Taller de Modelación I

Rivera Hernández León Diego,
Segundo Tovar Cristofer Ricardo.

Tarea 6

---

1. Hacer un programa que calcule el determinante de una matriz A, con una función computacional recursiva. Queremos saber qué tan costoso es computacionalmente implementarla.

   a) Se va a repetir para N, un número grande, el siguiente procedimiento: Para n natural fijo, genere una matriz A aleatoria donde cada entrada a_{ij} es un número aleatorio uniforme $(-1, 1)$, calcule su determinante y guarde el tiempo que tardó la compu en hacerlo, $t(k)$, para $k = 1, ..., N$.

   b) Haga el promedio de valores $t(1), ..., t(N)$; para n fijo. Tal promedio se llamará $m(n)$.

   c) Grafique $m(n)$, variando sobre n y haciéndolo *crecer a infinito*. ¿Cómo se comporta la gráfica? ¿Qué tipo de función creen que sea la $m(n)$?

R:

Se computaron los promedios $m(n)$ para distintos valores máximos de n ($nmax$), que es la dimensión de las matrices aleatorias y N, que es el número de total de matrices. Los resultados para distintos valores de $nmax$ y N se encuentran graficados en las figuras 1 a 3.

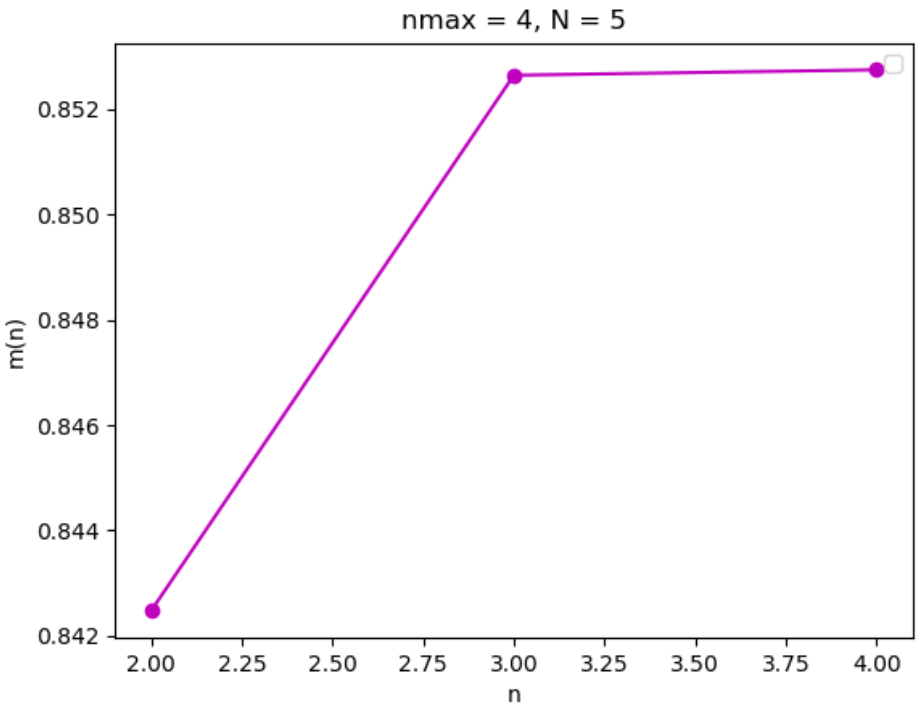


Figura 1: Primer experimento, con parámetros $nmax = 4, N = 5$.

Por la naturaleza recursiva del algoritmo utilizado para calcular el determinante

de una matriz, incluso para parámetros pequeños el programa toma una cantidad considerable de tiempo para obtener los promedios.

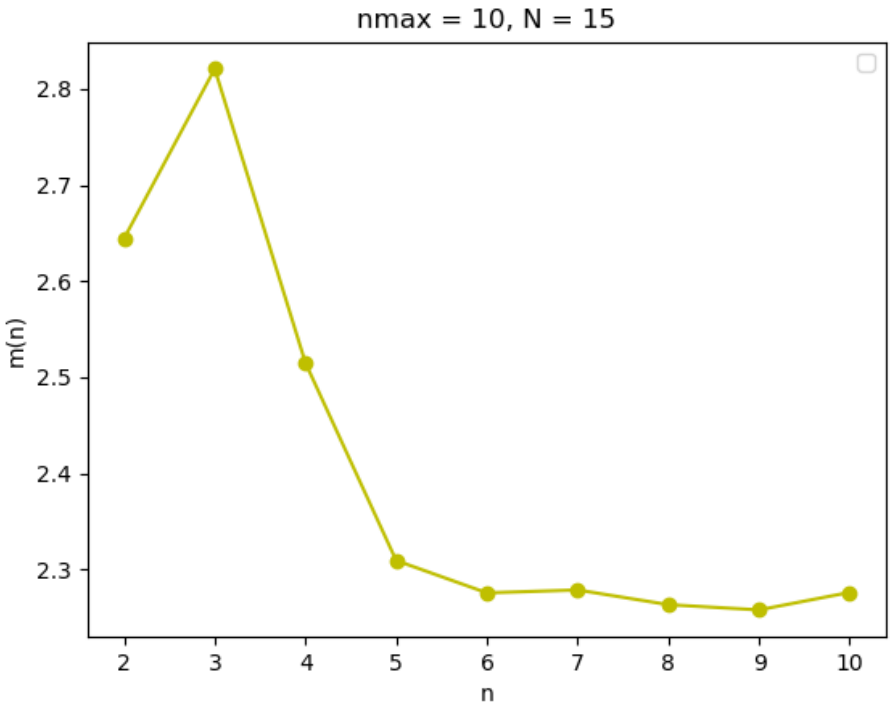


Figura 2: Segundo experimento, con parámetros $nmax = 10, N = 15$.

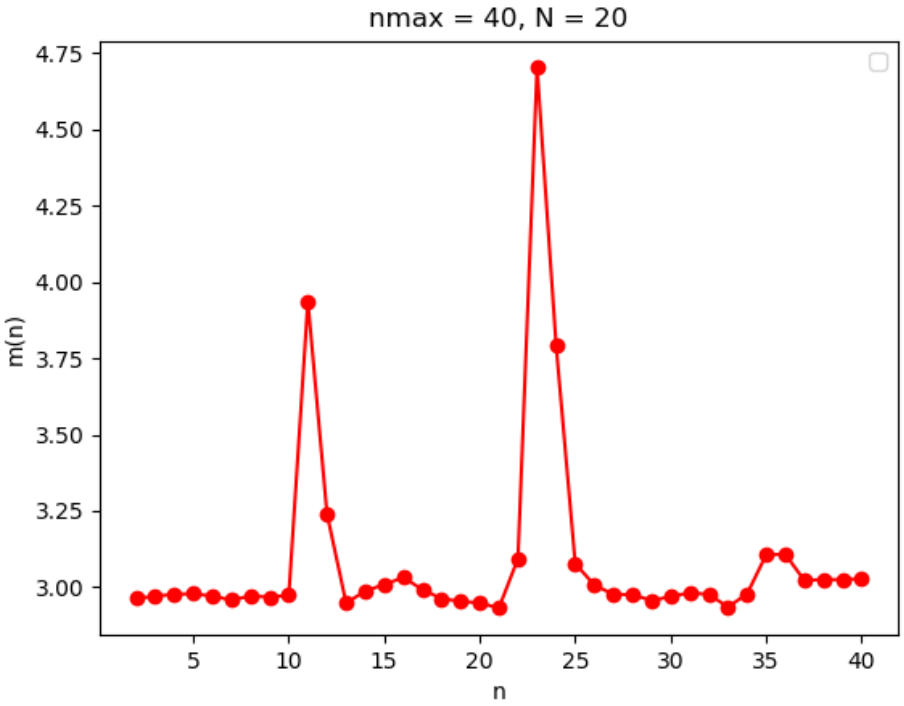


Figura 3: Tercer experimento, parámetros: $nmax = 40, N = 20$

De estas dos últimas gráficas, observamos que en general el tiempo promedio va aumentando de forma aparentemente lineal (con pendiente positiva pero muy pequeña), con excepción de un par de intervalos donde el tiempo promedio se dispara. El tiempo de cómputo es, por supuesto, muy largo.

2. En este ejercicio vamos a comparar el desempeño (de manera empírica) de dos métodos para resolver sistemas de ecuaciones de la forma $Ax = b$.

   a) Haga un programa que calcule la factorización LU para una matriz de tamaño arbitrario.

```
def LU(A):

    m = np.array(A)
    n,n = m.shape
    L = np.diag((1.0)*np.ones(n))
    U = np.matrix.copy(m)

    for i in range(n):
        for j in range (1, n-i):
            L[i+j, i] = U[i+j,i]/U[i,i]
            U[i+j,:] = U[i+j,:] - (U[i+j,i]/U[i,i])*U[i,:]
    return L,U
```

   b) Haga un programa que calcule la factorización QR para una matriz de tamaño arbitrario.

```
def QR(A):
    if np.linalg.det(A) != 0: #comprobar que tenga solución
        n,n = A.shape
        Q=np.zeros((n,n))
        u=A[:,0]
        e=u/np.linalg.norm(u)
        Q[:,0]=e
        for i in range(1,n):
            u=A[:,i]
            for j in range(i):
                aux=np.dot(A[:,j+1],Q[:,j])*Q[:,j]
                u=u-aux
            e=u/np.linalg.norm(u)
            Q[:,i]=e
        R=np.matmul(Q.transpose(),A)
    return Q,R
```

c) Para cada k y n fijos, donde k corre sobre los valores $1, ..., N$ y N es un número grande, genere una matriz A aleatoria de tamaño $n \times n$ donde cada entrada a_{ij} es un número aleatorio uniforme en $(-1, 1)$. Haga lo mismo para un vector b de tamaño $n \times 1$.

d) Resuelva el sistema $Ax = b$ usando ambos métodos. Guarde $l_k(n)$, y $m_k(n)$ como el tiempo que Python tardó en computar la respuesta para cada método (haga que la computadora calcule este tiempo).

```
def tomar_tiempo(N,n):
    tiempoLU=[]
    tiempoQR=[]
    for i in range(N):
        #Que tome valores entre -1,1
        A=np.random.uniform(-1,1,(n,n))
        b=np.random.uniform(-1,1,(n,1))

        #tomar_tiempo a LU
        tempo1 = timeit.default_timer()
        L,U=LU(A)
        b1=np.dot(np.linalg.inv(L),b)
        x=np.dot(np.linalg.inv(U),b1)
        tempo2 = timeit.default_timer()
        tiempoLU.append(tempo2-tempo1)

        #tomar_tiempo a QR
        tempo3 = timeit.default_timer()
        Q,R=QR(A)
        b1=np.dot(np.linalg.inv(Q),b)
        x=np.dot(np.linalg.inv(R),b1)
        tempo4 = timeit.default_timer()
        tiempoQR.append(tempo4-tempo3)
    l=promedio(tiempoLU)
    m=promedio(tiempoQR)
    return tiempoLU, tiempoQR, l,m
```

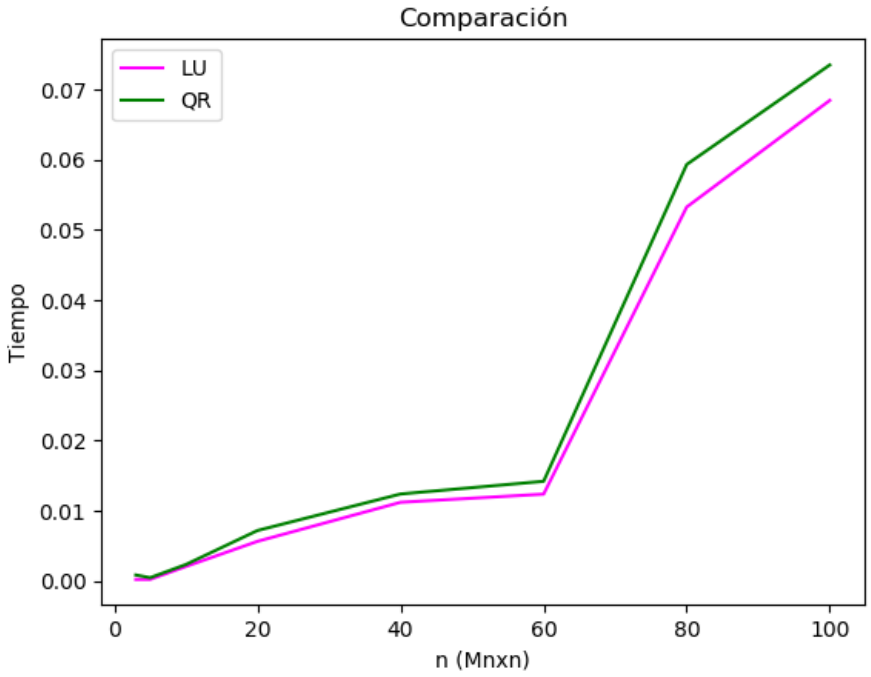
e) Finalmente calcule

$$l(n) = \frac{\sum_{k=1}^{N} l_k(n)}{N}, \text{ y } m(n) = \frac{\sum_{k=1}^{N} m_k(n)}{N},$$

donde $l(n)$ (respectivamente $m(n)$) es el tiempo de cómputo promedio para resolver el sistema para una matriz A de tamaño n.

f) Haga una gráfica de $l(n)$ y $m(n)$ variando n y haciéndolo crecer mucho.

```
PS C:\Users\crist> & python c:/Users/crist/Downloads/Ejercicio2.py
El tiempo promedio de LU es 0.0004315800000000536
El tiempo promedio de QR es 0.003710119999999906
```



g) ¿Qué método fue mejor? Concluya.

En el tiempo de ejecución, QR se ve superior a lo que es para ejecuciones en este intervalo. Recursivamente, son métodos muy ineficientes, pues requieren ser llamadas muchas veces y podemos observar que para gráficas de dimensión mayor a 100, tarda mucho tiempo.