Problema 1

Implementar los algoritmos de *Backward* y *Forward* substitution.

Se implementaron los algoritmos de *Backward* y *Forward* substitution con las funciones **backward_subsitution** y **forward_subsitution** en el archivo **substitution.py**. Para cada uno de los algoritmos verificamos primero que las matrices recibidas sean triangular superior e inferior, respectivamente. También verificamos que los valores en la diagonal no sean cero para asegurar que existe una solución única.

Algorithm 1: Backward Substitution. Un extracto de la implementación donde se realiza la substitución para encontrar x.

```
x = np.zeros(n) # Initialize solution vector

for idx in range(n - 1, -1, -1): # Start loop backwards
    x[idx] = b[idx]
    for jdx in range(idx + 1, n):
        x[idx] -= x[jdx] * A[idx, jdx]
    x[idx] /= A[idx, idx]
```

Problema 2

Implementar el algoritmo de eliminación gaussiana con pivotteo parcial LUP, 21.1 del Trefethen (p. 160).

Al implementar el algritmo debemos verificar que se encuentra un pivote valido en cada iteración. Nos interesa el pivote más grande en valor absoluto. No podremos dar la factorización LUP con este algoritmo si todos los pivotes posibles son cero. El algoritmo fue implementado con la función **lup_factorization** que se encuentra en el archivo **factorization.py**.

Algorithm 2: Eliminación Gaussiana con pivoteo parcial. Un extracto del algoritmo donde se verifica si existen o no pivotes validos.

```
...
for kdx in range(0, n - 1):
    # Check if there are valid pivots.
    if len(U[kdx:, kdx]) - np.count_nonzero(U[kdx:, kdx]) > 0:
        print("Can't find a valid pivot.")
        return (-1, -1, -1)

    # Find pivot.
    idx_max = kdx + np.argmax(np.absolute(U[kdx:, kdx]))
...
```

Problema 3

Dar la descomposicion LUP para una matriz aleatoria de entradas $U(0,1)$ de tamaño 5 x 5, y para la matriz :

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 1 \\ -1 & 1 & 1 & 0 & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 \end{pmatrix}$$

La descomposicíon LUP de la matriz A definida anteriormente es :

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ -1 & 1 & 1 & 0 & 0 \\ -1 & -1 & -1 & 1 & 0 \\ -1 & -1 & -1 & 1 & 1 \end{bmatrix} U = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & 4 \\ 0 & 0 & 0 & 1 & 8 \\ 0 & 0 & 0 & 0 & 16 \end{bmatrix} P = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Para ver los resultados de este ejercicio ejecutar el archivo **factorization.py**.

Generamos una matriz aleatoria de entradas $U(0,1)$ de tamaño 5 x 5 utilizando la función **np.random.rand** del modulo numpy que está encapsulada dentro de **generate_random_matrix** que se encuentra el archivo **substitution.py**.

Nota: En los archivos **factorization.py** y **solve_system.py** se fijo la semilla por defecto igual a 0 para poder recrear los resultados presentados en este reporte. Se puede cambiar la semilla llamando al código de la forma ***python factorization.py semilla***.

$$B = \begin{pmatrix} 0{,}549 & 0{,}715 & 0{,}603 & 0{,}545 & 0{,}424 \\ 0{,}646 & 0{,}438 & 0{,}892 & 0{,}964 & 0{,}383 \\ 0{,}792 & 0{,}529 & 0{,}568 & 0{,}926 & 0{,}071 \\ 0{,}087 & 0{,}020 & 0{,}833 & 0{,}778 & 0{,}870 \\ 0{,}979 & 0{,}799 & 0{,}461 & 0{,}781 & 0{,}118 \end{pmatrix}$$

Y su descomposición LUP es :

$$L = \begin{bmatrix} 1{,}000 & 0 & 0 & 0 & 0 \\ 0{,}561 & 1{,}000 & 0 & 0 & 0 \\ 0{,}089 & -0{,}191 & 1{,}000 & 0 & 0 \\ 0{,}660 & -0{,}337 & 0{,}820 & 1{,}000 & 0 \\ 0{,}809 & -0{,}441 & 0{,}404 & -0{,}413 & 1{,}000 \end{bmatrix} U = \begin{bmatrix} 0{,}979 & 0{,}799 & 0{,}461 & 0{,}781 & 0{,}118 \\ 0 & 0{,}267 & 0{,}344 & 0{,}107 & 0{,}357 \\ 0 & 0 & 0{,}857 & 0{,}729 & 0{,}928 \\ 0 & 0 & 0 & -0{,}113 & -0{,}335 \\ 0 & 0 & 0 & 0 & -0{,}380 \end{bmatrix} P = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Problema 4

Usando la descomposición LUP anterior, resolver el sistema de la forma

$$\mathbf{D\ x = b}$$

donde D son las matrices del problema 3, para 5 diferentes matrices aleatorias con entradas $U(0,1)$. Verificando si es o no posible resolver el sistema.

Resolveremos los sistemas mencionados con la función **solve_system(A, b)** que se encuentra en el archivo **solve_system.py**.

Algorithm 3: Resolver sistema Ax = b

```
def solve_system(A, b):
    # Get LUP decomposition
    (L, U, P) = lup_decomposition(A)

    # Solve system Ly = Pb with y = Ux
    y = forward_substitution(L, P @ b)

    # Solve system Ux = y
    x = backward_substitution(U, y)

    return x
```

Para ver los resultados de este ejercicio ejecutar el archivo **solve_system.py**.

Utilizando las matrices A y B del problema anterior, generamos 5 sistemas de ecuaciones de la forma $Ax = b$ y $Bx = b$, para cinco b's aleatorios con entradas $U(0,1)$. Estos fueron los resultados:

	x **para** $Ax = b$	x **para** $Bx = b$
$b_1 = \begin{bmatrix} 0{,}640 \\ 0{,}143 \\ 0{,}945 \\ 0{,}522 \\ 0{,}415 \end{bmatrix}$	$x = \begin{bmatrix} 0{,}108 \\ -0{,}282 \\ 0{,}238 \\ 0{,}054 \\ 0{,}532 \end{bmatrix}$	$x = \begin{bmatrix} -7{,}441 \\ 5{,}426 \\ -4{,}162 \\ 6{,}914 \\ -0{,}982 \end{bmatrix}$
$b_2 = \begin{bmatrix} 0{,}265 \\ 0{,}774 \\ 0{,}456 \\ 0{,}568 \\ 0{,}019 \end{bmatrix}$	$x = \begin{bmatrix} -0{,}155 \\ 0{,}200 \\ 0{,}081 \\ 0{,}275 \\ 0{,}420 \end{bmatrix}$	$x = \begin{bmatrix} -1{,}170 \\ 0{,}268 \\ 1{,}610 \\ 0{,}442 \\ -1{,}172 \end{bmatrix}$
$b_3 = \begin{bmatrix} 0{,}618 \\ 0{,}612 \\ 0{,}617 \\ 0{,}944 \\ 0{,}682 \end{bmatrix}$	$x = \begin{bmatrix} -0{,}023 \\ -0{,}051 \\ -0{,}098 \\ 0{,}131 \\ 0{,}641 \end{bmatrix}$	$x = \begin{bmatrix} -0{,}283 \\ 0{,}462 \\ -1{,}233 \\ 1{,}316 \\ 1{,}106 \end{bmatrix}$
$b_4 = \begin{bmatrix} 0{,}360 \\ 0{,}437 \\ 0{,}698 \\ 0{,}060 \\ 0{,}667 \end{bmatrix}$	$x = \begin{bmatrix} -0{,}062 \\ -0{,}047 \\ 0{,}167 \\ -0{,}303 \\ 0{,}422 \end{bmatrix}$	$x = \begin{bmatrix} -0{,}589 \\ 0{,}775 \\ -0{,}615 \\ 1{,}222 \\ -0{,}394 \end{bmatrix}$
$b_5 = \begin{bmatrix} 0{,}671 \\ 0{,}210 \\ 0{,}129 \\ 0{,}315 \\ 0{,}364 \end{bmatrix}$	$x = \begin{bmatrix} 0{,}224 \\ -0{,}012 \\ -0{,}105 \\ -0{,}024 \\ 0{,}446 \end{bmatrix}$	$x = \begin{bmatrix} -0{,}775 \\ 1{,}314 \\ -0{,}209 \\ 0{,}142 \\ 0{,}482 \end{bmatrix}$

Problema 5

Implementar el algoritmo de descomposición de Cholesky 23.1 del Tre-fethen (p. 175).

Para implementar el algoritmo verificamos que la matriz sea simétrica, que las entradas en la diagonal no sean cero y en cada paso intermedio que el pivote que estamos utilizando no sea negativo. La implementación se encuentra en la función **cholesky_factorization** en el archivo **factorization.py**.

Algorithm 4: Algoritmo de Cholesky. Un extracto del código donde se verifica la posibilidad de completar el algoritmo sin errores.

```
...
for jdx in range(kdx + 1, n):
    R[jdx, jdx:] -= R[kdx, jdx:] * (R[kdx, jdx] / R[kdx, kdx])
```

```
    if R[kdx, kdx] < 0:
        print("Not an hermitian positive definite matrix.")
        return -1
...
```

Notemos que el mismo algoritmo de Cholesky es una buena prueba para verificar si una matriz es hermitiana positiva definida ó no.

Problema 6

Comparar la complejidad de su implementación de los algoritmos de factorización de Cholesky y LUP mediante la medición de los tiempos que tardan con respecto a la descomposición de una matriz aleatoria hermitiana definida positiva. Graficar la comparación.

Utilizamos el modulo **timeit** para medir los tiempos de ejecución de las funciones **lup_factorization** y **cholesky_factorization** (presentes en el archivo **factorization.py**). El calculo del tiempo se hace con la función **measure_factorization_time** que tiene un parámetro opcional llamado *number*. Este parámetro especifica la cantidad de veces que se repetirá la factorización y se regresará el promedio de los tiempos, esto para eliminar la variabilidad en los tiempos de ejecución del método.

La **Figura 1** representa representa la ejecución con el parḿetro por defecto $number = 1$ para matrices generadas de tamanõ n x n, para $n \in \{1, 2, ..., 2000\}$.

La **Figura 2** representa la ejecución con el parḿetro $number = 5$ para matrices generadas de tamaño n x n, para $n \in \{5, 10, 15, ..., 2000\}$. Notemos que en este caso se ejecuto 5 veces cada factorización y se calculo el promedio de los tiempos, obteniendo una curva más suave (aunque también se guardo la imagen en una diferente escala, por lo se ve más nítida).

Nota : Para las gráficas. Eje x: Tamaño del lado la matriz. Eje y: Tiempo de ejecución en segundos.

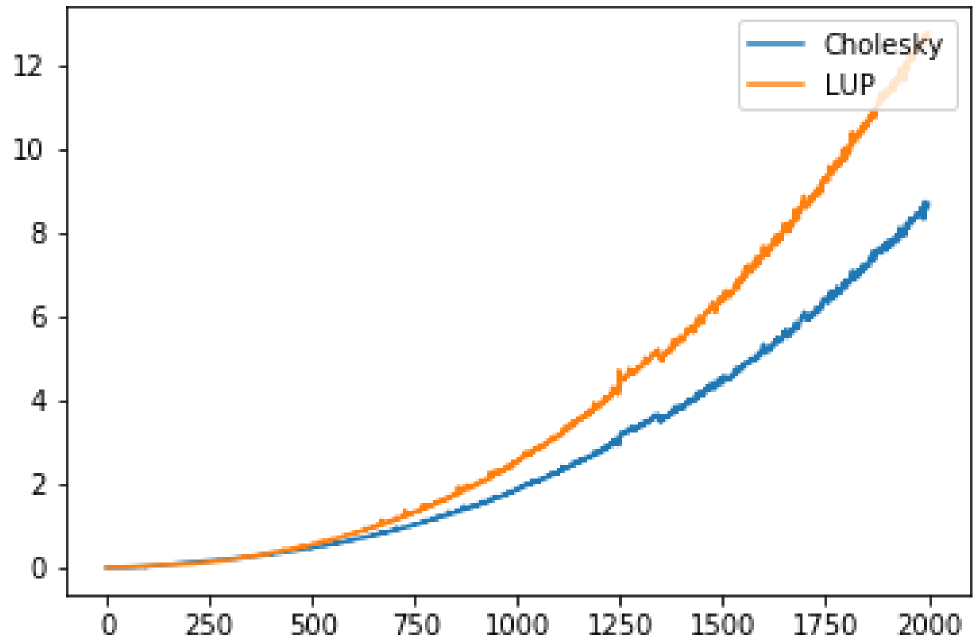

Figura 1: Comparación entre el tiempo de ejecución en segundos de los métodos de factorizacin LUP y Cholesky para matrices de distintos tamaños.

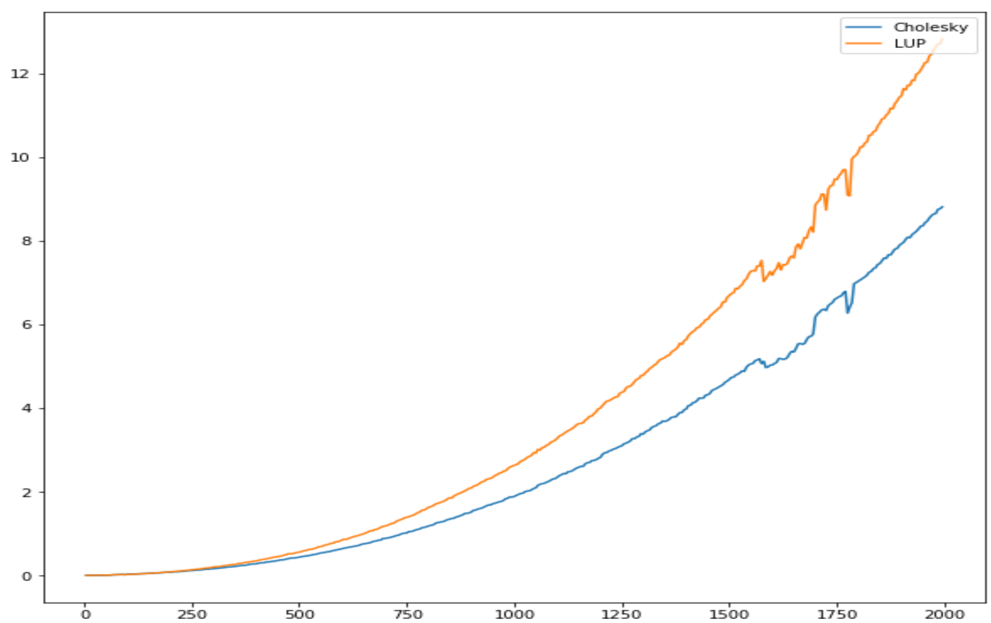

Figura 2: Comparación entre el tiempo de ejecución en segundos de los métodos de factorizacin LUP y Cholesky para matrices de distintos tamaños.