Tarea 3
Estabilidad

Joel Chacón Castillo
Cómputo Científico

11 de septiembre de 2019

1. Formulas y repaso

El número de condición es quel métrica que mide un grado de cambio que puede sufrir una función $f(x)$ ante perturbaciones en el dominio $f(x + \Delta x)$. Esto se utiliza para medir la sensibilidad de una función. Un problema con un número de condición bajo es definido como *Un problema bien condicionado*, mientras un problema con un número de condición elevado es definido como *Un problema mal condicionado*. Una regla de aproximación popularmente utilizada es que si el número de condición es $k(A) = 10^k$, entonces se puede decir que se pueden perder k dígitos de exactitud. Dada las complicaciones para calcular el número de condición teórico, en la literatura se ha propuesto una cota superior.

Dada una matriz A, no singular se puede calcular el número de condición de la siguiente forma:

$$k(A) = ||A^{-1}||||A|| \geq ||A^{-1}A|| = 1 \tag{1}$$

El número de condición considerando la norma $||.||_2$ es la siguiente

$$k(A) = \frac{\sigma_{max}(A)}{\sigma_{min}(A)}, \tag{2}$$

donde $\sigma_{max}(A)$ y $\sigma_{min}(A)$ son el máximo y mínimo valores singulares de A respectivamente. Donde si A es unitario entonces $k(A) = 1$

2. Condicionamiento y Cholesky

Sea A una matriz aleatoria de tamaño 20×50, calcular su descomposición QR. Sean $\lambda_1 > \lambda_2 > ... \geq \lambda_{20} = 1 > 0$ y $B = Q^* diag(\lambda_1, \lambda_2, .., \lambda_{20})Q$ y $B_\epsilon = Q^* diag(\lambda_1 + \epsilon_1, \lambda_2 + \epsilon_2), ..., \lambda_{20} + \epsilon_{20})Q$, con $\epsilon_i \sim N(0, \sigma)$, con $\sigma = 0{,}01\lambda_{20} = 0{,}01$.

1. 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.

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

3. Medir el tiempo de ejecución de su algoritmo de Cholesky con el scipy.

Comentarios

Debido a que la matriz aleatoria A es de dimension 20×50 se tiene que aplicar un paso adicional al algoritmo de descomposición QR de la tarea anterior pues en este último se considera $m > n$, pero en este caso se tiene $m < n$, donde m y n corresponden al número de filas y columnas respectivamente. El procedimiento de factorización QR implica generar una base Q de la matriz A, la dimensión de Q debe corresponder al rango de A (se considera la forma reducida), así dada una matriz A linealmente independiente por filas o columnas dependiendo del caso el rango es $Rank(A) = min(m, n)$. Lo anterior implica que Q debe ser $Q \in \mathbb{C}^{20\times 20}$ y $R \in \mathbb{C}^{20\times 50}$.

En el algoritmo implementado se genera una matrix $Q \in \mathbb{C}^{20\times 50}$ y $R \in \mathbb{C}^{50\times 50}$. El paso extra consiste en únicamente tomar las primeras 20 columnas de Q, el resto 30 columnas son linealmente dependientes (o colineales a las primeras 30 o no forman un conjunto generador).

Caso con un buen número de condición: Partiendo de la base Q y dados valores propios $\lambda_i = 21 - i = \{1, ..., 20\}$, en resultado se tendría un valor de condición de $\frac{20}{1} = 20$, es decir, el mayor valor de λ_i corresponde al número de condición en este caso.

Caso con un mal número de condición: Para generar una matriz con un mal número de condición se parte de una matriz Q y valores propios $\lambda_i = 1 \times 10^{10} \times 0{,}5 \times (21 - i) = 1 \times 10^{10} \times 0{,}5 \times \{1, ..., 20\}$

En la tabla 1 se presentan los resultados del caso bien condicionado y del mal condicionado. Se consideran dos matrices: B que corresponde a una matriz simétrica (positiva definida) y R que corresponde a la factorización aplicada a B. En la tabla 1 se presenta el número de condición de B y de R. También se muestra el número de condición de B y R después de aplicar una perturbación. A continuación se enlistan algunas observaciones relevantes:

- En el caso bien condicionado aún existe un problema numérico ya que aunque el problema tiene un número de condición teórico de $k(B) = 20$ en la práctica se otiene 19,99999. La fila *Difference* indica la diferencia entre el número de condición obtenido despues de la perturbación. Principalmente en el problema mal condicionado existe un incremento significativo 56333202 en comparación al caso bien condicionado 0.103292160193934.

- El número de condición de la matríz R (después de la factorización Cholesky) es igual (o muy similar) al de factorización con Cholesky de Scipy.

- Los tiempos de la implementación con Scipy son menores en comparación a la implementación.

- En los casos bien condicionados los tiempos de la propuesta no son significativamente distintos, pero en el caso mal condicionado si se observa un cambio entre factorizar la matriz sin ruido y con ruido 0,00127696990967 y 0,00130081176758.

En la figura 1 se grafica el tiempo requerido para realizar la factorización de Cholesky con matrices de distintos números de condición, recordando que en este caso se puede cambiar el número de condición cambiando el mayor eigenvalor y manteniendo el menor eigenvalor en uno. Además se presentan los tiempos de la implementación con Cholesky y de de implementación de Scipy con y sin ruido. Principalmente se observa que el Cholesky de Scipy requiere un menor tiempo. En los dos casos se observa que conforme aumenta el número de condición el tiempo de éstos métodos es relativamente constante.

En la figura 2 se presenta la diferencia entre el número de condición entre la matriz B y la matriz perturbada B_ϵ. Esta diferencia indica el efecto de aplicar la misma perturbación a los mismos valores propios y a la matriz base Q. Esto indica el efecto de incrementar el número de condición y las perturbaciones, es decir, las mismas perturbaciones tienen un efecto lineal, este ejemplo es sólo para propósitos didácticos.

	Well-conditioned	**Ill-conditioned**
Condition B	19.9999	10000000000
Condition B_epsilon	20.1032921601939	10056333202.534
Difference	-0.103292160193934	-56333202.533947
Condition R	2.0581349647302	32204.2263956753
Condition R_epsilon	2.06233469477467	32294.8131714291
Time Cholesky B	0.00158286094666	0.00127696990967
Time Cholesky B_epsilon	0.00152707099915	0.00130081176758
Condition R Scipy	2.058134964730204	32204.210955494193
Condition R_epsilon Scipy	2.062334694774668	32294.811578955512
Time Cholesky B Scipy	0.0001220703125	3.60012054443e-05
Time Cholesky B_epsilon Scipy	3.09944152832e-05	3.00407409668e-05

Cuadro 1: Todas las condiguraciones donde se considera tanto el caso bien condicionado como el mal condicionado.

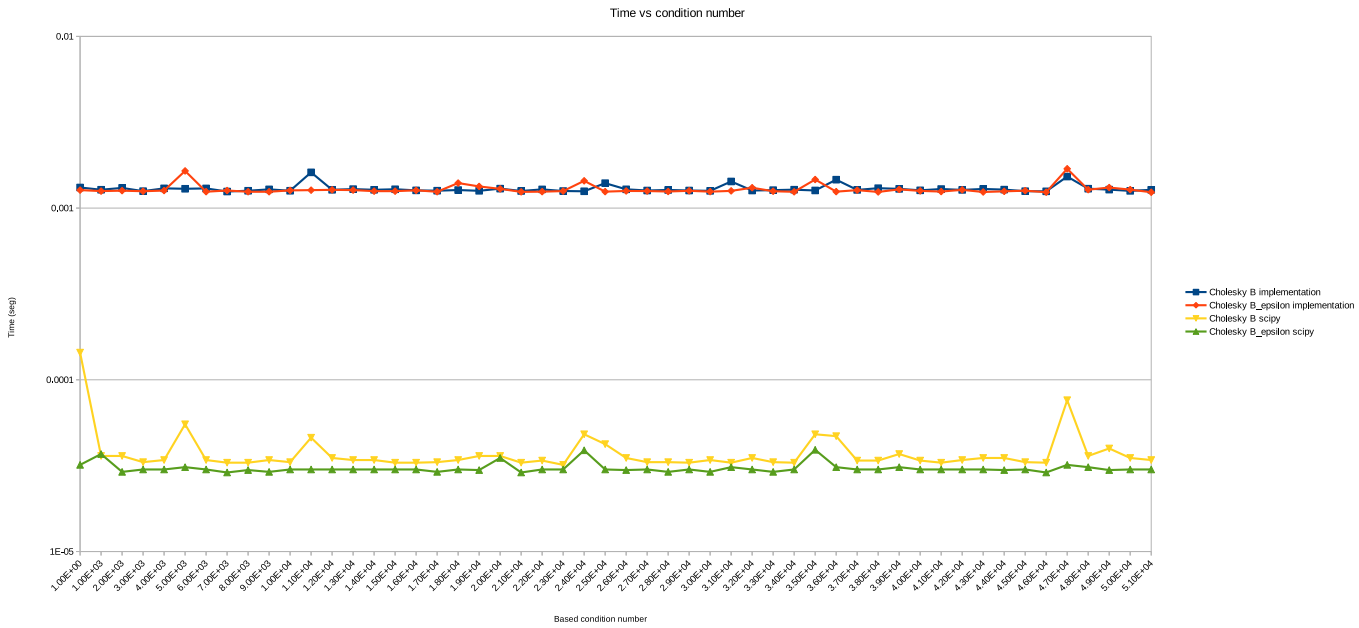

Figura 1: Tiempo de aplicar la factorización Cholesky a la misma matriz base Q con distintos vectores propios máximos (el mínimo es uno).

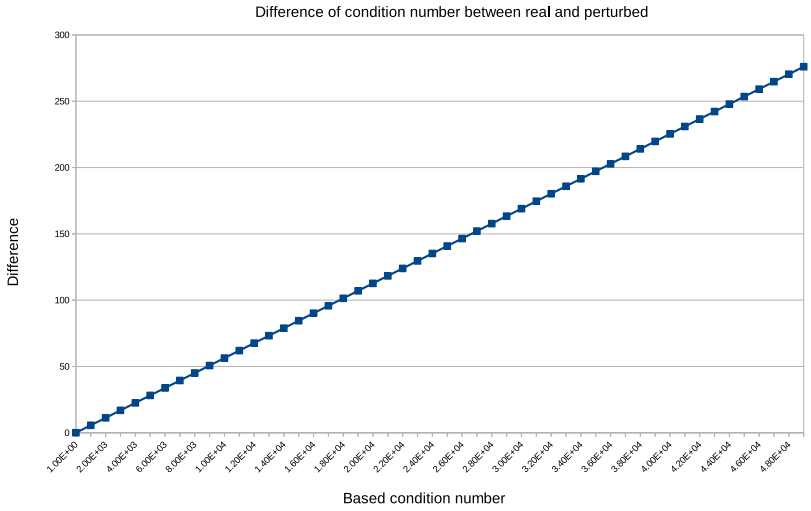

Figura 2: Tiempo de aplicar la factorización Cholesky a la misma matriz base Q con distintos vectores propios máximos (el mínimo es uno).

3. Condicionamiento y Mínimos cuadrados

Resolver el problema de mínimos cuadrados

$$y = X\beta + \epsilon, \quad \epsilon_i \sim N(0, \sigma) \tag{3}$$

usando su implementación de la decomposición QR, β es de tamaño $d \times 1$ y X de tamaño $n \times d$. Sean $d = 5$, $n = 20$, $\beta = (5, 4, 3, 2, 1)^*$ y $\sigma = 0{,}15$

1. 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 $N(0, 0{,}01)$. 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 scipu.linalg.inv.

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

Problema bien condicionado

En este punto se desea observar empíricamente el efecto que tiene aplicar una perturbación a una matriz bien condicionada, en este caso la perturbación se basa en una distribución normal con parámetros $N(0{,}0, 0{,}01)$. En la tabla 2 se presentan los valores obtenidos al resovler el probelma de mínimos cuadrados, lo más importante es ver que el valor de $\hat{\beta}$ no cambia de forma significativa al aplicar la perturbación, además se ve en el número de condición de la matriz X. En este caso utilizar el algoritmo genérico dió los mismos resultados que en la implementación.

		Condition number
β	{1,2,3,4,5}	
$\hat{\beta}$	{ 4.83356395, 3.86257303, 2.91181074, 2.04939016, 1.27259595}	6.2610
$\hat{\beta}_p$	{4.78005588, 3.91084359, 2.91956899, 2.02357267, 1.30699658 }	6.2007
$\hat{\beta}_c$	{4.78005588, 3.91084359, 2.91956899, 2.02357267, 1.30699658}	6.2007

Cuadro 2: Vector β estimado con una matriz X bien condicionada.

Problema mal condicionado

Para generar un problema mal condicionado se tomó la matriz X y se duplicó una columna mas una perturbación mínima: $X[:, 2] = X[:, 1] + U(0, 1 \times 10^{15})$. En este caso (tabla 3) pasa que al tener casi colinealidad entre dos columnas el número de condición es muy elevado, sin embargo al aplicar la perturbación la matriz se disminuye dramáticamente el número de condición de 4579227,131014605 a 151,46192194509328. Esto quiere decir que al tener una matriz mal condicionada resulta en otra más estable al agregar una perturbación a las soluciones. Esta estrategia de aplicar una perturbación a los datos se ha visto en muchas áreaa como realizar el cálculo del *Convex-hull* con puntos colineales, o en el caso de aplicar *Suppor Vector Machine SVM*. Además en la tabla 3 se observa que utilizar el algoritmo genérico de scipy aún produce resultados iguales $\hat{\beta}_p = \hat{\beta}_c$.

		Condition number
β	{1,2,3,4,5}	
$\hat{\beta}$	{-1.15195050e+05, 1.15204061e+05, 2.96864264e+00, 2.08947262e+00, 8.50706221e-01}	4579227.131014605
$\hat{\beta}_p$	{3.50326971, 5.49219943, 3.06949281, 2.00797932, 0.92492514}	151.46192194509328
$\hat{\beta}_c$	{3.50326971, 5.49219943, 3.06949281, 2.00797932, 0.92492514}	151.46192194509328

Cuadro 3: Vector β estimado con una matriz X mal condicionada.

Se hizo otro experimento donde se considera el caso mal condicionado y una perturbación $\Delta X \sim N(0, 0{,}0001)$. Los resultados se muestran en la tabla 4, principalmente se observa que implementar el algoritmo genérico para invertir matrices de scipy genera una diferencia, es decir, $\hat{\beta}_p \neq \hat{\beta}_c$. Posiblemente este problema se debe al mal condicionamiento de la matriz, además se calcula la inversa de ésta matriz, es bien sabido que calcular la inversa de una matriz es un proceso bastante sensible y puede provocar errores numéricos, principalmente no es *Backward-stable* a diferencia de Cholesky que sí es *Backward-stable*.

		Condition number
β	{1,2,3,4,5}	
$\hat{\beta}$	{-1.15195050e+05, 1.15204061e+05, 2.96864264e+00, 2.08947262e+00, 8.50706221e-01}	4891570186505504.0
$\hat{\beta}_p$	{119.71388581, -110.71297472, 3.02680463, 2.02684214, 0.9588729}	15066.775025727575
$\hat{\beta}_c$	{119.71388452, -110.71297347, 3.02680463, 2.02684214, 0.9588729}	15066.775025727575

Cuadro 4: Vector β estimado con una matriz X mal condicionada.