuestos a manera de pregunta orientadora durante el desarrollo del componente práctico del mismo.

2. Desarrollo

1. Preguntas propuestas

   a) ¿Qué es una factorización LU?
   Una factorización LU es un tipo de factorización que se realiza principalmente en sistemas de ecuaciones con el fin de dar solución a estos. Esta se realiza a partir de la factorización de los valores presentes en la matriz en 2 partes, una parte superior, llamada U; y la parte inferior, L, tal que para una matriz A de tamaño $n \times n$, se cumple $A = LU$.
   La factorización LU tiene comúnmente la siguiente forma:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1j} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2j} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3j} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{i1} & a_{i2} & a_{i3} & \cdots & a_{ij} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{21} & 1 & 0 & \cdots & 0 \\ l_{31} & l_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{i1} & l_{i2} & l_{i3} & \cdots & 1 \end{bmatrix} \cdot \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1j} \\ 0 & u_{22} & u_{23} & \cdots & u_{2j} \\ 0 & 0 & u_{33} & \cdots & u_{3j} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{ij} \end{bmatrix} \tag{1}$$

   Sin embargo, esta puede verse modificada en el caso de que se realicen permutaciones las cuales afecten el orden de las filas.

   b) ¿Cómo se realiza una factorización LU?
   Existen varias maneras de realizar una factorización LU, sin embargo, en el presente informe se trabajará la factorización LU a partir de una eliminación Gaussiana de una matriz.

De manera general para realizar una factorización LU, se deberá realizar una eliminación Gaussiana únicamente sobre los valores propios de la matriz, es decir, sobre los valores presentes en la matriz A sin simplificar las columnas y ubicar los valores de los factores usados durante la eliminación en sus posiciones correspondientes en una matriz identidad. Un ejemplo de esto se puede ver a continuación:

$$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}\begin{bmatrix}3&3&9\\4&8&0\\2&4&2\end{bmatrix}\begin{matrix}\leftarrow Pivote\\ f_{21}=\frac{4}{3}\\ f_{31}=\frac{2}{3}\end{matrix}\longrightarrow\begin{bmatrix}1&0&0\\\frac{4}{3}&1&0\\\frac{2}{3}&0&1\end{bmatrix}\begin{bmatrix}3&3&9\\0&4&-12\\0&2&-4\end{bmatrix}$$

$$\begin{bmatrix}1&0&0\\\frac{4}{3}&1&0\\\frac{2}{3}&0&1\end{bmatrix}\begin{bmatrix}3&3&9\\0&4&-12\\0&2&-4\end{bmatrix}\begin{matrix}\leftarrow Pivote\\ f_{32}=\frac{1}{2}\end{matrix}\longrightarrow\begin{bmatrix}1&0&0\\\frac{4}{3}&1&0\\\frac{2}{3}&\frac{1}{2}&1\end{bmatrix}\begin{bmatrix}3&3&9\\0&4&-12\\0&0&2\end{bmatrix}$$

De esto, tenemos 2 matrices resultantes. La matriz de la izquierda siendo L y la matriz de la derecha U dando como resultado una matriz factorizada por LU.

2. Implementación

   a) factorización triangular LU

   De manera inicial, se plantea la necesidad de realizar un algoritmo que permita la factorización LU de una matriz de $n \times n$. Con el fin de cumplir este proceso, se realizó la función `luFact(inTrix)`. Esta función, en términos simples, realiza la factorización de una matriz cuadrada de cualquier tamaño a partir del proceso descrito en la parte b las preguntas propuestas.

   A continuación, se presenta la función `luFact(intrix)`:

```
function [L, U] = luFact(inTrix)
  [vert, hori] = size(inTrix);
  L = NaN;
  U = NaN;
  counter = 1;

  if vert == hori
    U = inTrix;
    L = eye(vert, hori);

    for outer = 1:hori
      counter = counter + 1;

      temp = U(outer, :);
      tempFac = temp / temp(outer);

      for subber = counter:vert
        aux = tempFac * U(subber, outer);
        L(subber, outer) = aux(1, outer) / temp(outer);
        U(subber, :) = U(subber, :) - aux;
      end
    end
  end
end
```

   La función, de manera inicial, determina y asigna el tamaño de la matriz de entrada a las variables `vert`, para la altura de esta; y `hori`, para el ancho de la matriz. De igual manera, asigna a los valores de salida `L` y `U` en `NaN` e inicializa un contador independiente que será usado para determinar la posición de los elementos de la matriz.

   Seguidamente, y tras verificar que efectivamente la matriz de entrada sea una matriz cuadrada, se modificaran los valores de `U` y `L` a ser la matriz de entrada y una matriz identidad respectivamente. Esto se realiza debido a la forma que tienen cada una de las matrices de salida.

Tras esto, se inicia el primer bucle for con el fin de realizar los cálculos respectivos del pivote, trabajado como `temp`, y `tempFac` refiriéndose al pivote simplificado. Seguidamente, se inicial el bucle interno que es el responsable de realizar las operaciones entre el pivote y las demás filas, al igual que guardar los factores empleados en la matriz identidad.

Finalmente, tras la realización de todas la iteraciones del bucle externo, se dará fin a la función y salida de las matrices `L` y `U` correspondientes para la matriz `inTrix` dada.

b) Comprobación de la función `luFact(inTrix)`
Con el fin de comprobar el funcionamiento, se factorizará la matriz A con la función `luFact(inTrix)`. Con el fin de realizar esto, se creó el script de ejecución `triangRunner.m`. En este script, se define el valor de la matriz A, como es prepuesto por la guía de laboratorio, y se ejecuta la función `luFact`.

```
A = [1 1 0 4; 2 -1 5 0; 5 2 1 2; -3 0 2 6];

[L, U] = luFact(A);

L
U
```

Tras la ejecución en la consola, se imprimen los siguientes resultados:

```
L =

    1.0000         0         0         0
    2.0000    1.0000         0         0
    5.0000    1.0000    1.0000         0
   -3.0000   -1.0000   -1.7500    1.0000


U =

    1.0000    1.0000         0    4.0000
         0   -3.0000    5.0000   -8.0000
         0         0   -4.0000  -10.0000
         0         0         0   -7.5000
```

De esto, podemos apreciar varias cosas. De manera principal, es posible apreciar el como estas salidas tienen la forma descrita con anterioridad en la ecuación (1) lo que es un primer indicador respecto al funcionamiento correcto de la función.

Seguidamente, y como comprobación final del funcionamiento de `luFact(inTrix)`, se realizará la multiplicación de `L` y `U` debido a que, como está definida la factorización LU, $A = LU$. En la misma consola se realizó `L*U` y, tras ejecución, dió como resultado:

```
ans =

     1     1     0     4
     2    -1     5     0
     5     2     1     2
    -3     0     2     6
```

Comprobando así el funcionamiento correcto de la función `luFact(inTrix)` para la realización de factorizaciones LU.

c) Solución de un sistema de ecuaciones Una de las aplicaciones principales de la factorización LU es la solución de sistemas de ecuaciones. Esta manera de solucionar sistemas de ecuaciones se realiza en 2 pasos, resolviendo $LY = B$ para Y y luego resolviendo $UX = Y$ para X. Con el fin de probar esto, se buscó dar solución al sistema de ecuaciones planteado:

$$\begin{bmatrix} 1 & 1 & 0 & 4 \\ 2 & -1 & 5 & 0 \\ 5 & 2 & 1 & 2 \\ -3 & 0 & 2 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 3 \\ 60 \\ 1 \\ 5 \end{bmatrix}$$

Con el fin de realizar esto, se planteó `triangularASolver(A, B)`. Esta función realiza el despeje de las variables de un sistema de ecuaciones matricial de 4 variables.

```
function output = triangularASolver(A, B)
  [vert, hori] = size(A);
  [alt, anc] = size(B);

  if vert == hori && vert == alt && anc == 1
    [L, U] = luFact(A);

    y(1, 1) = B(1, 1);
    y(2, 1) = B(2, 1) - y(1, 1) * L(2, 1);
    y(3, 1) = B(3, 1) - y(1, 1) * L(3, 1) - y(2, 1) * L(3, 2);
    y(4, 1) = B(4, 1) - y(1, 1) * L(4, 1) - y(2, 1) * L(4, 2) - y(3, 1) * L(4, 3);

    x(4, 1) = y(4, 1) / U(4, 4);
    x(3, 1) = (y(3, 1) - x(4, 1) * U(3, 4)) / U(3, 3);
    x(2, 1) = (y(2, 1) - x(4, 1) * U(2, 4) - x(3, 1) * U(2, 3)) / U(2, 2);
    x(1, 1) = (y(1, 1) - x(4, 1) * U(1, 4) - x(3, 1) * U(1, 3) - x(2, 1) * U(1, 2)) / U(1, 1);

    output = x;
  end
end
```

De manera simple, podemos observar el como, tras realizar comprobaciones del tamaño de la matriz y el vector solución, se llama a la función `luFact(A)` con el fin de realizar la factorización LU de la matriz A de entrada y se asigna a las variables locales `L` y `U` respectivamente. Seguidamente, se realiza de manera directa el despeje de Y de la ecuación $LY = B$ y, a partir de esto, el despeje de X de la ecuación $UX = Y$ y la salida de la función.

Con el fin de probar esto, se creó `triangRunner2.m`, el cual es una script que contiene tanto la matriz y el vector necesarios para el realizar el despeje del sistema de ecuaciones.

```
A = [1 2 -3 4; 4 8 12 -8; 2 3 2 1; -3 -1 1 -4];

B = [3; 60; 1; 5];

triangularASolver(A, B)
```

Tras la ejecución en la consola, se imprimen los siguientes resultados:

```
ans =

   NaN
   NaN
   NaN
   NaN
```

Esto, en términos simples, se debe a que los valores presentes en la matriz, específicamente los valores entre las dos primeras filas, en el momento de realizar la eliminación Gaussiana, coinciden los factores para eliminar más valores de lo esperado como se puedo apreciar a continuación:

$$\begin{matrix} Pivote \rightarrow \\ f_{21} = 4 \\ f_{31} = 2 \\ f_{41} = -3 \end{matrix} \begin{bmatrix} 1 & 2 & -3 & 4 \\ 4 & 8 & 12 & -8 \\ 2 & 3 & 2 & 1 \\ -3 & -1 & 1 & -4 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 2 & -3 & 4 \\ 0 & 0 & 24 & -24 \\ 0 & -1 & 8 & -7 \\ 0 & 5 & -8 & 8 \end{bmatrix}$$

Con esto podemos ver una de las limitaciones que tiene `luFact(inTrix)`, puesto que en casos donde sea necesario realizar algún tipo de movimiento de filas para así factorizar el valor, la función no está en la capacidad de realizar dichos movimientos. Sin embargo, con el fin de resolver el sistema, se modificará manualmente el sistema de ecuaciones. Esta modificación únicamente afectará el como se ve la el sistema de ecuaciones pero no los resultados.

$$\begin{bmatrix} 1 & 1 & 0 & 4 \\ 2 & -1 & 5 & 0 \\ 5 & 2 & 1 & 2 \\ -3 & 0 & 2 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 3 \\ 60 \\ 1 \\ 5 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 1 & 0 & 4 \\ 5 & 2 & 1 & 2 \\ -3 & 0 & 2 & 6 \\ 2 & -1 & 5 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \\ 5 \\ 60 \end{bmatrix}$$

De igual manera, se creó `triangRunner3.m`, el cual contiene la versión modificada del sistema de ecuaciones.

```
moddedA = [1 2 -3 4; 2 3 2 1; -3 -1 1 -4; 4 8 12 -8];

moddedB = [3; 1; 5; 60];

triangularASolver(moddedA, moddedB)
```

Tras la ejecución en la consola, se imprimen los siguientes resultados:

```
ans =

    12
     6
   -13
   -15
```

Ahora, como es posible apreciar, tras la modificación de forma realizada al sistema de ecuaciones, se ejecuta la función de la manera esperada dando como resultado los valores del vector x. Con el fin de comprobar que efectivamente sean los valores correctos, se ejecutó en consola `moddedA*ans` dando como resultado los siguientes valores:

```
ans =

     3
     1
     5
    60
```

Confirmando así los valores correctos para solucionar el sistema de ecuaciones planteado.

$$\begin{bmatrix} 1 & 1 & 0 & 4 \\ 5 & 2 & 1 & 2 \\ -3 & 0 & 2 & 6 \\ 2 & -1 & 5 & 0 \end{bmatrix} \begin{bmatrix} 12 \\ 6 \\ -13 \\ -15 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \\ 5 \\ 60 \end{bmatrix}$$

d) Comparación de algoritmos Dentro de las funciones presentes en las librerías por defecto de Matlab, se encuentra presente la función `lu(A)`. Esta función realiza el mismo proceso realizado por la función `luFact(inTrix)` aunque de maneras ligeramente diferentes.

La principal diferencia puede verse al ejecutar el script `functionComparator.m`.

```
A = [3, 5, 6; 7, 7, 1; 3, 8, 9];

[mL, mU] = lu(A)
[L, U] = luFact(A)
```

```
mL =

    0.4286   0.4000   1.0000
    1.0000        0        0
    0.4286   1.0000        0


mU =

    7.0000   7.0000   1.0000
         0   5.0000   8.5714
         0        0   2.1429


L =

    1.0000        0        0
    2.3333   1.0000        0
    1.0000  -0.6429   1.0000


U =

    3.0000   5.0000   6.0000
         0  -4.6667 -13.0000
         0        0  -5.3571
```

De manera principal, es posible ver el como las matrices de salida de `lu(A)` no coinciden ni en forma ni valores en comparación con `luFact(A)`. Dentro de una análisis simple, esto puede deberse a que, la función `lu(A)`, está en la capacidad de realizar diferentes permutaciones las cuales permiten la optimización, facilidad del cálculo y precisión de la factorización. Sin embargo, en términos generales, ambas funciones dan el mismo resultado aunque `luFact(inTrix)` tiende a ser más limitada.

3. Interpretación

   De manera inicial se nos plantea la situación de Sophia. Ella vende fotos a diferentes precios y se nos plantea la necesidad de calcular la cantidad de cada una de las fotos para poder cubrir los costos de producción bajo algunas condiciones.

   a) Sistema de ecuaciones

   La información que se nos plantea se basa en tres partes. La primera se refiere a que, el costo a cubrir de la venta de las fotos es de 300$. Teniendo este valor sabemos que la suma de las fotos miniatura (f_m), fotos normales (f_n) y enormes (f_e) multiplicados por sus respectivos precios de venta, dan como total un valor de 300$, es decir, $10f_m + 15f_n + 40f_e = 300$.

   La segunda dicta que, de las ventas realizadas, la suma de las fotos normales (f_n) y enormes (f_e) vendidas es igual al número de fotos miniatura (f_m) vendidas. Es decir que para ese día, $f_m = f_n + f_e$ o $f_m - f_n - f_e = 0$.

   La tercera se refiere a que, durante ese día, se vendieron el doble de fotos de tamaño normal f_n que de tamaño enorme f_e, es decir, $f_n = 2f_e$ o $f_n - 2f_e = 0$

   Con esto, podemos establecer un sistema de ecuaciones de 3 variables y 3 ecuaciones:

$$10f_m + 15f_n + 40f_e = 300$$
$$f_m - f_n - f_e = 0$$
$$f_n - 2f_e = 0$$

el cual puede ser re-escrito como un sistema de ecuaciones matricial de la siguiente manera:

$$\begin{bmatrix} 10 & 15 & 40 \\ 1 & -1 & -1 \\ 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} f_m \\ f_n \\ f_e \end{bmatrix} = \begin{bmatrix} 300 \\ 0 \\ 0 \end{bmatrix}$$

b) Script de ejecución Con el fin de dar solución al problema, se creó una función y un script. La función, `triangularBSolver(A,B)`, al igual que `triangularASolver(A,B)`, realiza las mismas operaciones necesarias para realizar el despeje del vector de las variables indeterminadas con la diferencia de que realiza esto para una matriz de 3×3.

```
function output = triangularBSolver(A, B)
  [vert, hori] = size(A);
  [alt, anc] = size(B);

  if vert == hori && vert == alt && anc == 1
    [L, U] = luFact(A);

    y(1, 1) = B(1, 1);
    y(2, 1) = B(2, 1) - y(1, 1) * L(2, 1);
    y(3, 1) = B(3, 1) - y(1, 1) * L(3, 1) - y(2, 1) * L(3, 2);

    x(3, 1) = (y(3, 1)) / U(3, 3);
    x(2, 1) = (y(2, 1) - x(3, 1) * U(2, 3)) / U(2, 2);
    x(1, 1) = (y(1, 1) - x(3, 1) * U(1, 3) - x(2, 1) * U(1, 2)) / U(1, 1);

    output = x;
  end

end
```

De igual manera, se realizó el script `interpretingRunner.m`, el cual realiza la ejecución de la función `triangular-BSolver(A,B)` con la matriz y vector de valores respectivos para la solución del problema.

```
photos = [10 15 40; 1 -1 -1; 0 1 -2];
values = [300; 0; 0];

[L, U] = luFact(photos);

triangularBSolver(photos, values)
```

c) Solucionando el problema
Tras la ejecución en la consola, el script da la siguiente salida:

```
ans =

    9
    6
    3
```

Con esta salida, finalmente, se realiza la verificación de los resultados con la multiplicación de la matriz con los valores en `ans`, `photos*ans`, lo que da, como era de esperarse, lo siguientes valores:

```
ans =

   300
     0
     0
```

De esto, podemos determinar que la salida dada por `triangularBSolver(A,B)`, es correcta para la situación presentada y, por ende, el sistema de ecuaciones ha quedado resuelto.

$$\begin{bmatrix} 10 & 15 & 40 \\ 1 & -1 & -1 \\ 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} 9 \\ 6 \\ 3 \end{bmatrix} = \begin{bmatrix} 300 \\ 0 \\ 0 \end{bmatrix}$$

4. Proposición
   De manera final, se propone la creación de un ejercicio similar al anteriormente resuelto. De esto se planteó el siguiente problema.

   a) Problema propuesto
   En un restaurante, una familia le propone a un mesero que, de poder descifrar sus edades, estos le darán 100$. El mesero tras aceptar, recibe las siguientes pistas:

   1) La suma de las edades del padre, la madre y su hija es 70.
   2) La diferencia de las edades del padre y la hija es igual a la edad de la madre.
   3) La edad de la hija es 13 veces menor que la suma de las edades de sus padres.

   ¿Cuáles son las edades de los integrantes de la familia?

   b) Solución
   A partir de las pistas dadas, se puede desarrollar un sistema de ecuaciones el cual permitía calcular las edades de la familia. De la primera pista, sabemos que la suma de las edades es 70, es decir $P + M + H = 70$.
   De la segunda pista, podemos sabemos que la diferencia de las edades es la edad de la madre, es decir, $P - H = M$ o $P - H - M = 0$.
   Finalmente, de la tercera, sabemos que la edad de la hija, multiplicada por 13, es la la suma de la edad de los padres, es decir, $13H = P + M$ o $13H - P - M = 0$.
   Con esto, podemos desarrollar un sistema de ecuaciones el cual nos permite describir de manera apropiada el problema.

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & -1 \\ -1 & -1 & 13 \end{bmatrix} \begin{bmatrix} P \\ M \\ H \end{bmatrix} = \begin{bmatrix} 70 \\ 0 \\ 0 \end{bmatrix}$$

   Para resolver el problema, se creo el script `proposingRunner.m`, este script emplea el mismo algoritmo `triangularBSolver(A,B)` con el fin de dar solución al problema.

```
APro = [1 1 1; 1 -1 -1; -1 -1 13];
BPro = [70; 0; 0];

triangularBSolver(APro, BPro)
```

Tras la ejecución en la terminal, la función da la siguiente salida:

```
ans =

    35
    30
     5
```

De esto, podemos verificar el resultado obtenido tras realizar `APro*ans`, lo que nos da como salida:

```
ans =

    70
     0
     0
```

Confirmando los resultados de la ejecución para la solución del sistema de ecuaciones, y ganándole 100$ al mesero del restaurante.

3. Anexos

luFact.m

```
function [L, U] = luFact(inTrix)
  [vert, hori] = size(inTrix);
  L = NaN;
  U = NaN;
  counter = 1;

  if vert == hori
    U = inTrix;
    L = eye(vert, hori);

    for outer = 1:hori
      counter = counter + 1;

      temp = U(outer, :);
      tempFac = temp / temp(outer);

      for subber = counter:vert
        aux = tempFac * U(subber, outer);
        L(subber, outer) = aux(1, outer) / temp(outer);
        U(subber, :) = U(subber, :) - aux;
      end
    end
  end
end
```

triangularASolver.m

```
function output = triangularASolver(A, B)
  [vert, hori] = size(A);
  [alt, anc] = size(B);

  if vert == hori && vert == alt && anc == 1
    [L, U] = luFact(A);

    y(1, 1) = B(1, 1);
    y(2, 1) = B(2, 1) - y(1, 1) * L(2, 1);
    y(3, 1) = B(3, 1) - y(1, 1) * L(3, 1) - y(2, 1) * L(3, 2);
    y(4, 1) = B(4, 1) - y(1, 1) * L(4, 1) - y(2, 1) * L(4, 2) - y(3, 1) * L(4, 3);

    x(4, 1) = y(4, 1) / U(4, 4);
    x(3, 1) = (y(3, 1) - x(4, 1) * U(3, 4)) / U(3, 3);
    x(2, 1) = (y(2, 1) - x(4, 1) * U(2, 4) - x(3, 1) * U(2, 3)) / U(2, 2);
    x(1, 1) = (y(1, 1) - x(4, 1) * U(1, 4) - x(3, 1) * U(1, 3) - x(2, 1) * U(1, 2)) / U(1, 1);

    output = x;
  end

end
```

triangularBSolver.m

```
function output = triangularBSolver(A, B)
  [vert, hori] = size(A);
  [alt, anc] = size(B);
```

```
4
5   if vert == hori && vert == alt && anc == 1
6     [L, U] = luFact(A);
7
8     y(1, 1) = B(1, 1);
9     y(2, 1) = B(2, 1) - y(1, 1) * L(2, 1);
10    y(3, 1) = B(3, 1) - y(1, 1) * L(3, 1) - y(2, 1) * L(3, 2);
11
12    x(3, 1) = (y(3, 1)) / U(3, 3);
13    x(2, 1) = (y(2, 1) - x(3, 1) * U(2, 3)) / U(2, 2);
14    x(1, 1) = (y(1, 1) - x(3, 1) * U(1, 3) - x(2, 1) * U(1, 2)) / U(1, 1);
15
16    output = x;
17  end
18
19 end
```

triangRunner.m

```
1 A = [1 1 0 4; 2 -1 5 0; 5 2 1 2; -3 0 2 6];
2
3 [L, U] = luFact(A);
4
5 L
6 U
```

triangRunner2.m

```
1 A = [1 2 -3 4; 4 8 12 -8; 2 3 2 1; -3 -1 1 -4];
2
3 B = [3; 60; 1; 5];
4
5 triangularASolver(A, B)
```

triangRunner3.m

```
1 moddedA = [1 2 -3 4; 2 3 2 1; -3 -1 1 -4; 4 8 12 -8];
2
3 moddedB = [3; 1; 5; 60];
4
5 triangularASolver(moddedA, moddedB)
```

functionComparator.m

```
1 A = [3, 5, 6; 7, 7, 1; 3, 8, 9];
2
3 [mL, mU] = lu(A)
4 [L, U] = luFact(A)
```

interpretingRunner.m

```
1 photos = [10 15 40; 1 -1 -1; 0 1 -2];
2 values = [300; 0; 0];
3
4 [L, U] = luFact(photos);
5
6 triangularBSolver(photos, values)
```

proposingRunner.m

```
APro = [1 1 1; 1 -1 -1; 1 1 -13];
BPro = [70; 0; 0];

triangularBSolver(APro, BPro)
```