eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

TAREA 4

EJERCICIOS

Esta tarea tiene por objeto desarrollar con ayuda de S-PLUS (o R) los cálculos necesarios para la estimación de algunos parámetros sobre unos datos sintéticos, generados por una estructura conocida.

1. Genera, del modo que quieras, una matriz X de regresores de dimensión 15×3, con una columna de "unos". Lo único que debes evitar es que sea de rango deficiente o casi deficiente.

2. Genera el vector respuesta $\vec{Y}$ como combinación lineal de las columnas de X con parámetros β_i escogidos a tu antojo más una perturbación aleatoria ϵ. Observa que has generado $\vec{Y}$ precisamente del modo que supone la teoría del modelo de regresión lineal, y con parámetros de valores conocidos. Tus datos son sintéticos, y te dan la oportunidad de comparar las estimaciones con los verdaderos valores de los parámetros —algo que no podemos hacer en la práctica—.

   Con los datos generados en el apartado anterior,

   a) Calcula los estimadores mínimo-cuadráticos de los parámetros, $\hat{\beta}$. Si lo has hecho bien, y la varianza especificada para la perturbación no es muy grande, no debieran separarse demasiado de los $\vec{\beta}$ empleados en el apartado anterior.

   b) Estima la varianza de la perturbación. Compara con la real.

   c) Estima la matriz de covarianzas de los estimadores $\hat{\beta}$. ¿Es diagonal? ¿En que casos lo sería?

   d) Estima la matriz de covarianzas de los residuos. ¿Es diagonal? ¿Es de rango completo? ¿Podría serlo? Explica.

   e) Utiliza la función `lsfit`. Familiarízate con su output —algunas de cuyos componentes, como la descomposición `qr`, no necesitan inquietarte por el momento— y compara los resultados con los obtenidos antes. Debieran coincidir. Observa que la función `lsfit` añade ella la columna de "unos"; como ya figura entre tus regresores, has de invocar `lsfit` con el argumento `intercept=F`. Observa también que hay pequeñas diferencias entre los argumentos admisibles en S-PLUS y en R.

3. Repite la estimación en el apartado anterior imponiendo como condición que la suma de los estimadores de los parámetros coincida con la suma de los verdaderos parámetros empleados en 2) en la generación de la muestra artificial. Comprueba que los estimadores verifican la restricción impuesta.

4. Ahora que ya sabes como se hace, repite 100 veces la estimación anterior (con y sin restricciones) generando un vector $\vec{Y}$ diferente cada vez (con distintas perturbaciones, pero idénticos parámetros). Guarda las estimaciones de los parámetros. Mira las indicaciones en las Ayudas sobre como hacer cálculos repetitivos (¡no se trata de que teclees lo mismo cien veces!).

   a) Compara la media y varianza de las distribuciones teóricas con las empíricas obtenidas en las 100 replicaciones del experimento.

   b) Compara la distribución de SSE (o, equivalentemente, de $\hat{\sigma}^2$) con la distribución teórica.

   c) Compara la distribución de SSE_h (o, equivalentemente, de $\hat{\sigma}_h^2$) con la distribución de SSE (o $\hat{\sigma}^2$).

   (Ayuda: Puedes hacer esto de varias formas, empleando un bucle, o empleando la función `lsfit` con un argumento `y` que sea **una matriz** de 100 columnas. Este último procedimiento será más rápido. El primero, probablemente es más educativo, y es el recomendado. Pero si empleas un bucle, **no recomputes en cada iteración mas que lo que requiera ser recomputado**. Observa que $(X'X)^{-1}X'$ no varía).

AYUDAS, SUGERENCIAS Y COMPLEMENTOS

1. La función `rnorm` permite generar con comodidad variables aleatorias con distribución normal. Mira su sintaxis en el manual (o mediante `help(rnorm)`).

2. Cuando es preciso repetir una misma operación muchas veces —acaso con diferentes datos— puede recurrirse, en S-PLUS, en R y en cualquier lenguaje de alto nivel, a sentencias de control de flujo. Por ejemplo, si quisiéramos generar 25 vectores de 50 observaciones normales independientes con media 0 y varianza 1, calcular su media aritmética e imprimirla, no sería preciso que tecleásemos las mismas dos instrucciones 25 veces. Podríamos en cambio hacer,

```
for (i in 1:25) {
   a <- rnorm(50)
   m <- mean(a)
   print(m)
}
```

3. Observa que el bucle presentado más arriba realmente genera lo que se ha dicho. *Cada vez* que ejecutas `rnorm` (o cualquiera de las funciones generando números aleatorios) obtienes un resultado diferente (`rnorm` "se acuerda" de lo último que proporcionó). Observa también que en dos ejecuciones sucesivas obtendrías números aleatorios diferentes: no debe extrañarte que un compañero con un programa idéntico —o tú mismo en dos ocasiones sucesivas— obtengas resultados diferentes.

4. En uno de los ejercicios has de guardar los resultados de 100 ejecuciones. Es muy fácil. Imagina que, en el bucle del apartado anterior, quisieras conservar la media aritmética de cada una de las 25 muestras de 50 normales generadas. Lo podrías hacer así:

```
medias <- rep(0,25)
for (i in 1:25) {
   a <- rnorm(50)
   m <- mean(a)
   print(m)
   medias[i] <- m
}
```

Explicación: inmediatamente antes del bucle, inicializas un vector con 25 elementos que rellenas con ceros (o con cualquier otra cosa). Dentro del bucle, en la pasada i-ésima, el elemento correspondiente de `medias` toma el valor de `m`, la media calculada en esa iteración. Cuando el bucle finaliza, `medias` contendrá lo deseado.

Si lo que quieres guardar no son 100 números sino 100 vectores $\hat{\beta}$ de dimensión, por ejemplo, cinco, podrías inicializar una matriz y guardar cada uno de los vectores generados en una fila.

5. Cuando has de hacer un trabajo que lleva cierto tiempo, habitualmente lo ejecutarás en batch. En `anboto` crearás un fichero con la fuente (las instrucciones que vas a someter a R) e invocarás el programa así:

   ```
   R BATCH  programa.R
   ```

   Los resultados quedrán en un fichero de nombre `programa.Rout` que puedes a continuación editar a tu satisfacción.

   Si trabajas en un ordenador personal, emplea el procedimiento descrito en clase.

6. Si trabajas abriendo una sesión en R desde el editor Emacs (recomendado, y en breve posible en `anboto`), seguramente te interesará dividir la ventana de Emacs en dos (Ctrl-X 2), invocar desde una de ellas R (Esc-X R) y editar en la otra el programa o función hasta tenerlo depurado. Lo verás hacer en clases prácticas.

7. Si escribes un bucle que haga algo muchas veces, pruébalo primero sobre una instancia sencilla del problema. Por ejemplo, transforma un programa como:

   ```
   for (i in 1:1000) {
    ...
   }
   ```

   en otro como

   ```
   iter <- 2
   for (i in 1:iter) {
    ...
   }
   ```

   Cuando éste último se ejecute a tu entera satisfacción, bastará que asignes a `iter` el valor 1000 y lo reejecutes. Las pruebas sólo requerirán el tiempo de dos iteraciones.

8. En las notas de clase, Ejemplo 4.1, pág. 37, tienes un ejemplo en el que te puedes basar.

9. Para examinar si la distribución empírica de algo se aproxima a la que predice la teoría, puedes recurrir a comparar sus momentos (media y varianza, por ejemplo). Puedes también dibujar el histograma de los valores obtenidos (haz `help(hist)`), o utilizar funciones más avanzadas para dibujar su función de densidad estimada, y ver si su forma se aproxima a la teórica.

10. Podrías también recurrir a contrastes de ajuste como los que estudiaste en Estadística I y II: contraste χ^2 o de Kolmogorov-Smirnov, por ejemplo (mira [8] o cualquier otro texto de Estadística que hayas manejado). Hay contrastes más especializados para detectar desviaciones respecto a la normalidad. Puedes mirar la ayuda de la sentencia `qqnorm` y [6] o [3].

11. La función `rnorm` (y similares) emplean un vector de nombre `.Random.seed` donde dejan información sobre el último número aleatorio que han generado. Esto les permite continuar la serie en ejecuciones sucesivas: un buen generador de números aleatorios proporciona series de muchos millones o decenas de millones de valores antes de repetirse. Si —lo que no será el caso en esta tarea— quisieras obtener siempre los mismos números aleatorios (esto es de utilidad, por ejemplo, cuando estás depurando un programa) podrías manipular `.Random.seed`. Haz `help(.Random.seed)` para obtener más información si la quieres. Sobre generadores de números seudo-aleatorios puedes ver si tienes curiosidad [4].

12. Recuerda que siempre tienes ayuda disponible desde dentro de ambos programas invocando `help` y (en el caso de R) `help.start`, que proporciona la ayuda en ventana aparte y en HTML.

Referencias

[1] R.A. Becker, J.M. Chambers, and A.R. Wilks. *The New S Language. A Programming Environment for Data Analysis and Graphics*. Wadsworth & Brooks/Cole, Pacific Grove, California, 1988.

[2] J.M. Chambers and T.J. Hastie. *Statistical Models in S*. Wadsworth & Brooks/Cole, Pacific Grove, Ca., 1992.

[3] R.B. D'Agostino. An omnibus test of normality for moderate and large sample sizes. *Biometrika*, 58:341–348, 1971.

[4] D.E. Knuth. Seminumerical algorithms. In *The Art of Computer Programming*, volume 2. Addison-Wesley, Reading, Ma., 1969.

[5] G.A.F. Seber. *Linear Regression Analysis*. Wiley, New York, 1977.

[6] S.S. Shapiro and M.B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52:591–611, 1965.

[7] A. Fdez. Trocóniz. *Modelos Lineales*. Serv. Editorial UPV/EHU, Bilbao, 1987.

[8] A. Fz. Trocóniz. *Probabilidades. Estadística. Muestreo.* Tebar-Flores, Madrid, 1987.