Wednesday, 12/21/11:

Representation of Matrices in C:

Perspective of Pacheco = standard C perspective

```
double A[10][10]   or
double A[n][n]
```

How are the data actually organised?

dynamic way:

```
double **A
A = (double**)malloc(n * sizeof(double*))
for (i=0; i<n; i++)
  A[i] = (double*)malloc(n * sizeof(double))      <- row
```

⇒ elements A[i][j] = A_{ij}.

⇒ the malloc's are all independent, so data are not consecutive in memory!

The same is true for A[n][n].

Also notice: C organizes matrices by row = row-major ordering. This is incompatible with Fortran and Matlab and thus also all major numerical libraries. More subtly, the double-pointer in A[n][n] is a problem, since no other language perceives arrays in this way. A[i][j] is simply not the same as A[i,j].

Therefore, use 1-D arrays for all arrays including 2-D matrices or higher-dimensional arrays and then index yourself manually.

```
double *A
A = (double*) malloc(n*n* sizeof(double))
```

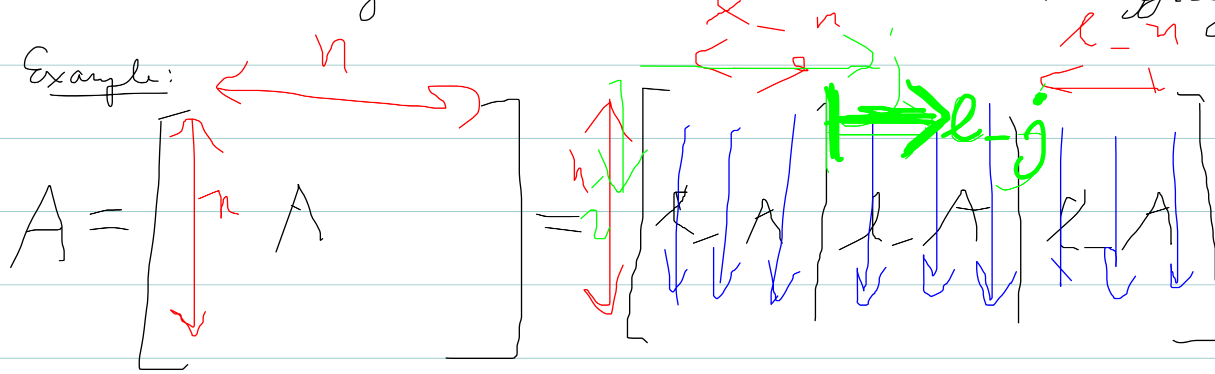
Use it as $A[i + n*j] = A_{ij}$ (or $A[i+n*j+n*n*k] = A_{ijk}$)

This is <u>column-major</u> ordering, since $A[i+1+n*j]$ is next to $A[i+n*j]$ in memory, which is $A_{i+1,j}$ being next to A_{ij}.

$\Rightarrow$ This way we are in complete control of where our data are!

This is vital to using MPI communication commands and to efficient code.

Example:



We divide into blocks of columns to accomodate the column-major ordering, since then also l_A are consecutive in memory.

$\Rightarrow$ Now we can do a single MPI_Gather to collect all l_A into A on Process 0 for fast output ($\to$ HW (a))

```
MPI_Gather(l_A, n*l_n, MPI_DOUBLE, A, n*l_n, MPI_DOUBLE,
           0, MPI_COMM_WORLD)
```

Clarify:

$j = l\_j + id * l\_n$

```
double *l_A
l_A = allocate_double_vector(n*l_n)
for (l_j<0; l_j<l_n; l_j++)
  for (i=0; i<n; i++)
    l_A[i+n*l_j] = A_ij
```