[MN] Apunte 5 - 2021

October 26, 2021

Métodos numéricos

Segundo cuatrimestre 2021

Práctica 5: Ecuaciones elípticas

Cátedra: Pablo Dmitruk

1 Ecuaciones elípticas

1.1 Ecuación de Poisson

Como ejemplo paradigmático de una ecuación elíptica vamos a considerar la ecuación de Poisson escalar

$$\nabla^2 h(\boldsymbol{x}) = f(\boldsymbol{x}).$$

Otras ecuaciones elípticas pueden transfomarse mediante un cambio de variables en una ecuación de este tipo. En contextos físicos, a la función f suele denominársela forzado o fuente.

Vale remarcar que el único caso de ecuaciones de Poisson vectoriales (i.e. donde $h \in \mathbb{R}^n$) que vamos a considerar es el cartesiano, donde $\nabla^2 \boldsymbol{h}$ es sencillamente el laplaciano escalar de cada componente cartesiana de $\boldsymbol{h}$, es decir $\nabla^2 \boldsymbol{h} = \nabla^2 h_x \hat{\boldsymbol{x}} + \nabla^2 h_y \hat{\boldsymbol{y}}$ para el caso 2D.

En el caso 1D, la ecuación de Poisson escalar resulta sencillamente

$$\frac{\mathrm{d}^2 h}{\mathrm{d}x^2} = f(x),$$

y será objeto de los primeros ejercicios de esta Pŕactica.

A la hora de describir problemas físicos, esta tipo de ecuaciones siempre aparecen aparejadas con condiciones sobre los bordes del dominio $\partial\Omega$. Las condiciones de contorno más comunes que podemos hallar son sobre el valor de h, sobre su derivada normal o una combinación de estas opciones, es decir que podemos tener

$$f|_{\partial\Omega} = g(\boldsymbol{s}), \qquad \text{(Condición de Dirichlet)}$$
$$(\boldsymbol{\nabla} f)|_{\partial\Omega} \cdot \hat{\boldsymbol{n}} = g(\boldsymbol{s}), \qquad \text{(Condición de Neumann)}$$
$$(\boldsymbol{\nabla} f)|_{\partial\Omega} \cdot \hat{\boldsymbol{n}} + a f|_{\partial\Omega} = g(\boldsymbol{s}), \qquad \text{(Condición de Robin)}$$

donde $\hat{\boldsymbol{n}}$ es un versor localmente normal al contorno $\partial\Omega$ y $\boldsymbol{s} \in \partial\Omega$. Observen que la condición de Robin además del valor $g(x)$ agrega un parámetro extra a que establece una combinación lineal entre el valor de la función y su derivada normal.

1.2 Imposición de condiciones de contorno

1.2.1 Vector con condiciones de Dirichlet

Para resolver ecuaciones en presencia de contornos hemos utilizado hasta el momento un vector auxiliar $\mathbf{b}$. La idea de este procedimiento es la de construir una matriz de diferenciación $\mathbb{D}_{xx}$ (asumimos que opera en la dirección x) que calcula la derivada segunda en todos los puntos interiores (donde podemos integrar nuestra ecuación).

Para recordar esto, consideremos una ecuación de Poisson 1D discretizada a 4to orden con condiciones de Dirichlet $h(x_0) = a$ y $h(x_0) = b$. Con la estrategia mencionada el problema discretizado resulta

$$\mathbb{D}_{xx}\mathbf{h} + \mathbf{b} = \mathbf{f},$$

con

$$\mathbb{D}_{xx} = d \begin{pmatrix} 45 & -154 & 214 & -156 & 61 & -10 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 16 & -30 & 16 & -1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 16 & -30 & 16 & -1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 16 & -30 & 16 & -1 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & -1 & 16 & -30 & 16 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & -1 & 16 & -30 & 16 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & -1 & 16 & -30 & 16 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & -10 & 61 & -156 & 214 & -154 & 45 \end{pmatrix},$$

$$\mathbf{h} = \begin{pmatrix} h_1 \\ h_2 \\ \vdots \\ h_{N-2} \\ h_{N-1} \end{pmatrix}, \qquad \mathbf{b} = d \begin{pmatrix} 0 \\ -a \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ -b \\ 0 \end{pmatrix}, \qquad \mathbf{f} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{N-2} \\ f_{N-1} \end{pmatrix}, \qquad d = \frac{1}{12(\Delta x)^2}.$$

Como es usual, notamos con ϕ_i a $\phi(x_i)$ y consideramos $N-1$ puntos interiores ($N+1$ puntos totales). Noten que como sucede para problemas de órdenes superiores, las primeras (respect. últimas) filas usan diferencias finitas adelantas (respect. atrasadas) de 4to orden, mientras que el resto de las filas utilizan diferencias finitas centradas de igual orden. En nuestro ejemplo esto se observa en la primer y última fila. Adicionalmente, y como seguramente habrán reparado, en $\mathbf{b}$ colocamos las condiciones de contorno, que en este caso aparecen en la segunda y penúltima fila.

Si bien resulta poco estético que la condición de contorno aparezca en la segunda (penúltima) fila y no en la primera (última), de cualquier modo da lugar a un problema bien formulado. Esto podría modificarse mediante la utilización de esquemas que no sean ni centrados ni hacia un solo lado, por ejemplo, uno que use 1 punto a izquierda y 4 a derecha para el ejemplo en cuestión. Sin embargo, el ejemplo propuesto utiliza esquemas o bien centrados o bien hacia un solo lado para

que resulte consistente con el funcionamiento de la biblioteca *Findiff* que introduciremos en una sección posterior.

Reparen en que abstrayendo los detalles relacionados con el orden de aproximación, cada fila de la expresión $\mathbb{D}_{xx}\mathbf{h} + \mathbf{b} = \mathbf{f}$ establece sencillamente

$$h_i'' = f_i.$$

Finalmente, vale mencionar que, como vienen haciendo hasta ahora, para hallar los valores desconocidos de h: $h_1, \ldots, h_{N-1}$ basta con realizar la operación $\mathbf{h} = \mathbb{D}_{xx}^{-1}(\mathbf{f} - \mathbf{b})$.

1.2.2 Matriz de diferenciación ampliada con condiciones tipo Dirichlet

Otra forma, mencionada tangencialmente en prácticas anteriores, es la de agrandar nuestro sistema de ecuaciones y formar una matriz $\tilde{\mathbb{D}}_{xx}$ y un vector $\tilde{\mathbf{f}}$ ampliados que contengan a las ecuaciones para los contornos. Noten que para el caso Dirichlet las ecuaciones asociadas a los contornos resultan sencillamente

$$h_0 = a, \qquad h_N = b.$$

Agregando dos filas y dos columnas a $\mathbb{D}_{xx}$ (y consecuentemente dos filas a $\mathbf{h}$ y $\mathbf{f}$), una forma de incorporar las ecuaciones de los contornos es mediante

$$\tilde{\mathbb{D}}_{xx} = d \begin{pmatrix} \frac{1}{d} & 0 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 45 & -154 & 214 & -156 & 61 & -10 & \ldots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 16 & -30 & 16 & -1 & 0 & 0 & \ldots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 16 & -30 & 16 & -1 & 0 & \ldots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 & -1 & 16 & -30 & 16 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & -1 & 16 & -30 & 16 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots & -10 & 61 & -156 & 214 & -154 & 45 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{d} \end{pmatrix},$$

$$\mathbf{h} = \begin{pmatrix} h_0 \\ h_1 \\ h_2 \\ \vdots \\ h_{N-2} \\ h_{N-1} \\ h_N \end{pmatrix}, \qquad \tilde{\mathbf{f}} = \begin{pmatrix} a \\ f_1 \\ f_2 \\ \vdots \\ f_{N-2} \\ f_{N-1} \\ b \end{pmatrix}, \qquad d = \frac{1}{12(\Delta x)^2},$$

y de esta forma nuestro problema resulta

$$\tilde{\mathbb{D}}_{xx}\mathbf{h} = \tilde{\mathbf{f}}.$$

Si bien este problema matricial resulta ligeramente más grande que la formulación empleando $\mathbf{b}^{\dagger}$, permite una escritura bastante más sencilla, en particular para órdenes superiores, donde nos valdremos de la biblioteca *Findiff* para obtener automáticamente las matrices de diferenciación $\mathbb{D}$.

†: Y, por lo tanto, marginalmente más costoso computacionalmente en caso que se lo programe de manera ingenua.

1.2.3 Matriz de diferenciación ampliada con condiciones de Neumann

Otra ventaja de la formulación ampliada, es que permite incorporar fácilmente condiciones de tipo Neumann a nuestro problema. Las ecuaciones sobre los bordes serán

$$\left[-\frac{\mathrm{d}h}{\mathrm{d}x}\right]_{x_0} = a, \qquad \left[\frac{\mathrm{d}h}{\mathrm{d}x}\right]_{x_N} = b.$$

Noten que en x_0 prescribimos $-h'_0$ ya que la derivada normal a la frontera en dicho extremo está dada por $-\mathrm{d}/\mathrm{d}x$ (consideramos la normal exterior, como es la convención usual en física). La condición de contorno en x_0 (respct. x_N) vamos a discretizarlas usando un estimador adelantado (respect. atrasado) para la derivada primera, de forma de que utilice información hacia adentro del dominio, es decir

$$\begin{aligned} -h'_0 &= \frac{25h_0 - 48h_1 + 36h_2 - 16h_3 + 3h_4}{12\Delta x}, \\ h'_N &= \frac{3h_{N-4} - 16h_{N-3} + 36h_{N-2} - 48h_{N-1} + 25h_N}{12\Delta x}. \end{aligned}$$

Luego, podemos incorporar las condiciones de contorno a nuestro problema formulando

$$\tilde{\mathbb{D}}_{xx}\mathbf{h} = \tilde{\mathbf{f}}.$$

con

$$\tilde{\mathbb{D}}_{xx} = d \begin{pmatrix} 25q & -48q & 36q & -16q & 3q & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 45 & -154 & 214 & -156 & 61 & -10 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 16 & -30 & 16 & -1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 16 & -30 & 16 & -1 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & -1 & 16 & -30 & 16 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & -1 & 16 & -30 & 16 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & -10 & 61 & -156 & 214 & -154 & 45 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 3q & -16q & 36q & -48q & 25q \end{pmatrix},$$

$$\mathbf{h} = \begin{pmatrix} h_0 \\ h_1 \\ h_2 \\ \vdots \\ h_{N-2} \\ h_{N-1} \\ h_N \end{pmatrix}, \qquad \tilde{\mathbf{f}} = \begin{pmatrix} a \\ f_1 \\ f_2 \\ \vdots \\ f_{N-2} \\ f_{N-1} \\ b \end{pmatrix}, \qquad d = \frac{1}{12(\Delta x)^2}, \qquad q = \Delta x.$$

Esta estrategia se puede aplicarse fácilmente par el caso donde se tienen condiciones de contorno mixtas, i.e. Dirichlet en un contorno (por ejemplo x_0) y Neumann en otro (en este caso x_N), medi-

ante la matriz

$$\tilde{\mathbb{D}}_{xx} = d \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 45 & -154 & 214 & -156 & 61 & -10 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 16 & -30 & 16 & -1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 16 & -30 & 16 & -1 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & -1 & 16 & -30 & 16 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & -1 & 16 & -30 & 16 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & -10 & 61 & -156 & 214 & -154 & 45 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 3q & -16q & 3q & -48q & 25q \end{pmatrix}$$

Naturalmente, la generalización a condiciones de tipo Robin o cualquier otro resulta inmediata.

1.3 Combinación de operadores de derivación en distintas direcciones

Un problema que resolvimos en la práctica previa es el de estimar el laplaciano 2D. Dado una matriz con los valores discretos de $h(x_i, y_j) = h_{ij}$ y sendas matrices de diferenciación con respecto a x y a y, $\mathbb{D}_{xx}$ y $\mathbb{D}_{yy}$, podiamos calcular

$$\mathbf{f}_{ij} = (\nabla^2 h)_{ij} \approx [\mathbb{D}_{xx}\mathbb{H}]_{ij} + [\mathbb{H}\,\mathbb{D}_{yy}^{\intercal}]_{ij},$$

con $\mathbb{H}$ la matriz que contiene a los h_{ij}. Sin embargo, esta operación no resulta fácil de invertir.

Para solucionar esto, asumiendo que nuestro problema discreto cuanta con N_x puntos en la dirección x y N_y en y, podriamos transformar la matriz h_{ij} en un vector $\check{h}_k$ mediante la transformación $k = N_y i + j$. Vamos a llamar *aplanar* a esta operación, y para fijar ideas vamos a considerar $N_x = 3$ y $N_y = 4$ puntos totales (i.e. incluyen los contornos), teniendo entonces

$$\mathbb{H} = \begin{pmatrix} h_{00} & h_{01} & h_{02} & h_{03} \\ h_{10} & h_{11} & h_{12} & h_{13} \\ h_{20} & h_{21} & h_{22} & h_{23} \end{pmatrix} \qquad \longrightarrow \qquad \check{\mathbf{h}} = \begin{pmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{03} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{13} \\ h_{20} \\ h_{21} \\ h_{22} \\ h_{23} \end{pmatrix}.$$

Luego debería ser posible construir matrices $\check{\mathbb{D}}_{xx}$ tal que $(\partial_{xx} h)_k = (\check{\mathbb{D}}_{xx}\check{\mathbf{h}})_k$ y $\check{\mathbb{D}}_{yy}$ de forma que $(\partial_{yy} h)_k = (\check{\mathbb{D}}_{yy}\check{\mathbf{h}})_k$ (en ambos casos, en los puntos interiores).

1.3.1 Mapear $\mathbb{D}\,\mathbb{H}$

Veamos cómo lograr esto con estimadores de segundo orden y asumiendo condiciones de Dirichlet. En primer lugar veamos el resultado de cómo veniamos trabajando hasta el momento:

$$\begin{pmatrix} h_{00} & h_{01} & h_{02} & h_{03} \\ (\partial_{xx}h)_{10} & (\partial_{xx}h)_{11} & (\partial_{xx}h)_{12} & (\partial_{xx}h)_{13} \\ h_{20} & h_{21} & h_{22} & h_{23} \end{pmatrix} \approx \mathbb{D}_{xx}\mathbb{H} = \frac{1}{d}\begin{pmatrix} d & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & d \end{pmatrix}\begin{pmatrix} h_{00} & h_{01} & h_{02} & h_{03} \\ h_{10} & h_{11} & h_{12} & h_{13} \\ h_{20} & h_{21} & h_{22} & h_{23} \end{pmatrix}$$

$$= \begin{pmatrix} h_{00} & h_{01} & h_{02} & h_{03} \\ \frac{h_{20}-2h_{10}+h_{00}}{d} & \frac{h_{21}-2h_{11}+h_{01}}{d} & \frac{h_{22}-2h_{12}+h_{02}}{d} & \frac{h_{23}-2h_{13}+h_{03}}{d} \\ h_{20} & h_{21} & h_{22} & h_{23} \end{pmatrix}.$$

Luego, podemos definir $\breve{\mathbb{D}}_{xx}$ de la siguiente manera de forma de realizar la misma operación sobre la matriz aplanada

$$\begin{pmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{03} \\ (\partial_{xx}h)_{10} \\ (\partial_{xx}h)_{11} \\ (\partial_{xx}h)_{12} \\ (\partial_{xx}h)_{13} \\ h_{20} \\ h_{21} \\ h_{22} \\ h_{23} \end{pmatrix} \approx \breve{\mathbb{D}}_{xx}\breve{\mathbf{h}} = \frac{1}{d}\begin{pmatrix} d & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & d & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & d & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & d & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & d & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & d & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & d & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & d \end{pmatrix}\begin{pmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{03} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{13} \\ h_{20} \\ h_{21} \\ h_{22} \\ h_{23} \end{pmatrix}$$

$$= \begin{pmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{03} \\ \frac{h_{20}-2h_{10}+h_{00}}{d} \\ \frac{h_{21}-2h_{11}+h_{01}}{d} \\ \frac{h_{22}-2h_{12}+h_{02}}{d} \\ \frac{h_{23}-2h_{13}+h_{03}}{d} \\ h_{20} \\ h_{21} \\ h_{22} \\ h_{23} \end{pmatrix}$$

con $d = 1/(\Delta x)^2$.

Este ejemplo puede generalizarse y, en general, podremos mapear un operador $\mathbb{D}$ que opera sobre las columnas de nuestra matriz (como es el caso de $\mathbb{D}_{xx}$) de la siguiente manera

$$\mathbb{D}_{ij} \qquad \longrightarrow \qquad \breve{\mathbb{D}}_{[iN_y,\ldots,iN_y+N_y-1]\circ[jNy,\ldots,jN_y+N_y-1]}.$$

con $\circ$ el operador que dadas las secuencias a_p y b_p, $a \circ b$ genera el conjunto de duplas (a_p, b_p). Dicho de otra manera, cada elemento $\mathbb{D}_{ij}$ aparece repetido para todas las filas de $\breve{\mathbb{D}}$ entre iN_y y $(i+1)N_y$ (no incluída). En cada fila sucesiva, dicho elemento va siendo desplazado un lugar hacia la derecha.

1.3.2 Mapear $\mathbb{H}\,\mathbb{D}^{\intercal}$

De forma análoga, analicemos con un ejemplo el caso de un operador que, en contraposición, opera sobre las filas de una matriz, como es el caso de $\mathbb{D}_{yy}$. Consideremos un problema como el de antes pero donde ahora tenemos condiciones de contorno periódicas. Las prácticas anteriores esto lo podíamos resolver como

$$\begin{pmatrix} (\partial_{yy}h)_{00} & (\partial_{yy}h)_{01} & (\partial_{yy}h)_{02} & (\partial_{yy}h)_{03} \\ (\partial_{yy}h)_{10} & (\partial_{yy}h)_{11} & (\partial_{yy}h)_{12} & (\partial_{yy}h)_{13} \\ (\partial_{yy}h)_{20} & (\partial_{yy}h)_{21} & (\partial_{yy}h)_{22} & (\partial_{yy}h)_{23} \end{pmatrix} \approx \mathbb{H}\,\mathbb{D}_{yy}^{\intercal} = \begin{pmatrix} h_{00} & h_{01} & h_{02} & h_{03} \\ h_{10} & h_{11} & h_{12} & h_{13} \\ h_{20} & h_{21} & h_{22} & h_{23} \end{pmatrix} \frac{1}{(\Delta y)^2} \begin{pmatrix} -2 & 1 & 0 & 1 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 1 & 0 & 1 & -2 \end{pmatrix}$$

$$\approx \frac{1}{(\Delta y)^2} \begin{pmatrix} h_{03}-2h_{00}+h_{01} & h_{00}-2h_{01}+h_{02} & h_{01}-2h_{02}+h_{03} & h_{02}-2h_{03}+h_{00} \\ h_{13}-2h_{10}+h_{11} & h_{10}-2h_{11}+h_{12} & h_{11}-2h_{12}+h_{13} & h_{12}-2h_{13}+h_{10} \\ h_{23}-2h_{20}+h_{21} & h_{20}-2h_{21}+h_{22} & h_{21}-2h_{22}+h_{23} & h_{22}-2h_{23}+h_{20} \end{pmatrix}.$$

que podemos escribir para $\check{\mathbf{h}}$

$$\begin{pmatrix} (\partial_{yy}h)_{00} \\ (\partial_{yy}h)_{01} \\ (\partial_{yy}h)_{02} \\ (\partial_{yy}h)_{03} \\ (\partial_{yy}h)_{10} \\ (\partial_{yy}h)_{11} \\ (\partial_{yy}h)_{12} \\ (\partial_{yy}h)_{13} \\ (\partial_{yy}h)_{20} \\ (\partial_{yy}h)_{21} \\ (\partial_{yy}h)_{22} \\ (\partial_{yy}h)_{23} \end{pmatrix} \approx \check{\mathbb{D}}_{yy}\check{\mathbf{h}} = \frac{1}{(\Delta y)^2} \begin{pmatrix} -2 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -2 \end{pmatrix} \begin{pmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{03} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{13} \\ h_{20} \\ h_{21} \\ h_{22} \\ h_{23} \end{pmatrix}$$

$$= \frac{1}{(\Delta y)^2} \begin{pmatrix} h_{03}-2h_{00}+h_{01} \\ h_{00}-2h_{01}+h_{02} \\ h_{01}-2h_{02}+h_{03} \\ h_{02}-2h_{03}+h_{00} \\ h_{13}-2h_{10}+h_{11} \\ h_{10}-2h_{11}+h_{12} \\ h_{11}-2h_{12}+h_{13} \\ h_{12}-2h_{13}+h_{10} \\ h_{23}-2h_{20}+h_{21} \\ h_{20}-2h_{21}+h_{22} \\ h_{21}-2h_{22}+h_{23} \\ h_{22}-2h_{23}+h_{20} \end{pmatrix}.$$

Esta expresión resulta inmediata de generalizar a otros operadores matriciales $\mathbb{D}$: consiste sencillamente en repetir cada N_y filas la matriz $\mathbb{D}$ y desplazarla en N_y columnas cada vez. En forma indicial podemos escribir

$$\mathbb{D}_{ij} \qquad \longrightarrow \qquad \check{\mathbb{D}}_{[i,i+N_y,\ldots,i+(N_x-1)N_y]\circ[j,j+Ny,\ldots,j+(N_x-1)N_y]},$$

es decir, el elemento $_{ij}$ aparece cada N_y filas, desplazado N_y columnas cada vez.

1.3.3 Operador laplaciano sobre matrices aplanadas

Luego del tedioso proceso de la aritmética de índices, podemos ahora expresar a nuestro operador laplaciano como

$$(\nabla^2 h)_i = [\breve{\mathbb{D}}_{xx}]_{ik}\breve{\mathbf{h}}_k + [\breve{\mathbb{D}}_{yy}]_{ik}\breve{\mathbf{h}}_k = [\breve{\mathbb{L}}]_{ik}\breve{\mathbf{h}}_k,$$

con $\breve{\mathbb{L}} = \breve{\mathbb{D}}_{xx} + \breve{\mathbb{D}}_{yy}$. Luego para obtener $\breve{\mathbf{h}}$, resolviendo la ecuación de Poisson que mencionamos al inicio de esta sección, basta hallar la solución a

$$\breve{\mathbb{L}}\breve{\mathbf{h}} = \breve{\mathbf{f}},$$

por ejemplo, invirtiendo $\breve{\mathbb{L}}$, pudiendo finalmente revertir el proceso de aplanado para recuperar $\mathbb{H}$, que era nuestra incógnita original.

1.3.4 Sobre los contornos en 2 dimensiones

Si bien la estrategia mencionada más arriba funciona correctamente para condiciones de contorno periódicas en ambas direcciones, como habrán notado la misma posee algunas dificultades en el tratamiento de los contornos. Por ejemplo, considerando operadores de derivada segunda (el orden es indistinto) con condiciones de Dirichlet para un problema con $N_x = 4$, $N_y = 4$, $\mathbb{D}_{xx}\mathbb{H}$ devuelve

$$\mathbb{D}_{xx}\mathbb{H} \approx \begin{pmatrix} h_{00} & h_{01} & h_{02} & h_{03} \\ (\partial_{xx}h)_{10} & (\partial_{xx}h)_{11} & (\partial_{xx}h)_{12} & (\partial_{xx}h)_{13} \\ (\partial_{xx}h)_{20} & (\partial_{xx}h)_{21} & (\partial_{xx}h)_{22} & (\partial_{xx}h)_{23} \\ h_{30} & h_{31} & h_{32} & h_{33} \end{pmatrix},$$

y por lo tanto estamos calculando la derivada para h_{10}, h_{20}, h_{13} y h_{23} que son también contornos de nuestro problema donde no tiene sentido calcular derivadas. Luego, procediendo de manera análoga para el término $\mathbb{H}\,\mathbb{D}_{yy}^{\intercal}$ y sumando ambas expresiones tendremos

$$\mathbb{D}_{xx}\mathbb{H} + \mathbb{H}\,\mathbb{D}_{yy}^{\intercal} \approx \begin{pmatrix} 2h_{00} & h_{01} + (\partial_{yy}h)_{01} & h_{02} + (\partial_{yy}h)_{02} & 2h_{03} \\ h_{10} + (\partial_{xx}h)_{10} & (\nabla^2 h)_{11} & (\nabla^2 h)_{12} & h_{13} + (\partial_{xx}h)_{13} \\ h_{20} + (\partial_{xx}h)_{20} & (\nabla^2 h)_{21} & (\nabla^2 h)_{22} & h_{23} + (\partial_{xx}h)_{23} \\ 2h_{30} & h_{31} + (\partial_{yy}h)_{31} & h_{32} + (\partial_{yy}h)_{32} & 2h_{33} \end{pmatrix},$$

y por lo tanto, sobre los contornos, $\mathbb{D}_{xx}\mathbb{H} + \mathbb{H}\,\mathbb{D}_{yy}^{\intercal}$ no representa correctamente nuestro problema diferencial.

Sin embargo, si planteamos el problema como $[\breve{\mathbb{D}}_{xx}]_{ik}\breve{\mathbf{h}}_k + [\breve{\mathbb{D}}_{yy}]_{ik}\breve{\mathbf{h}}_k$, esto es fácil de solucionar, ya que contamos con la libertad suficiente para definir cómo operar cada elemento. Luego, alcanza con igualar a cero en $\breve{\mathbb{D}}_{xx}$ las filas que operan sobre h_{i0} y h_{i3} (sin incluir las esquinas), resultando en un nuevo operador $\breve{\mathbb{D}}'_{xx}$ de forma que

$$\breve{\mathbb{D}}'_{xx}\breve{\mathbf{h}} \approx \begin{pmatrix} h_{00} & h_{01} & h_{02} & h_{03} & 0 & (\partial_{xx}h)_{11} & (\partial_{xx}h)_{12} & 0 & 0 & (\partial_{xx}h)_{21} & (\partial_{xx}h)_{22} & 0 & h_{30} & h_{31} & h_{32} & h_{33} \end{pmatrix}^{\intercal}$$

y procediendo análogamente en $\breve{\mathbb{D}}_{yy}$ con las filas que operan sobre h_{0j} y h_{3j} (nuevamente excluyendo a las esquinas), tendremos

$$\breve{\mathbb{D}}'_{xx}\breve{\mathbf{h}} + \breve{\mathbb{D}}'_{yy}\mathbb{H} \approx \begin{pmatrix} 2h_{00} & h_{01} & h_{02} & 2h_{03} & h_{10} & (\partial_{xx}h)_{11} & (\partial_{xx}h)_{12} & h_{13} & h_{20} & (\partial_{xx}h)_{21} & (\partial_{xx}h)_{22} & h_{23} & 2h_{30} & h_{31} \end{pmatrix}$$

y por lo tanto los puntos interiores siguen estimando el laplaciano, mientras que los exteriores estiman correctamente los bordes, excepto sobre las esquinas.

De forma general, las esquinas requieren un poco más de cuidado. Sin embargo, si tenemos un problema de tipo Dirichlet las mismas no propagarán información hacia el interior del dominio y por lo tanto el esquema funcionará adecuadamente.

Por otro lado, si se tienen condiciones de tipo Neumann (en cuyo caso las matrices $\mathbb{D}_{xx}$ y $\mathbb{D}_{yy}$ contienen estimadores de la derivada primera en los extremos) la estrategia de igualar a cero filas en $\check{\mathbb{D}}_{xx}$ y $\check{\mathbb{D}}_{yy}$ sigue funcionando adecuadamente para todos los puntos, excepto las esquinas. **En la medida que utilicemos diferencias finitas hacia adelante cerca de $x = 0$ e $y = 0$ y atrasadas cerca de $x = L_x$ e $y = L_y$**[†], una forma de lograr solucionar este problema, es desacoplar explícitamente las esquinas del resto del dominio. Esto podemos hacerlo imponiendo sencillamente

$$\alpha h_{00} = a, \qquad \beta h_{03} = b, \qquad \gamma h_{30} = c, \qquad \delta h_{33} = d,$$

para valores de α, β, γ, δ, a, b, c y d cualesquiera. De esta forma las esquinas quedan explícitamente desacopladas del resto del problema para cualquier condición que tengamos sobre los contornos y podemos removerlas luego de resolver el sistema lineal (también podríamos eliminar completamente las ecuaciones asociadas a las esquinas, aunque el álgebra de índices para aplastar nuestro problema se volvería más tedioso).

Vale mencionar que la forma de tratar los contornos que mencionamos en esta subsección resulta bastante general, pudiéndose aplicar a dominios con una dirección periódica y una no-periódica. Por ejemplo, si tenemos condiciones de Dirichlet en x y periódicas en y, basta con utilidar $\check{\mathbb{D}}'_{xx}$ (notar el primado) junto con $\check{\mathbb{D}}_{yy}$.

En la sección de *Funciones útiles* del apunte presentamos una función operador_flatten que realiza todas las tareas mencionadas en esta sección, inclusive el tratamiento de los bordes. En caso que opten por resolver el Problema 5, les recomendamos estudiar las salidas de esta función para una cantidad reducida de puntos de grilla a fines de entender qué ecuación algebraica se está planteando sobre cada punto del espacio, en particular cuando el argumento opcional `borde` posee un valor `True`.

[†]: La biblioteca que presentaremos en la próxima sección genera matrices que efectivamente verifican esta propiedad.

1.3.5 Aplanado en Numpy

Veamos ahora como aplanar un arreglo bidimensional en Numpy mediante la instrucción `ndarray.flatten()`, donde `ndarray` es una referencia a la matriz en cuestión.

[ ]:
```
import numpy as np

NX, NY = 3, 4

A = np.arange(0,NX*NY).reshape( (NX,NY) )
print(f"A =\n{A}")

A_aplanado = A.flatten()
print(f"A_aplanado =\n{A_aplanado}")

A_aplanado2 = 2*A_aplanado
print(f"2*A_aplanado =\n{A_aplanado2}")
```

```
A2 = A_aplanado2.reshape( (NX,NY) )
print(f"2*A_aplanado como matriz =\n{A2}")
```

```
A =
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
A_aplanado =
[ 0  1  2  3  4  5  6  7  8  9 10 11]
2*A_aplanado =
[ 0  2  4  6  8 10 12 14 16 18 20 22]
2*A_aplanado como matriz =
[[ 0  2  4  6]
 [ 8 10 12 14]
 [16 18 20 22]]
```

Vemos también que luego podemos utilizar `ndarray.reshape( forma )` para devolver el vector a la forma original (en nuestro caso, una matriz).

Por otra parte, les proporcionamos al final de este apunte una función `operador_arreglo_aplanado` que permite obtener $\check{\mathbb{D}}$ a partir de $\mathbb{D}$. Podrán utilizarla para los ejercicios en que les parezca relevante. Diferiremos un ejemplo con su uso hasta la próxima sección.

1.4 Biblioteca *findiff*

Hasta ahora tratamos los problemas con contornos siempre a orden 2, mientras que los periódicos los tratamos hasta a orden 8 (y fácilmente podríamos aumentar el orden). Esto se debe a que la construcción de matrices de diferenciación con contornos es ligeramente más tediosa cuando aumentamos el orden.

Como mostramos en la sección anterior, la estrategia con diferencias finitas suele ser utilizar esquemas adelantados cerca de uno de los extremos, luego un esquema centrado en la mayor parte del dominio y finalmente un esquema atrasado cerca del extremo restante. Para abstraer esta complejidad, vamos a usar la biblioteca *findiff*. Esta biblioteca no viene incluidad en Google Colab por defecto, pero podemos instalarla fácilmente mediante la siguiente instrucción:

[ ]:
```
!pip install findiff
```

En caso que quieran colocar esa instrucción en una celda junto con otro código, les recomendamos usar la siguiente versión:

[ ]:
```
!if ! pip list | grep findiff >> /dev/null; then pip install findiff; fi
```

que instala *findiff* solo si no fue ya instalado durante la sesión actual (y ganando unos segundos en cada ejecución de la celda).

1.4.1 Obtención de matrices de diferenciación

Para obtener matrices de diferenciación, debemos primero crear un operador de diferencias finitas, mediante la clase `FinDiff`. Para ello, debemos especificar sobre qué eje opera este operador (será 0 en todos los casos que usaremos), el espaciamiento entre puntos, el orden de la derivada y, opcionalmente, el orden de precisión buscado (que debe ser par). **Noten que `FinDiff` no soporta generar operadores para dominios periódicos.**

Veamos en acción a un operador que calcula la derivada segunda

[ ]:
```
!if ! pip list | grep findiff >> /dev/null; then pip install findiff; fi
import numpy as np
from findiff import FinDiff

NX = 16
x, dx = np.linspace(0, 1, NX, retstep=True)

f   =   x*np.sin(2*x)                  # Función
fpp = 4*(np.cos(2*x) - x*np.sin(2*x))  # Derivada segunda

# Creo un operador para la derivada segunda, que opera sobre el eje 0
# con espaciamiento `dx` y de 6to orden de precisión.
d2_dx2 = FinDiff(0, dx, 2, acc=4)

print("Máx. error en la estimación numérica:", np.max(np.abs(fpp - d2_dx2(f))))
```

```
Máx. error en la estimación numérica: 0.002725501376493522
```

Para ver mejor que es lo que está haciendo `FinDiff`, podemos obtener una representación matricial del operador utilizando el método `.matrix(forma)` sobre el mismo. Esto devuelve una matriz rala, por lo que debemos usar `toarray()` para imprimirla en la pantalla. Veamos entonces el resultado

[ ]:
```
with np.printoptions(linewidth=100):  # Para controlar el ancho de la impresión
    print(d2_dx2.matrix((NX,1)).toarray()*12*dx**2)         # Imprimo la matriz
```

```
[[  45. -154.  214. -156.   61.  -10.    0.    0.    0.    0.    0.    0.    0.
0.    0.    0.]
 [   0.   45. -154.  214. -156.   61.  -10.    0.    0.    0.    0.    0.    0.
0.    0.    0.]
 [  -1.   16.  -30.   16.   -1.    0.    0.    0.    0.    0.    0.    0.    0.
0.    0.    0.]
 [   0.   -1.   16.  -30.   16.   -1.    0.    0.    0.    0.    0.    0.    0.
0.    0.    0.]
 [   0.    0.   -1.   16.  -30.   16.   -1.    0.    0.    0.    0.    0.    0.
0.    0.    0.]
 [   0.    0.    0.   -1.   16.  -30.   16.   -1.    0.    0.    0.    0.    0.
0.    0.    0.]
 [   0.    0.    0.    0.   -1.   16.  -30.   16.   -1.    0.    0.    0.    0.
0.    0.    0.]
```

```
 [   0.    0.    0.    0.    0.   -1.   16.  -30.   16.   -1.    0.    0.    0.
0.    0.    0.]
 [   0.    0.    0.    0.    0.    0.   -1.   16.  -30.   16.   -1.    0.    0.
0.    0.    0.]
 [   0.    0.    0.    0.    0.    0.    0.   -1.   16.  -30.   16.   -1.    0.
0.    0.    0.]
 [   0.    0.    0.    0.    0.    0.    0.    0.   -1.   16.  -30.   16.   -1.
0.    0.    0.]
 [   0.    0.    0.    0.    0.    0.    0.    0.    0.   -1.   16.  -30.   16.
-1.    0.    0.]
 [   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.   -1.   16.  -30.
16.   -1.    0.]
 [   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.   -1.   16.
-30.   16.   -1.]
 [   0.    0.    0.    0.    0.    0.    0.    0.    0.  -10.   61. -156.  214.
-154.   45.    0.]
 [   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  -10.   61. -156.
214. -154.   45.]]
```

Notemos que la forma resultó sencillamente `(NX,1)` para una matriz compatible con un vector de longitud `NX` (si olvidan el `,1` en la forma, *findiff* reportará un error).

Con esto será suficiente para poder resolver problemas con contorno con órden de aproximación ajustable sin tener que reparar en la construcción de las matrices de diferenciación.

1.4.2 Sobre *findiff*

Vale remarcar que, aunque útil para esta práctica, la biblioteca *findiff* posee una cantidad de desarrolladores relativamente pequeña que desarrollan la biblioteca como pasatiempo. Es por ello que, aunque la hemos encontrado correcta para la funcionalidad mencionada más arriba, no podemos garantizarles que todas las funcionalidades de *findiff* funcionen apropiadamente, ni que la misma siga disponible en el futuro.

Dicho eso, tal vez valga la pena mencionarles que *findiff* permite diferenciar arreglos a lo largo de un eje arbitrario, inclusive sobre grillas no uniformes. Más aún, incorpora varias herramientas útiles para calculo vectorial, como operadores de gradiente, divergencia, rotor y laplaciano. También incorpora un módulo para resolver EDPs, aunque no podemos recomendar su uso ya que parece hallarse en desarrollo activo y solo tiene implementado funcionalidades bastante elementales con respecto a otras bibliotecas. Pueden consultar toda la funcionalidad en la documentación del proyecto.

2 Funciones útiles

Adjuntamos a continuación algunas funciones que pueden resultarles de utilidad para la realización de esta Práctica o el análisis de los resultados obtenidos.

2.0.1 Matrices de diferenciación para condiciones de contorno periódicas

[ ]:
```
def diferenciacion_centrada_periodica(N, d, orden=1, precision=2):
    """
    Devuelve una representación rala de la matriz de diferenciación que
    aproxima a la derivada de un cierto orden. Puede devolver esquemas con
    distintos órdenes de precisión.

    Entrada:
        - `N`:         Cantidad de puntos a diferenciar.
          (entero)
        - `d`:         Espaciamiento entre puntos.
          (flotante)
        - `orden`:     Orden de la derivada a aproximar.
          (entero)
        - `precision`: Orden de precisión del aproximante utilizado.
          (entero)

    Salida:
        - `D`: Representación rala de la matriz de diferenciación.
          (`scipy.sparse.dia.dia_matrix`)
    """
    from scipy.sparse import diags

    if precision > N-1:
        raise ValueError("Cantidad de puntos insuficiente para"
                         " la precisión requerida.")

    # Derivada primera
    if orden == 1:
        if precision == 2:
            coefs = [ [-1], [0], [1] ]
            fact  = 1/2
        elif precision == 4:
            coefs = [ [1], [-8], [0], [8], [-1] ]
            fact  = 1/12
        elif precision == 6:
            coefs = [ [-1], [9], [-45], [0], [45], [-9], [1] ]
            fact  = 1/60
        elif precision == 8:
            coefs = [ [3], [-32], [168], [-672], [0], [672], [-168], [32], [-3] ]
            fact  = 1/840
        else:
            raise ValueError("Orden de precisión inexistente o"
                             " no implementado.")

        fact *= 1/d
```

```
    # Derivada segunda
    elif orden == 2:
        if precision == 2:
            coefs  = [ [1], [-2], [1] ]
            fact   = 1
        elif precision == 4:
            coefs  = [ [-1], [16], [-30], [16], [-1] ]
            fact   = 1/12
        elif precision == 6:
            coefs  = [ [2], [-27], [270], [-490], [270], [-27], [2] ]
            fact   = 1/180
        elif precision == 8:
            coefs  = [ [-9], [128], [-1008], [8064], [-14350] ]
            coefs += [ [8064], [-1008], [128], [-9] ]
            fact   = 1/5040
        else:
            raise ValueError("Orden de precisión inexistente o"
                             " no implementado.")

        fact *= 1/d**2
    else:
        raise ValueError("Orden de derivación inexistente o no implementado.")

    # Periodicidad
    l        = len(coefs)
    coefs   += coefs[:l//2] + coefs[l//2+1:]
    offsets  = list(range(-l//2+1, l//2+1))
    offsets += [  N + offsets[i] for i in range(0     , l//2) ]
    offsets += [ -N + offsets[i] for i in range(l//2+1,    l) ]

    return fact*diags(coefs, offsets=offsets, shape=(N,N))
```

2.0.2 Creación de matrices que operan sobre arreglos 2D aplanados

[ ]:
```
def operador_arreglo_aplanado(D, nx, ny, eje=None, borde=False):
    """
        Construye una versión ampliada (y rala) del operador D, Ď, de forma que
    tal que (D @ X).flatten() = Ď @ (X.flatten()), si `eje="x"`, o bien
    (X @ D.T).flatten() = (X.flatten()) @ Ď para `eje="y"`. En ambos casos X es
    una matriz arbitraria.

    Entrada:
        -`D`:     Operador sobre la forma matricial de un conjunto de datos.
        -`nx`:    Cantidad de filas de la matriz contra la que opera D.
```

```
    -`ny`:    Cantidad de columnas de la matriz contra la que opera D.
    -`eje`:   Caractér "x" o "y":
        * Si `eje="x"` (D @ X).flatten() = Ď @ (X.flatten()).
        * Si `eje="y"` (X @ D.T).flatten() = (X.flatten()) @ Ď.
    -`borde`: Si en la dirección contraria hay presente un borde o no.

Salida:
    -`Ď`: Operador sobre la forma aplastada de un conjunto de datos.
"""
import scipy.sparse as spsparse

if eje is None or eje not in ["x", "y"]:
    raise ValueError("Es necesario especificar si D opera sobre el primer "
                     "eje (eje='x') o sobre el segundo (eje='y').")

if (eje == "x" and D.shape[0] != nx) or (eje == "y" and D.shape[0] != ny):
    raise ValueError("El tamaño de D y  el tamaño ampliado solicitado "
                     "no coinciden.")

l   = nx*ny
ret = spsparse.lil_matrix((l,l))

#  Si el operador actuaba sobre las columnas en cada fila de D cada par de
# valores de se rellena con ny-1 ceros en el medio y repite para ny filas
# consecutivas consecutivas desplazándose 1 columna para la derecha cada vez.
if eje == "x":
    for i in range(nx):
        for j in range(ny):
            ret[ny*i+j,j::ny] = D[i]

    #  Si el problema tiene contornos no debe operar sobre la primer y
    # última columna.
    if borde:
        for i in range(0,nx):
            ret[ny*i]       = 0
            ret[ny*(i+1)-1] = 0

#  Si el operador actua sobre las filas tengo distintas D que se
# concatenan a lo largo de la diagonal.
elif eje == "y":
    for i in range(nx):
        ret[i*ny:(i+1)*ny, i*ny:(i+1)*ny] = D

    #  Si el problema tiene contornos no debe operar sobre la primer ni
    # la última fila
    if borde:
        for i in range(0, ny):
```

```
                ret[i]           = 0
                ret[(nx-1)*ny+i] = 0

    # Dejo la identidad en las esquinas para evitar un sistema indeterminado
    if borde:
        for i in [0, ny-1, (nx-1)*ny, nx*ny-1]:
            ret[i]   = 0
            ret[i,i] = 1

    return ret.tocsr()
```

`lil_matrix` es una clase que permite construir las matrices de forma incremental, usando una sintáxis análoga a aquella de arreglos Numpy. No es del todo eficiente para operar, por eso al final se la convierte a tipo `csr_matrix`.

Noten que existen formas más rápidas que `lil_matrix` de crear esta matriz ampliada, sin embargo esta función priorizó la legibilidad de las operaciones.

2.0.3 Gráfico 2D animado

[ ]:
```
def grafico2d_animado(x, y, escalar, dt, titulo="", etiqueta_x="",
                      etiqueta_y="",etiqueta_escalar="", figsize=None, paso=1):
    """
    Genera un gráfico animado 2D.

    Entrada:
        - `x`:       arreglo 1D (NX) con las abscisas de los datos datos.
        - `y`:       arreglo 1D (NY) con las ordenadas de los datos datos.
        - `escalar`: arreglo 2D (NX,NY) con los valores del campo escalar sobre
                     la grilla cartesiana.
        - `dt`:      paso temporal entre muestras.

        - `titulo`:           string con el título del gráfico [OPCIONAL].
        - `etiqueta_x`:       string con la etiqueta para el eje x [OPCIONAL].
        - `etiqueta_y`:       string con la etiqueta para el eje y [OPCIONAL].
        - `etiqueta_escalar`: string con la etiqueta del campo escalar
                              [OPCIONAL].
        - `figsize`:          2-úpla con el tamaño de la figura.
        - `paso`:             espaciamiento en los datos para cada fotograma.
                              [OPCIONAL]

    Salida:
        - `anim`:   referencia al objeto de animación creado.
    """
```

```
    import numpy as np
    import matplotlib.pyplot as plt
    import matplotlib.animation as animation

    # Guardo el estado de plt
    params_viejos = plt.rcParams
    plt.rc('animation', html='jshtml')

    num_foto =escalar.shape[0]

    fig, ax = plt.subplots(1, 1, figsize=figsize, constrained_layout=True)
    plt.close();  # Cerrar la figura, animation va a crear la suya propia

    # Inicializo las curvas
    plot = ax.imshow( np.ones((x.size, y.size)), extent=(x[0],x[-1],y[0],y[-1]),
                      origin="lower", interpolation='gaussian',
                      vmin=escalar.min(), vmax=escalar.max())

    cbar = fig.colorbar(plot, ax=ax, orientation="horizontal")
    cbar.set_label(etiqueta_escalar)

    ax.set_title(titulo + f" $t=0$")
    ax.set_xlabel(etiqueta_x)
    ax.set_ylabel(etiqueta_y)

    def init():
        """ Inicializador de la figura y gráfico de condiciones iniciales."""
        plot.set_data(escalar[0].T)
        return plot,

    def actualizar(t):
        """ Actualiza los datos al fotograma actual."""
        print(f"\rCalculando fotograma {t//paso} de {(num_foto-1)//paso}",
              end="")
        plot.set_data(escalar[t].T)

        ax.set_title(titulo + f" $t={t*dt:.5f}$")
        return plot,

    anim = animation.FuncAnimation(fig, actualizar, init_func=init,
                                   frames=range(0, num_foto, paso),
                                   blit=True, repeat=True)

    # Restauro el estado de plt
    plt.rc(params_viejos)

    return anim
```