Problema 1

Sea $\mathbf{A}$ una matriz de tamaño 20×50. Creenla aleatoriamente, fíjenla y calculen su descomposición QR. Sean $\lambda_1 > \lambda_2 > \cdots \geq \lambda_{20} = 1 > 0$ y

$$B = Q^* diag(\lambda_1, \lambda_2, ..., \lambda_{20}) Q,$$
$$B_\epsilon = Q^* diag(\lambda_1 + \epsilon_1, \lambda_2 + \epsilon_2, ..., \lambda_{20} + \epsilon_{20}) Q, \quad con \quad \epsilon_i \sim \mathcal{N}(0, \sigma),$$

para $\sigma = 0.01\lambda_{20} = 0.01$.

- **Comparar la descomposición de Cholesky de B y de B_ϵ usando el algoritmo de la tarea 1. Considerar los casos cuando B tiene un buen número de condición y un mal número de condición. Con el caso mal condicionado, comparar el resultado de su algoritmo con el del algoritmo de Cholesky de scipy.**

  Para comparar las factorizaciones $\mathbf{B} = \mathbf{Q^T D Q} = \mathbf{L^T L}$ de Cholesky obtenidas de nuestros algoritmos, utilizaremos la siguiente medida del error

  $$\mathbf{E}(\mathbf{B}) = \|\mathbf{B} - \mathbf{L^T L}\| \tag{1}$$

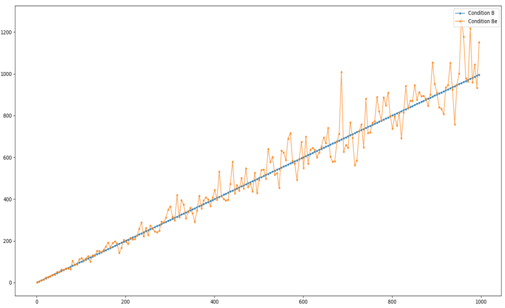
  Para generar distintos conjuntos de eigenvalores $\lambda_1 > \lambda_2 > \cdots \geq \lambda_{20} = 1 > 0$ de modo que podamos probar nuestros algoritmos con distintas matrices $\mathbf{B}, \mathbf{B}_\epsilon$ utilizaremos dos estrategias que explicaremos a continuación :

  - **Eigenvalores Equiespaciados** : Generamos un conjunto de valores equiespaciados entre un valor mínimo $\lambda_{20} = 1$ y un máximo $\lambda_1 = M_e$, de manera que los puntos (i, λ_i) forman una linea recta. La formula para esto es

    $$\lambda_i = 20 - i \cdot \left(\frac{M_e - 1}{20 - 1} \right) \tag{2}$$

  - **Uniformemente Distribuidos** : Fijamos el eigenvalor máximo como $\lambda_1 = M_e$ y el eigenvalor mínimo como $\lambda_{20} = 1$. Ahora consideramos los demas eigenvalores como una realización de las variables aleatorias con distribución uniforme $\lambda_2, \lambda_3, \ldots, \lambda_{19} \sim \mathbb{U}(\boldsymbol{\lambda_1}, \boldsymbol{\lambda_{20}})$.

  Ahora generaremos matrices con distintos números de condición al variar el eigenvalor máximo $\mathbf{M_e} \geq 1$ de manera lineal hasta llegar a un valor máximo $\mathbb{M}$. De manera que el número de condición de nuestra matriz $\mathbf{B}$ crecera linealmente con $\mathbf{M_e} = \frac{\boldsymbol{\lambda_1}}{\boldsymbol{\lambda_{20}}}$. Graficaremos a continuación el número de condición de las matrices B y de B_ϵ, tomando $\mathbb{M} = 1000$ como valor máximo, Equiespaciados



Uniformes

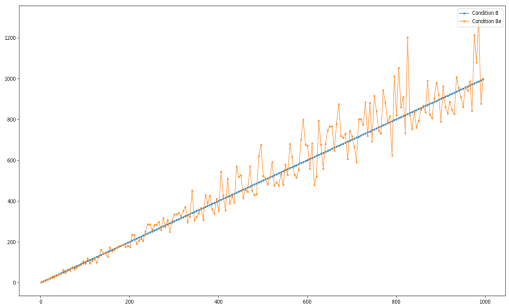
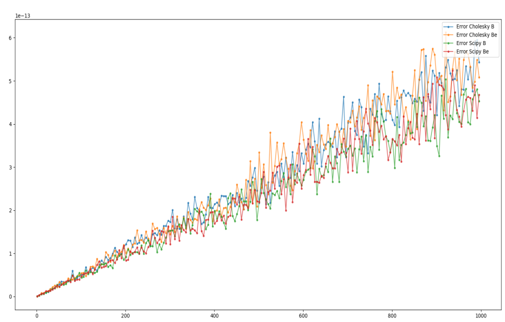


Figura 1: Número de condición : Graficamos el número de condición en el eje-y y el máximo eigenvalor M_e en el eje-x.

Notemos que el número condición de la matriz B_ϵ varia alrededor de la recta identidad, posiblemente es parecida a su media, puesto que su número de condición es de la forma $\frac{\lambda_1+\epsilon_1}{\lambda_{20}+\epsilon_{20}}$.

Ahora graficaremos la medición del error de la factorizaciones de Cholesky obtenidas, osea los valores de $\mathbf{E}(\mathbf{B}) = \|\mathbf{B} - \mathbf{L^T L}\|$ y $\mathbf{E}(\mathbf{B}_\epsilon) = \|\mathbf{B}_\epsilon - \mathbf{L}_\epsilon^\mathbf{T} \mathbf{L}_\epsilon\|$ para distintos valores de nuestro eigenvalor máximo $\mathbf{M_e}$. Compararemos el error de nuestras factorizaciones habiendo utilizado el algoritmo (Trefethen) que nosotros implementamos contra el de Scipy para la factorización de Cholesky.

Equiespaciados



Uniformes

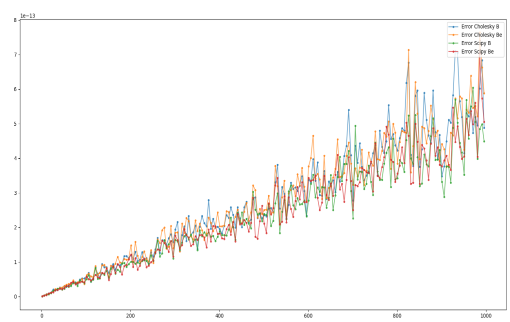


Figura 2: Medida del Error : Graficamos el número de condición en el eje-y y el máximo eigenvalor M_e en el eje-x.

Observamos una tendencia muy clara en la forma en que cambia el error al variar el número de condición de las matrices B y de B_ϵ. Primero notemos que el error es del orden de $E(B), E(B_e) \approx 1 \times 10^{-13}$. El error crece linealmente respecto al número de condición, aunque debemos siempre considerar que siguen siendo errores muy pequeños. También es claro que la varianza del error crece gradualmente con el número de condición.

Es interesante que la diferencia entre los errores observados obtenidos con el algoritmo de **Scipy** y el que nosotros implementamos es muy pequeña. Y también el error de estos dos algoritmos al calcular la factorización de $\mathbf{B}_\epsilon$ es muy similar.

Esto anterior puede ser un indicativo de que nuestro algoritmo es igual de estable que el algoritmo de **Scipy**. No se observo ninguna ganacia considerable respecto a trabajar con el eigenvalores sumandoles un coedificiente de ruido o no.

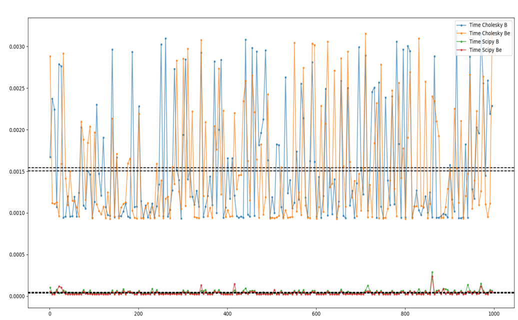
- **Medir el tiempo de ejecución de su algoritmo de Cholesky con el de scipy.**

  Antes de hablar de los tiempos de ejecución de nuestros experimentos, debemos notar el número de operaciones que nuestra implementación del algoritmo de **factorización de Cholesky** ejecuta solo depende del tamaño (m, m) de la matriz hermitiana positiva definida que se quiere factorizar. Por lo que la variabilidad en el tiempo de ejecución solo se puede atribuir al tiempo ejecución de las operaciones elementales en la computadora y en el propio procesador.

  Cualquier cambio que hagamos al número de condición de la matrices B y B_ϵ no debe afectar de ninguna manera el tiempo de ejecución de nuestro algoritmo pues las matrices siempre tienen tamaño 20×50.

  No conocemos el algoritmo específico de la implementación de **Scipy** de la factorización de Cholesky, pero a menos que se utilizen estrategias de pivoteo o un algoritmo no deterministico, no debe variar mucho su tiempo de ejecución por que el tamaño de la matriz es fijo.

Equiespaciados



Uniformes

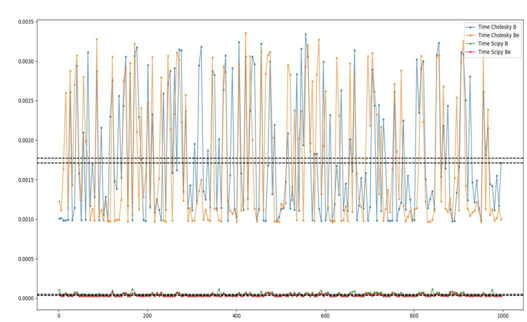


Figura 3: Tiempo de Ejecución : Graficamos el tiempo de ejecución para calcular la factorización y el máximo eigenvalor M_e en el eje-x.

Claramente el algoritmo de **Scipy** es mucho más rapido que nuestra implementación en python. Una cosa interesante es notar que el tiempo de ejecución de nuestro algoritmo está acotado inferiormente por $t = 0.0010$. Tambíen se graficaron las de los tiempos de ejecución para tener un punto de referencia.

Problema 2

Resolver el problema de mínimos cuadrados,

$$y = \mathbf{X}\beta + \epsilon, \quad \epsilon_i \sim \mathcal{N}(0, \sigma)$$

usando su implementación de la descomposición QR. β es de tamaño $n \times 1$ y $\mathbf{X}$ de tamaño $n \times d$.
Sean $d = 5$, $n = 20$, $\beta = (5, 4, 3, 2, 1)'$ y $\sigma = 0.15$.

- Construir $\mathbf{X}$ con entradas aleatorias $\mathcal{U}(0, 1)$ y similuar y. Encontrar $\hat{\beta}$ y compararlo con el obtenido $\hat{\beta}_p$ haciendo $\mathbf{X} + \mathbf{\Delta X}$, donde las entradas de donde las entradas de $\mathbf{\Delta X}$ son $\mathcal{N}(\mathbf{0}, \sigma = \mathbf{0.01})$. Comparar a su vez con $\hat{\beta}_c = ((\mathbf{X} + \hat{\beta})(\mathbf{X} + \hat{\beta}))^{-1}(\mathbf{X} + \hat{\beta})$ y usando el algoritmo genérico para invertir matrices **scipy.linalg.inv**.

Calculamos una estimación de β adicional considerando el algoritmo de estimación de mínimos cuadrados utilizando la factorización **QR** de **Scipy** que llamaremos $\hat{\beta}_q$.

A continuación presentamos las estimaciones que obtuvimos cuando los números de condición de las matrices $\mathbf{X}$ y $\mathbf{X} + \mathbf{\Delta X}$ fueron

	$\mathbf{X}$	$\mathbf{X} + \mathbf{\Delta X}$
Número de condición	5.8471	5.7254

siendo las estimaciones,

Estimación	**Error** $\lVert\beta - \hat{\beta}\rVert$
$\hat{\beta} = \begin{bmatrix} 5.06727 & 4.11598 & 2.98676 & 1.90551 & 0.97445 \end{bmatrix}$	0.167
$\hat{\beta}_p = \begin{bmatrix} 5.09721 & 4.06898 & 2.99652 & 1.90551 & 0.97445 \end{bmatrix}$	0.150
$\hat{\beta}_c = \begin{bmatrix} 5.09721 & 4.06898 & 2.99652 & 1.90551 & 0.97445 \end{bmatrix}$	0.150
$\hat{\beta}_q = \begin{bmatrix} 5.09721 & 4.06898 & 2.99652 & 1.90551 & 0.97445 \end{bmatrix}$	0.150

Notemos que las estimaciones del problema $y = (\mathbf{X} + \mathbf{\Delta X})\,\beta + \epsilon$ obtenidas en base a los distintos algoritmos de mínimos cuadrados son iguales. Esto habla bien de las matrices

$\mathbf{X}$ y $\mathbf{X} + \mathbf{\Delta X}$ que al tener un número de condición relativamente bajo no acumulan errores durante la estimación.

Otra cosa que nos ayuda a verificar que tan buena es la estimiación es el **error** $\|\beta - \hat{\beta}\|$. Podemos comparar los resultados de diferentes estimadores viendo que tan bueno fue el ajuste de la regresión.

- Lo mismo que el anterior pero con $\mathbf{X}$ mal condicionada (ie. con casi colinealidad).

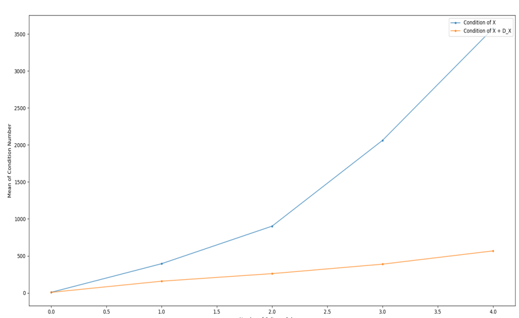
  Ahora transformaremos a la matriz $\mathbf{X}$ de manera que sea casi colineal. Para esto definiremos un parametro θ que determine el número de columnas en $\mathbf{X}$ que se transformaran a casi colineales a la primera columna. Esto se implemento de la siguiente manera en python,

Algorithm 1: Fijar número de columnas colineales. Del archivo **least_squares.py**.

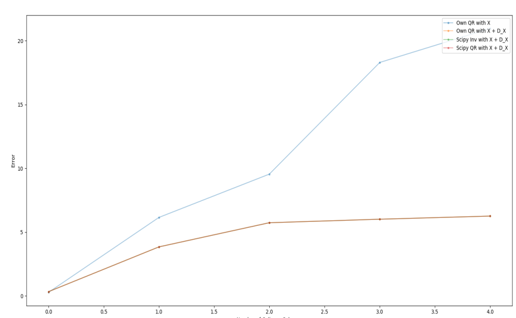
```
# Generate a random matrix
X = generate_random_matrix(dimension)

# Set theta columns to be colinear
for idx in range(1, min(1 + theta, 5)):
    X[:, idx] = (idx + 1) * X[:, 0] \
        + np.random.normal(loc=0.0, scale=.01, size=n)
```

Así podemos agregar diferentes grados de colinealidad a la matriz para estudiar como cambia el número de condición y los errores de las estimaciones de mínimos cuadrados. A continuación graficamos el promedio (habiendo hecho varias repeticiones con diferentes matrices X) del número de condición y del error de las estimaciones para los diferentes niveles de colinealidad usando el parámetro $\theta = 0, 1, 2, 3, 4$.

Número de Condición



Error $\|\beta - \hat{\beta}\|$



Observamos como se dispara el número de condición entre más columnas casi colineales agregamos. También el error de estimación se hace más grande, en particular el que usa nuestro algoritmo de **factorización QR**. Es dificil de notar, pero las gráficas de los otros

3 algoritmos son iguales y están una encima de otra. Lo que es muy interesante es que la condición y el error de estimación crece mucho más lentamente cuando trabajamos como las matriz $\mathbf{X} + \mathbf{\Delta X}$, puesto que a pesar de agregar colinealidad a las matrices, agregar un termino de ruido reduce considerablemente la condición de la matriz.

Presentaremos la estimación obtenida para una matriz muy mal condicionada al agregar 4 columnas casi colineales.

	$\mathbf{X}$	$\mathbf{X} + \mathbf{\Delta X}$
Número de condición	4773.2742	544.2308

Estimación	**Error** $\|\beta - \hat{\beta}\|$
$\hat{\beta} = \begin{bmatrix} 45.49929 & 1.68186 & 2.06297 & -3.99335 & -0.80989 \end{bmatrix}$	41.057
$\hat{\beta} = \begin{bmatrix} 4.37220 & 2.24914 & 4.06671 & 0.22307 & 2.61135 \end{bmatrix}$	3.217
$\hat{\beta} = \begin{bmatrix} 4.37220 & 2.24914 & 4.06671 & 0.22307 & 2.61135 \end{bmatrix}$	3.217
$\hat{\beta} = \begin{bmatrix} 4.37220 & 2.24914 & 4.06671 & 0.22307 & 2.61135 \end{bmatrix}$	3.217

Podemos observar claramente que es una estimación muy mala del primer algoritmo y menos mala de los otros 3 algoritmos que siempre arrojan los mismos resultados. De las observaciones pasadas y está última estimación, podemos asegurar que al agregar ruido cuando trabajamos con la matriz $\mathbf{X} + \mathbf{\Delta X}$ nos arroja mejores resultados siendo más estable. Esto anterior está ligado directamente con el número de condición de la matriz.