

Today: sparse matrices.

PDEs.  
(linear case).  $\partial_t u + Au = f$   
↑  
eg.  $A = -\nabla^2$   
~heat equation~.

$A: V \rightarrow W$   $A$  is an operator  
between vector spaces.

On a computer, all these spaces  
are finite dimensional.

$\dim V < \infty$  ;  $\dim W < \infty$ .

eg:  $A: \mathbb{R}^n \rightarrow \mathbb{R}^n$

And: all finite dimensional linear  
operators can be represented  
by matrices

(We pick a basis for our vector  
space).

Typically pick a basis s.t.

the matrix representation of  $A$ ,  
which we'll call  $A$ , is sparse.

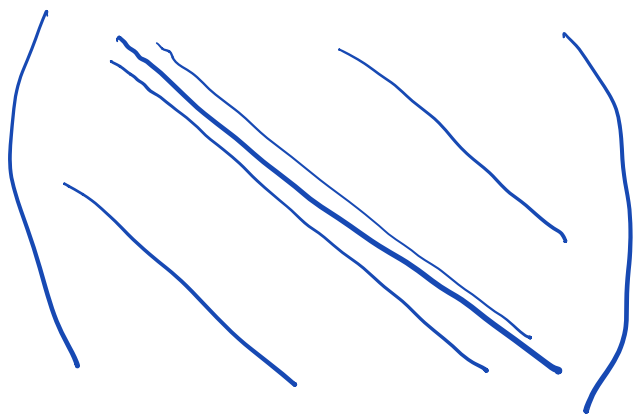
$A \in \mathbb{R}^{n \times n}$  real-value, sparse  
matrices.

And # nonzeros in  $A$

is  $O(n)$ , not  $O(n^2)$ .

Only a constant # of nonzero  
entries per row of  $A$ .

eg  $D^2$   Then each row  
has 5 non-zeros.



Want to store our matrices  
taking advantage of this sparsity.

Formats.

Dense: <sup>data</sup> array, n rows, n cols. (n rows x n cols)  
storage.

Sparse: nonzeros, [(row, col)]

↓  
nz nonzeros      nz "coordinates"  
that say where in  
the matrix the nonzero  
is.

$\begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 3 \\ 0 & 0 & 4 \end{bmatrix}$       nz: [1, 2, 3, 4]  
coords: [(0,1), (1,0), (1,2), (2,2)]  
"COO" format.

Sparse matrices are not friendly for  
high performance compute.

→ "unstructured".

Goal: reduce the amount of coordinate  
data I need to store.

Usual: run-length encode row indices.

" $AIJ$ " or " $CSR$ "

↓  
compressed sparse row.

nonzero array. #nz

row indices  $\rightarrow$  rind #rows + 1

col indices #nz.

$rind[i : i+1]$  says which entries  
in nonzero array  
are in row  $i$ .

\* where to get column indices  
from.

$\begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 3 \\ 0 & 0 & 4 \end{bmatrix}$     nz = 1, 4, 3, 4  
rind = (0, 1, 3, 4)  
col = 1, 0, 2, 2

eg row 1: slice

has  $nz[1:3] = 2, 3$

$col[1:3] = 0, 2$

Zoo of formats:

Wat interface that's agnostic.

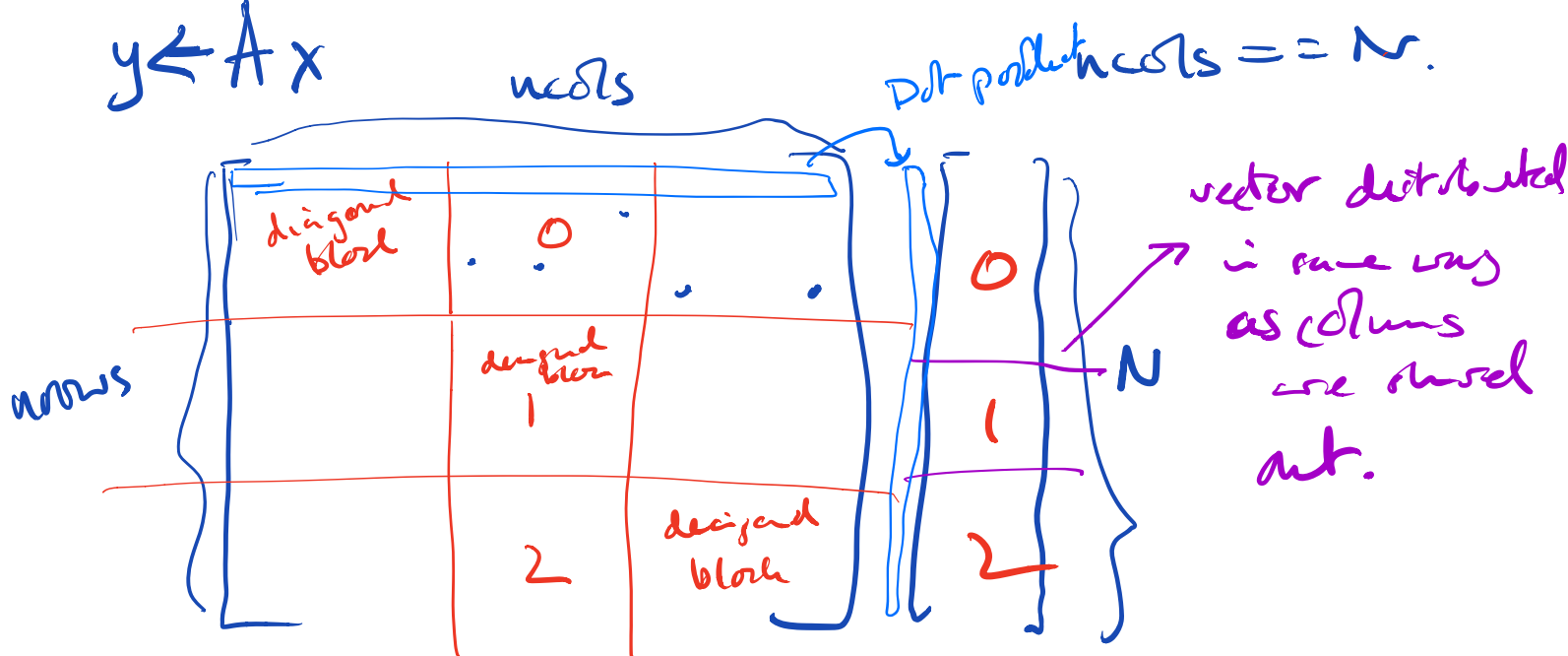
Petsc gives us this with its Mat type. Provides a bunch of implementations.

- AIJ.
- Dense
- Block-AIJ
- ⋮

## Parallel

Matrix-vector products.

$$y \leftarrow Ax$$



Distribute by row. Split load per of

matrix with diagonal block and "off-diagonal block".

Mat-vec: rows of  $A$  dotted against  $x$ .

For the "diagonal" block, the relevant vector entries are local.  
→ due to computable layout.

Implementation  $\hat{=}$ :

Do rendezvous to figure out which remote vector entries I need.

→ Communicate with neighboring nodes to get them.

Do my local multiplications

Finish comms & do the "remote" multiplications.

$A \cdot \text{mult}(x, y) \leftarrow$  PetSc does all this comms.