Cómputo Científico para Probabilidad, Estadística y Ciencia de Datos

Ezau Faridh Torres Torres

TAREA 3: Estabilidad.

Fecha de entrega: 18/Sep/2024.

NOTA: Los ejercicios se encuentran repartidos en los archivos:

- ejercicio1.py
- ejercicio2.py

---

1. Sea Q una matriz unitaria aleatoria de 20×20 (eg. con A una matriz de tamaño 20×20 aleatoria, calculen su descomposición QR). Sean $\lambda_1 > \lambda_2 > \cdots \geq \lambda_{20} = 1 > 0$ y

$$B = Q^* diag\left(\lambda_1, \lambda_2, \ldots, \lambda_{20}\right) Q, \tag{1}$$

$$B_\varepsilon = Q^* diag\left(\lambda_1 + \varepsilon_1, \lambda_2 + \varepsilon_2, \ldots, \lambda_{20} + \varepsilon_{20}\right) Q, \tag{2}$$

con $\varepsilon_i \sim N(0, \sigma)$ y $\sigma = 0.025$.

   a) 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.

   b) Con el caso mal condicionado, comparar el resultado de su algoritmo con el del algoritmo de Cholesky de scipy.

   c) Medir el tiempo de ejecución de su algoritmo de Cholesky con el de scipy.

Respuesta:

Se sabe que, dada una matriz A aleatoria de $n \times n$, entonces su factorización QR genera una matriz Q unitaria de $n \times n$. En el archivo ejercicio1.py, se genera la matriz Q de esta forma en la función *generar_Bs()*, la cual toma como argumentos al eigenvalor más grande de B : λ_1, al más pequeño: $\lambda_{20} = 1$, la desviación estándar del ruido: $\sigma = 0.025$ y la dimensión: $n = 20$. Tal función regresa a las matrices B y B_ε como en (1) y (2).

En clase se revisó que, para una matriz cuadrada A de $n \times n$ con eigenvalores $\{\lambda_1 \geq \cdots \geq \lambda_n\}$, su norma 2 cumple que $\|A\|_2 = |\lambda_1|$, y por lo tanto, el número de condición bajo la norma 2 está dado por

$$\kappa_2\left(A\right) = \left|\frac{\lambda_1}{\lambda_n}\right|. \tag{3}$$

Esto se usa en el archivo ejercicio1.py para generar B bien o mal condicionada. Se usa un valor de λ_1 bastante grande para mal condicionar la matriz, y uno relativamente cerca de $\lambda_{20} = 1$ para que esté bien condicionada.

(a)

Para el caso en que la matriz B esté bien condicionada, se usó $\lambda_1 = 2$, i.e., $\kappa_2 = 2$ y para cuando B esté mal condicionada, se usó $\lambda_1 = 1,000,000,000$, entonces $\kappa_2 = 1,000,000,000$.

Se ejecutó la función *LU_cholesky()* para ambas matrices B y B_ε para los casos bien y mal condicionados y los resultados fueron los esperados: para los 2 casos, las matrices de Cholesky R y R_ε eran tal que $R^TR = B$ y $R_\varepsilon^T R_\varepsilon = B_\varepsilon$, esto se verificó revisando que la normas $\left\|R^TR - B\right\|_2$ y $\left\|R_\varepsilon^T R_\varepsilon - B_\varepsilon\right\|_2$ fueran casi cero ($< 10^{-6}$). El resultado es esperado ya que, sin importar la magnitud del número de condición, se tiene que las matrices involucradas tienen eigenvalores positivos (dados por nosotros), implicando que sean definidas positivas y además, como B y B_ε son de la forma (1) y (2), también son simétricas, así que la factorización de Cholesky siempre funcionará bien.

(b)

Se usó la función *linalg.cholesky()* de la librería *Scipy* para calcular las matrices de Cholesky R_c y R_{ε_c} para el caso mal condicionado y se compararon con los resultados del inciso a). Se calcularon las normas $\|R_c - R\|_2$ y $\|R_{\varepsilon_c} - R_c\|_2$ y se verificó que fueran prácticamente cero ($< 10^{-6}$)

(c)

Se realizaron 1000 simulaciones de B y B_ε como en (1) y (2) en las que se midieron los tiempos de ejecución de la función *LU_cholesky()* para comparar con los de *linalg.cholesky()* de *Scipy*. Tales resultados se promediaron y se obtuvieron los siguientes resultados (variaba en cada repetición, sin embargo, el comportamiento es el mismo):

- Tiempo promedio LU_cholesky(B): 0.0002543556903979633.
- Tiempo promedio scipy.cholesky(B): 4.450875938346144^{-6}.
- Tiempo promedio LU_cholesky(B_ε): 0.0002526311719484511.
- Tiempo promedio scipy.cholesky(B_ε): 3.492367010039743^{-6}.

De aquí se puede notar que el método implementado en Scipy siempre es más rápido, pero con resultados igual de satisfactorios.

---

2. Resolver el problema de mínimos cuadrados,

$$y = X\beta + \varepsilon_i, \quad \varepsilon_i \sim N(0, \sigma) \tag{4}$$

usando su implementación de la descomposición QR; β es de tamaño $n \times 1$ y X de tamaño $n \times d$. Sean $d = 5$, $n = 20$, $\beta = (5, 4, 3, 2, 1)'$ y $\sigma = 0.12$.

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

b) Lo mismo que el anterior pero con X mal condicionada (i.e. con casi colinealidad).

Respuesta:

(a)

En el archivo ejercicio2.py, se implementa la función *ajuste_QR()*, la cual toma como argumentos a una matriz X de $n \times d$, el vector β de $d \times 1$ y la desviación estándar del ruido $\sigma = 0.12$. En esta función se generan los valores de y con ruido como en (4) y luego se toma

la factorización QR de X con ayuda de la función *MODIFIED_GRAM_SCHMIDT()* de la tarea 2. Con esto, se resuelve el sistema

$$R\beta = Q^T y \tag{5}$$

con backward substitution ya que R es una matriz triangular superior. Esta función da como salida a los coeficientes ajustados $\hat{\beta}$ que son solución a (5).

Se calculan $\hat{\beta}$ para X, $\hat{\beta}_p$ para $\tilde{X} = X + \Delta X$ y $\hat{\beta}_c = \left(\tilde{X}'\tilde{X}\right)^{-1}\tilde{X}'y$. Para una realización, se obtuvieron los siguientes resultados (se usó una semilla para reproducibilidad):

- β del modelo original: $(5, 4, 3, 2, 1)'$.
- $\hat{\beta}$ ajustados: $(4.840, 4.097, 3.201, 1.880, 1.025)'$.
- $\hat{\beta}_p$ ajustados y con perturbación: $(4.936, 3.966, 3.037, 1.957, 1.081)'$.
- $\hat{\beta}_c$ ajustados con mínimos cuadrados: $(4.992, 3.998, 2.961, 1.986, 0.997)'$.

El experimento se repitió 1000 veces y en cada una de estas, se midieron las normas: $\left\|\beta - \hat{\beta}\right\|_2$, $\left\|\beta - \hat{\beta}_p\right\|_2$ y $\left\|\beta - \hat{\beta}_c\right\|_2$ para revisar el error y finalmente tomar el promedio, resultando en:

- β vs $\hat{\beta}$: 0.19003.
- β vs $\hat{\beta}_p$: 0.19482.
- β vs $\hat{\beta}_c$: 0.11831.

Por lo que el que tuvo mejor rendimiento, fue el estimador $\hat{\beta}_c = \left(\tilde{X}'\tilde{X}\right)^{-1}\tilde{X}'y$.

(b)

Se generó una matriz X mal condicionada (con casi colinealidad). La forma en que se generó fue: se creó un vector aleatorio de tamaño $(n = 20) \times 1$, el cual será la primera columna de X. El resto de las columnas se generó a raíz de la primera, se le sumó un múltiplo muy pequeño de otro vector aleatorio. Con esto, se obtuvo la matriz X con casi colinealidad en sus columnas y con número de condición $\kappa_2 = 532,004.6$

Se repitió todo el análisis del inciso anterior (las 1000 simulaciones) y se obtuvieron los resultados para la última realización:

- β del modelo original: $(5, 4, 3, 2, 1)'$.
- $\hat{\beta}$ ajustados: $(-905.614, 129.147, -3831.882, 1039.486, 3583.851)'$.
- $\hat{\beta}_p$ ajustados y con perturbación: $(0.816, 1.463, 8.534, 3.564, 0.679)'$.
- $\hat{\beta}_c$ ajustados con mínimos cuadrados: $(2.573, 2.929, 3.809, 2.211, 3.458)'$.

Promediando las normas de interés, se obtuvo que:

- β vs $\hat{\beta}$: 7164.56341.
- β vs $\hat{\beta}_p$: 5.89879.
- β vs $\hat{\beta}_c$: 4.55228.

Notamos que, luego de 1000 iteraciones, el hecho de que X esté mal condicionada afectó de desempeño de los estimadores. En particular, el de peor error fue $\hat{\beta}$ y el "mejor" fue, de nuevo, $\hat{\beta}_c = \left(\tilde{X}'\tilde{X}\right)^{-1}\tilde{X}'y$.