Álgebra Lineal Computacional

Primer Cuatrimestre 2021

Trabajo Práctico N° 1.

Motivación:
[Esta primera parte busca explicar de dónde surge el problema que queremos resolver, para ponerlo en contexto. No es necesario comprender a la perfección todos los detalles para resolver el TP.]
Ciertos métodos para la resolución de ecuaciones diferenciales dan lugar a un sistema lineal cuyo tamaño puede crecer arbitrariamente dependiendo de la discretización que se realice de la ecuación. A modo de ejemplo, consideremos la ecuación de Poisson en un intervalo. Concretamente, buscamos una función $u : [0, 1] \to \mathbb{R}$ tal que:

$$\begin{cases} -u''(x) &= f(x) \quad \forall x \in (0,1) \\ u(0) &= 0 \\ u(1) &= 0 \end{cases}$$

Esta ecuación puede servir de modelo para diversos fenómenos. Por ejemplo: si f representa una fuente de calor, u es la distribución estacionaria de temperatura generada por f; si f es una distribución de cargas eléctricas, u es potencial eléctrico inducido por f, etc.
Para resolver esta ecuación podemos observar que dado un valor de $h > 0$, la siguiente expresión es una aproximación de la derivada segunda:

$$u''(x) \sim \frac{u(x+h) - 2u(x) + u(x-h)}{h^2}. \tag{1}$$

Consideremos entonces una grilla equiespaciada del intervalo $[0, 1]$ dada por los puntos $x_j = jh$, para $j = 0, \dots, J+1$, con $J + 1 = \frac{1}{h}$. Llamemos u_j a nuestra aproximación de $u(x_j)$. Es importante observar que u_0 y u_{J+1} no son incógnitas de nuestro problema, dado que se corresponden con $u(0)$ y $u(1)$ que sabemos que valen 0. Si utilizamos (1) para aproximar u'' en cada x_j obtenemos ecuaciones de la forma:

$$-\frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} = f(x_j), \qquad j = 2, \dots, J-1$$

Para $j = 1$ y para $j = J$ la ecuación anterior también vale, pero aparecen u_0 y u_{J+1} respectivamente, por lo que podemos escribir las ecuaciones correspondientes aparte:

$$-\frac{u_2 - 2u_1}{h^2} = f(x_1)$$

$$-\frac{-2u_J + u_{J-1}}{h^2} = f(x_J)$$

La ecuación general para $j = 2, \dots, J-1$ junto con estas dos nos dan un sistema de J ecuaciones para J incógnitas (u_j, con $j = 1, \dots, J$). Si escribimos el sistema matricialmente, obtenemos:

$$\underbrace{-\frac{1}{h^2}\begin{pmatrix} -2 & 1 & 0 & 0 & 0 & \dots & 0 \\ 1 & -2 & 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & -2 & 1 & 0 & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 0 & 1 & -2 & 1 \\ 0 & \dots & 0 & 0 & 0 & 1 & -2 \end{pmatrix}}_{\boldsymbol{A}} \underbrace{\begin{pmatrix} u_1 \\ u_2 \\ \\ \vdots \\ \\ u_{J-1} \\ u_J \end{pmatrix}}_{\boldsymbol{u}} = \underbrace{\begin{pmatrix} f(x_1) \\ f(x_2) \\ \\ \vdots \\ \\ f(x_{J-1}) \\ f(x_J) \end{pmatrix}}_{\boldsymbol{f}}.$$

En conclusión: para aproximar la solución de la ecuación (1) podemos resolver este sistema. Intuitivamente cuánto menor sea h, más *fina* será la grilla y mejor la aproximación. Pero achicar h implica agrandar J, lo que da lugar a sistemas cada vez más grandes, cuya resolución es computacionalmente más costosa. Sin embargo, el sistema tiene la ventaja de ser *ralo* (tiene muchos ceros), y, en particular, es tridiagonal. El objetivo de este trabajo es aprovechar este ejemplo para comparar la velocidad del método LU clásico con un método LU adaptado para matrices tridiagonales.

Ejercicio 1. Implementar un programa que reciba como input el tamaño J y devuelva la matriz $\boldsymbol{A}$. (Sug.: Revisar los usos del comandos np.diag).

Ejercicio 2. Implementar los programas de los ejercicios 13 y 16 de la Práctica 2. Verificar que funcionen correctamente en ejemplos pequeños.

Ejercicio 3. Convencerse de que al aplicar el algoritmo LU a una matriz tridiagonal sólo es necesario operar con los elementos de las tres diagonales no nulas y tanto $\boldsymbol{L}$ como $\boldsymbol{U}$ quedan bidiagonales.

Ejercicio 4. Implementar un programa que reciba como input una matriz $\boldsymbol{A}$, asumiendo que es tridiagonal y realice la descomposición LU operando realizando sólo las operaciones en las diagonales no nulas.

Ejercicio 5. Implementar funciones similares a las del ejercicio 13 de la Práctica 2 pero que realicen el despeje asumiendo que las matrices $\boldsymbol{L}$ y $\boldsymbol{U}$ son bidiagonales.

Ejercicio 6. Implementar un programa que, haciendo uso de los anteriores, resuelva un sistema tridiagonal sin realizar operaciones con los elementos de la matriz que se saben nulos a priori.

Ejercicio 7. Implementar un programa que resuelva un sistema de la forma $\boldsymbol{Ax} = \boldsymbol{b}$, donde $\boldsymbol{A}$ es tridiagonal. El programa debe recibir cuatro vectores (las tres diagonales de $\boldsymbol{A}$ y el vector $\boldsymbol{b}$), aprovechar el programa anterior obtener las diagonales de $\boldsymbol{L}$ y $\boldsymbol{U}$ tales que $\boldsymbol{A} = \boldsymbol{LU}$ y a partir de ellas despejar la solución. Verificar que funcione correctamente en ejemplos pequeños.

Ejercicio 8. Para testear todos los programas en un caso realista, podemos suponer que en la ecuación diferencial tenemos $f(x) = \operatorname{sen}(\pi x)$. La solución exacta en este caso es $u(x) = \frac{\operatorname{sen}(\pi x)}{\pi^2}$. Lo más conveniente es generar la grilla a partir del valor de J usando np.linspace. Se puede recuperar el valor de h desde la grilla. Completar el siguiente código, entenderlo y correrlo:

```
J       = 20
f       = lambda t: np.sin(np.pi*t)
x       = np.linspace(0,1,J+2) #J puntos interiores + los dos extremos
h       = x[1]
b       = f(x[1:-1])
A       = ... #correr el programa que genera A
u       = np.zeros(J+2)
u[1:-1] = ... #resolver el sistema Au=b
plt.plot(x,u)
U       = lambda t: np.sin(np.pi*t)/np.pi**2 #sol exacta
plt.plot(x,U(x))
```

Probarlo resolviendo el sistema con ambos métodos. Si todo anda bien, los dos gráficos (solución aproximada y solución exacta) deberían coincidir.

Ejercicio 9. Finalmente, queremos estudiar cómo evoluciona el tiempo de resolución de cada uno de los métodos anteriores en función de h. La idea es la siguiente: consideramos varios valores de J (y

por lo tanto, varios valores de h). Para cada J, creamos la matriz $\boldsymbol{A}$, resolvemos con el método LU tradicional y almacenamos el tiempo que llevó la resolución; hacemos lo mismo con el método LU adaptado a tridiagonales. De este modo obtenemos tres vectores: un vector j_vec, con los distintos valores de J, un vector t_lu con los tiempos de resolución del método LU para cada valor de J y un vector t_trid con los tiempos de resolución del método para tridiagonales. Graficamos t_lu en función de j_vec y t_trid en función de j_vec.
Sugerencias:

- Trabajar con el ejemplo $f(x) = \text{sen}(\pi x)$, como en el ejercicio anterior.
- Para calcular tiempos podemos usar la función time.time(), del siguiente modo:

```
import time
...
start = time.time()
comando_a_cronometrar
tiempo = time.time()-start
```

  La variable tiempo guarda el tiempo (en segundos) que llevó realizar el comando_a_cronometrar.

- Lo ideal es crear inicialmente un vector j_vec con los valores de J e irlo recorriendo. También hay que crear un vector para almacenar los tiempos de cada método. Por ejemplo:

```
j_vec  = np.linspace(100,1001,100) #valores de J de 100 a 1000, saltando
  de a 100
N      = len(j_vec)
t_lu   = np.zeros(N) #tiempos de lu
t_trid = np.zeros(N) #tiempos de lu tridiagonal
for i in .... :
    J         = j_vec[i]
    ....
    t_lu[i]   = ...
    ....
    t_trid[i] = ...
```