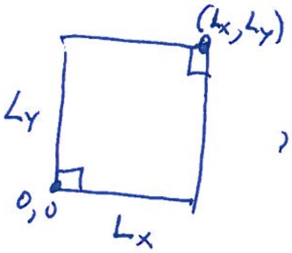
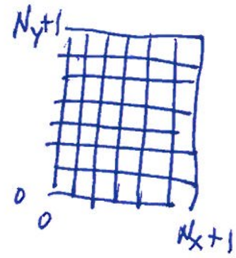
Lecture 10 : Finite differences in 2(+) dimensions

$*$ consider $\hat{A} = \nabla^2$, $\Omega =$  , $u|_{\partial\Omega} = 0$

+ approximate $u(x,y)$ by <u>grid</u> 

$$\Delta x = L_x/(N_x+1)$$

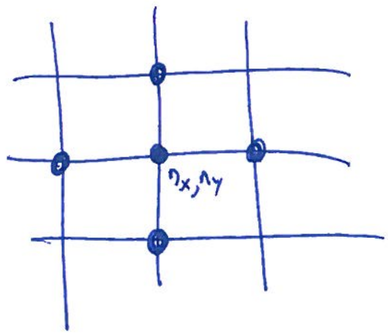
$$\Delta y = L_y/(N_y+1)$$

$$u_{n_x,n_y} \approx u(n_x \Delta x,\ n_y \Delta y), \quad u_{n_x,n_y}\Big|_{n_x=0,N_x+1} = u_{n_x,n_y}\Big|_{n_y=0,N_y+1} = 0$$

$\Longrightarrow$ by usual center-difference approximation:

$$\nabla^2 u\Big|_{n_x,n_y} \approx \frac{u_{n_x+1,n_y} - 2u_{n_x,n_y} + u_{n_x-1,n_y}}{\Delta x^2} + \frac{u_{n_x,n_y+1} - 2u_{n_x,n_y} + u_{n_x,n_y-1}}{\Delta y^2}$$

= "5-point stencil"



$\nabla^2 u\Big|_{n_x n_y}$ determined from 5 grid points (nearest neighbors)

* How do we write this as $A\vec{u}$ for some $\underbrace{(N_x N_y)}_{N} \times \underbrace{(N_x N_y)}_{N}$ matrix A and a vector $\vec{u}$ of $N_x N_y$ unknowns?

— key step: we must "flatten" the 2d array u_{n_x, n_y} into a "1d" vector $\vec{u}$ (components u_n)

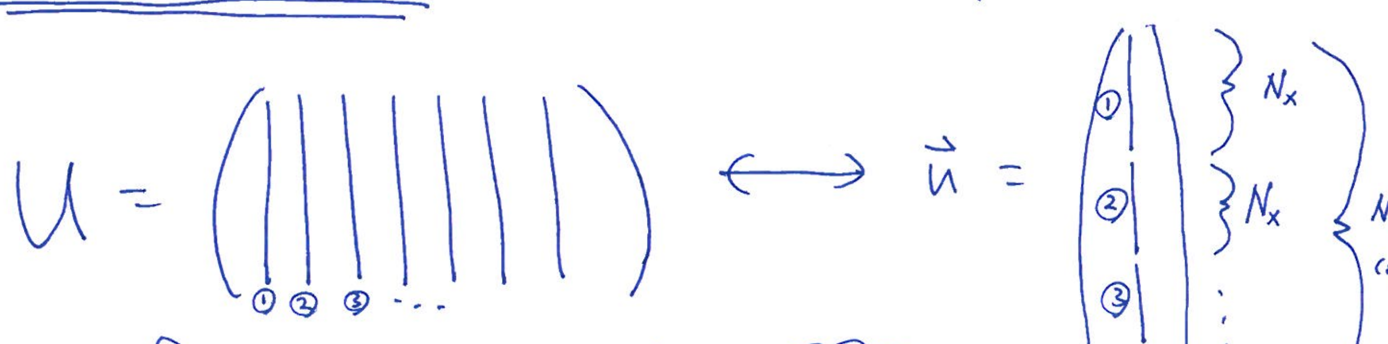
$\Rightarrow$ need a (1-to-1) mapping $(n_x, n_y) \leftrightarrow n$

write $u_{n_x n_y}$ = matrix
$$U = \begin{array}{c c} n_x \downarrow & \begin{pmatrix} \\ N_x \times N_y \\ \\ \end{pmatrix} \\ & \rightarrow n_y \end{array}$$

* multiple ways to "flatten this"

one common choice (Matlab's choice) is

column-major order: $\vec{u}$ = columns of U, in order



$$\left[\, n = n_x + N_x (n_y - 1) \,\right]$$

Matlab:
u= reshape(U,N,1)
U=reshape(u, N_x, N_y)

* constructing A :

— consider $\frac{\partial^2}{\partial x^2}$ of each <u>column</u> $\left(\Big| \, N_x \right)$ of U

$$= \text{1d } 2^{nd}\text{-deriv matrix } A_x = -D_x^T D_x = \frac{1}{\Delta x^2}\begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{pmatrix}$$

$\Longrightarrow \frac{\partial^2}{\partial x^2}$ on $\vec{u}$ does A_x on each N_x block :

$$\begin{pmatrix} A_x & & & \\ & A_x & & \\ & & A_x & \\ & & & \ddots \; A_x \end{pmatrix} \vec{u} = \begin{pmatrix} A_x \Big| \\ A_x \Big| \\ \vdots \end{pmatrix} \begin{matrix} \}\, N_x \\ \}\, N_x \\ \end{matrix}$$

— what about $\frac{\partial^2}{\partial y^2}$? consider $\frac{\partial^2}{\partial y^2}$ of whole columns of U :

($u_{:,n_y}$ in Matlab)

$$\left.\frac{\partial^2}{\partial y^2} u \right|_{n_y} \approx \frac{u_{:,n_y+1} - 2u_{:,n_y} + u_{:,n_y-1}}{\Delta y^2}$$

$$= \frac{\left(\Big| \right)_{n_y+1} - 2 \left(\Big| \right)_{n_y} + \left(\Big| \right)_{n_y+1}}{\Delta y^2}$$

in matrix form:

$$\frac{1}{\Delta y^2}\begin{pmatrix} -2I_x & I_x & & & \\ I_x & -2I_x & I_x & & \\ & \ddots & \ddots & \ddots & \\ & & I_x & -2I_x & I_x \\ & & & I_x & -2I_x \end{pmatrix} \vec{u}$$

like the "1d" matrix $A_y = -D_y^T D_y$

but the <u>entries</u> are <u>matrices</u>:

$I_x = N_x \times N_x$ identity matrix

* <u>Kronecker products</u>: an elegant way to make matrices out of matrices

$$\underset{m \times n}{A} \otimes \underset{p \times q}{B} = \begin{pmatrix} a_{11}B & a_{12}B & \cdots \\ a_{21}B & a_{22}B & \cdots \\ \vdots & \vdots & \end{pmatrix}$$

$$= \begin{pmatrix} a_{11} & a_{12} & \cdots \\ a_{21} & a_{22} & \cdots \\ \vdots & \vdots & \end{pmatrix} \otimes \begin{pmatrix} b_{11} & b_{12} & \cdots \\ b_{21} & b_{22} & \cdots \\ \vdots & \ddots & \end{pmatrix}$$

$mp \times nq$

[in Matlab: $A \otimes B$ = kron(A, B)]

... lots of nice properties + applications,

but especially gives elegant description of "multidimensional matrices" acting on "multidimensional vectors"

...

Here: $\begin{pmatrix} A_x & & & \\ & A_x & & \\ & & \ddots & \\ & & & A_x \end{pmatrix} = I_y \otimes A_x$

N_y times

($N_y \times N_y$ identity with entries $\cdot A_x$)

$$\frac{1}{\Delta y^2}\begin{pmatrix} -2I_x & I_x & & \\ I_x & -2I_x & I_x & \\ & \ddots & \ddots & \ddots \end{pmatrix} = A_y \otimes I_x$$

(A_y matrix with entries $\cdot I_x$)

~~in Matlab~~

... Matlab demo ...

$$\Longrightarrow \boxed{A = I_y \otimes A_x + A_y \otimes I_x}$$

Sparse matrices

* problem: A is huge, $N_x N_y \times N_x N_y$:
  - even $N_x = N_y = 100$ gives $10^4 \times 10^4$ matrix (~ 1 GB)
  - ... and much worse in 3d!
  - — merely storing A is a problem,
  - + solving $Au = f$ takes $\sim N^3$ operations ($\sim$ minutes for $N = 10^4$, $\sim$ years for $N = 10^6$)

* solution: A is mostly zeros (sparse): ≤ 5 entries on each row
  - $\Longrightarrow$ store only nonzero entries
  - + use special $Au = f$ + $Au = \lambda u$ solvers that exploit sparsity (take 18.335)

Matlab: $A_x \to$ sparse(A_x) etc.
u = A\f, eigs(A)