Istituto Nazionale di
Geofisica e Vulcanologia

MATLAB & Practical Applications on Climate Variability Studies tutorial

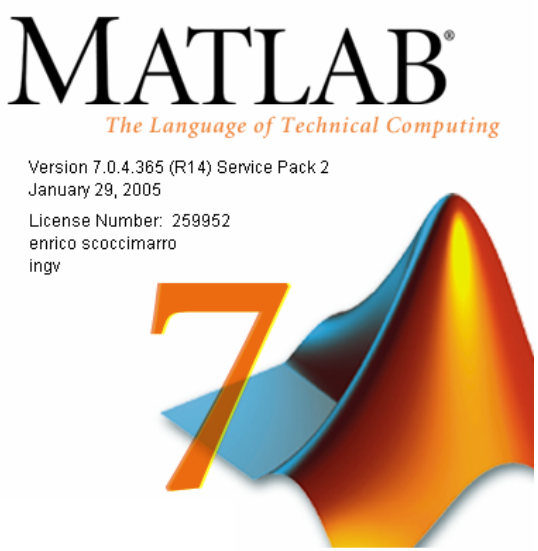
B.Aires, 20-24/02/06

Centro de Investigaciones del Mar y la Atmosfera &
Department of Atmospheric and Oceanic Sciences (UBA)
E.Scoccimarro, A.F.Carril

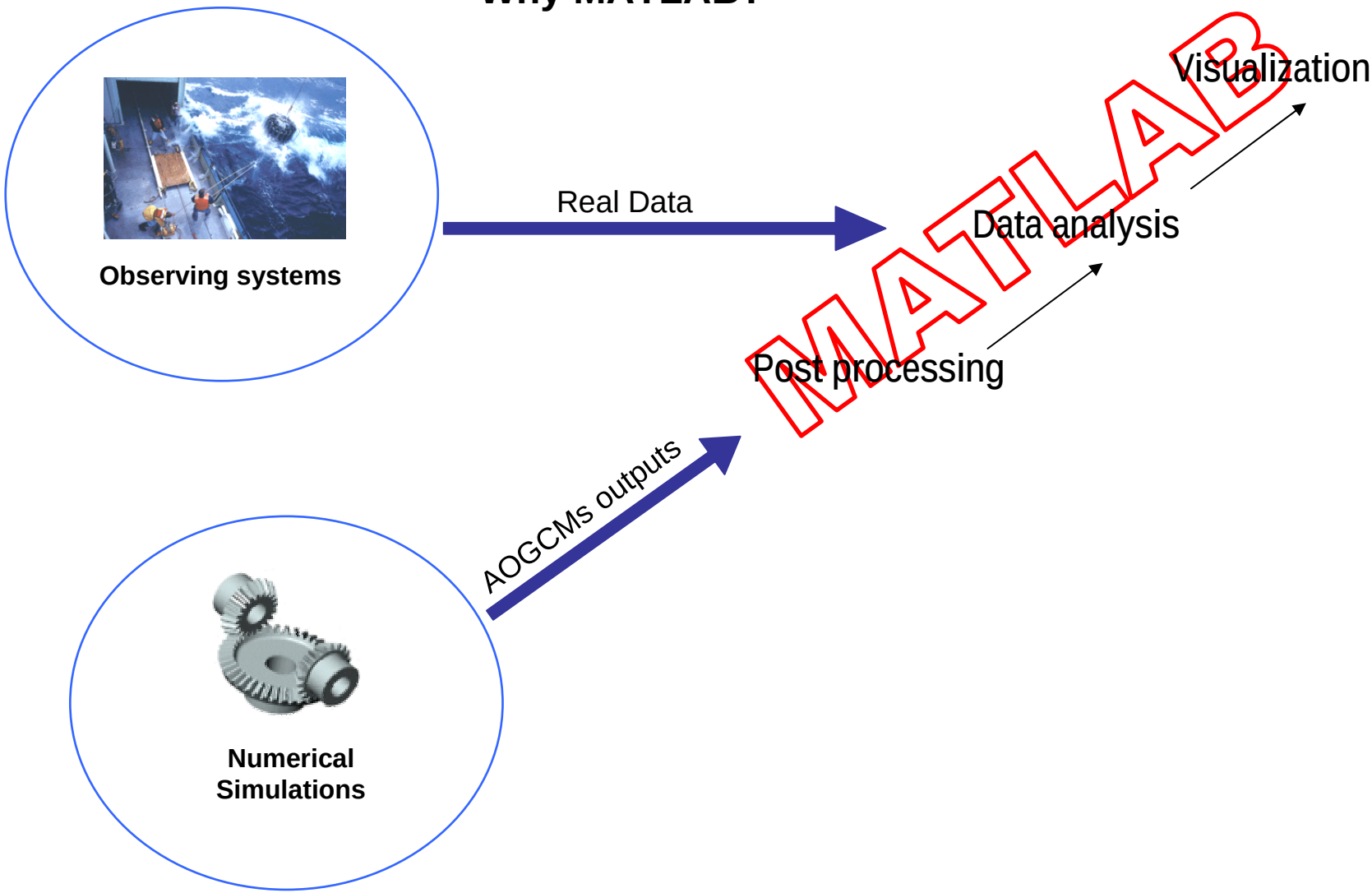
What is MATLAB?

Matlab is a contraction for "Matrix Laboratory" and, though originally designed as a tool for the manipulation of matrices, is now capable of performing a wide range of numerical computations.

Matlab also possesses extensive graphics capabilities.



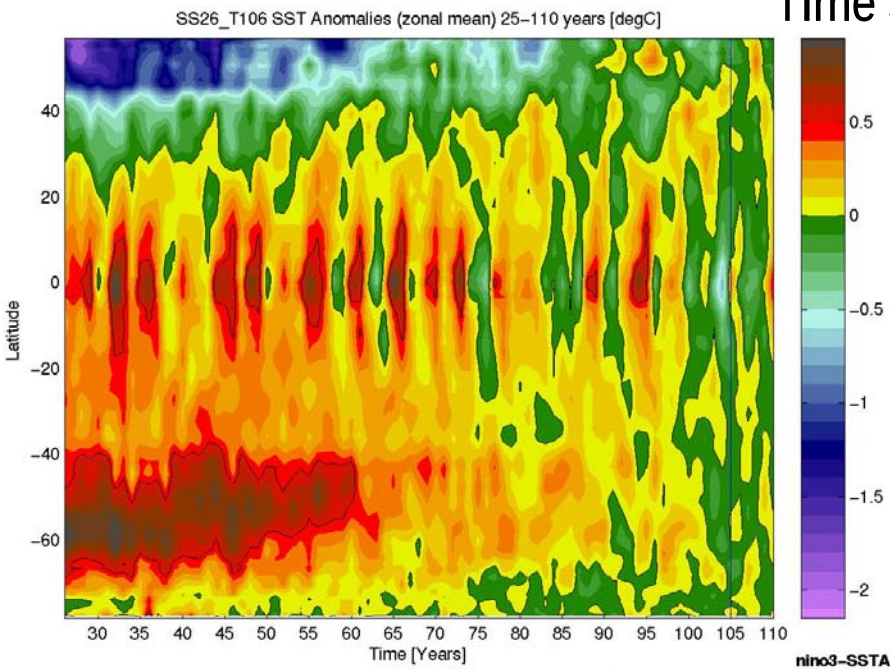
Why MATLAB?

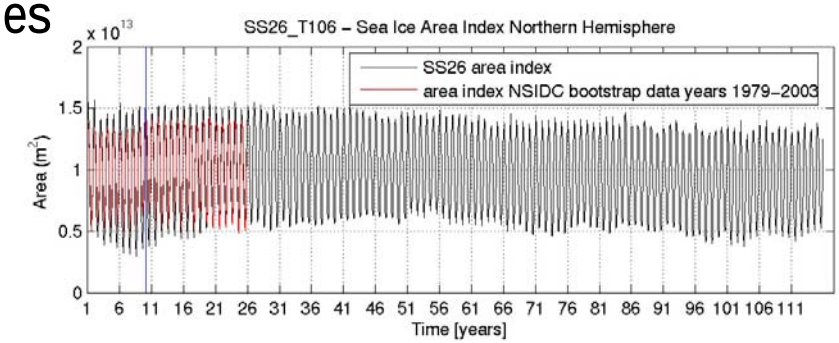

Observing systems
Real Data
Numerical Simulations
AOGCMs outputs
MATLAB
Post processing
Data analysis
Visualization

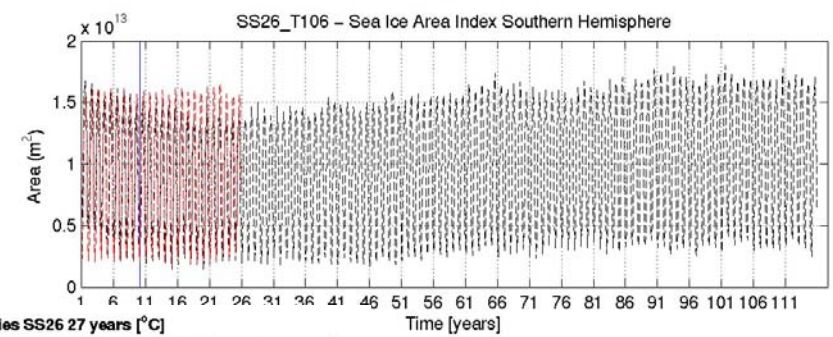
MATLAB & Practical Applications on Climate Variability Studies day 1

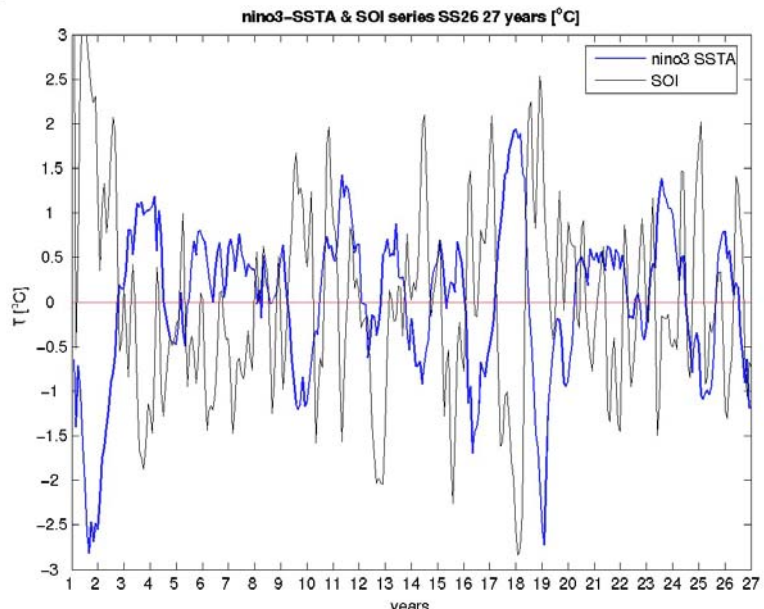
Istituto Nazionale di Geofisica e Vulcanologia

Time series





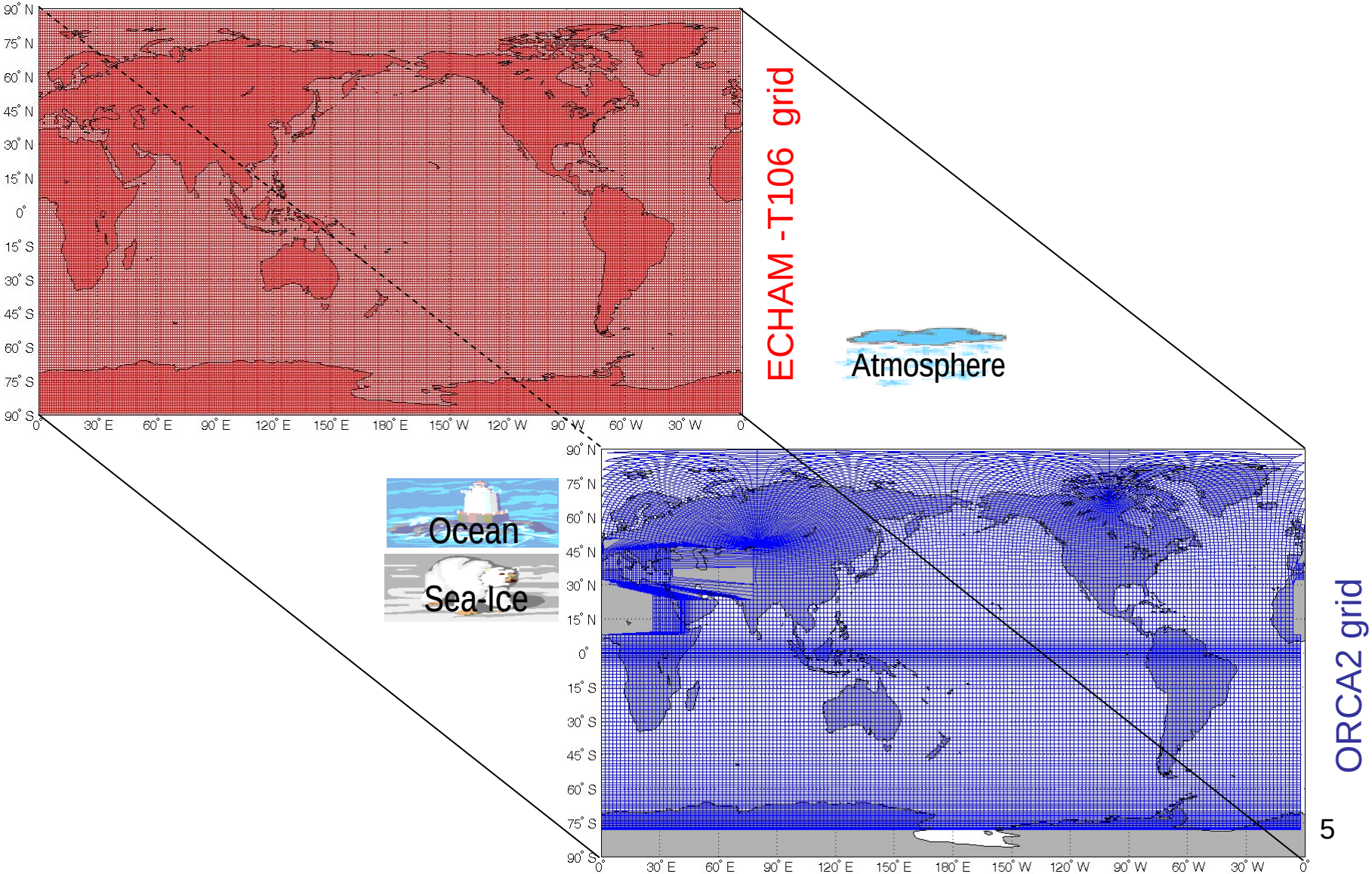




MATLAB & Practical Applications on Climate Variability Studies day 1

Istituto Nazionale di Geofisica e Vulcanologia

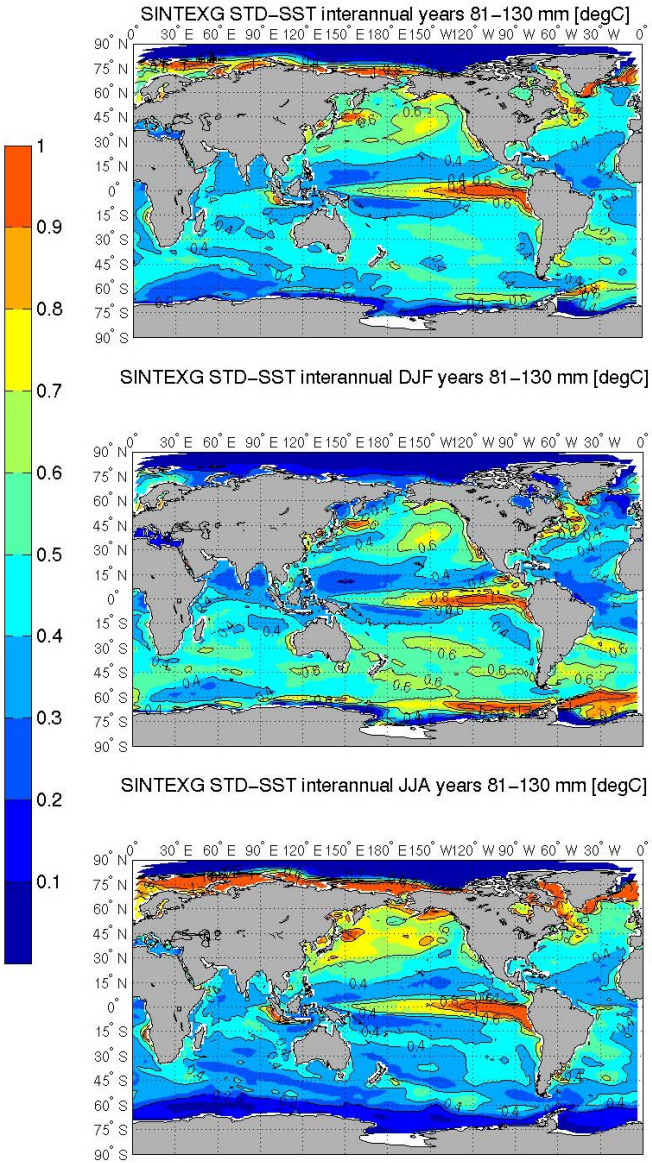
Fields interpolation

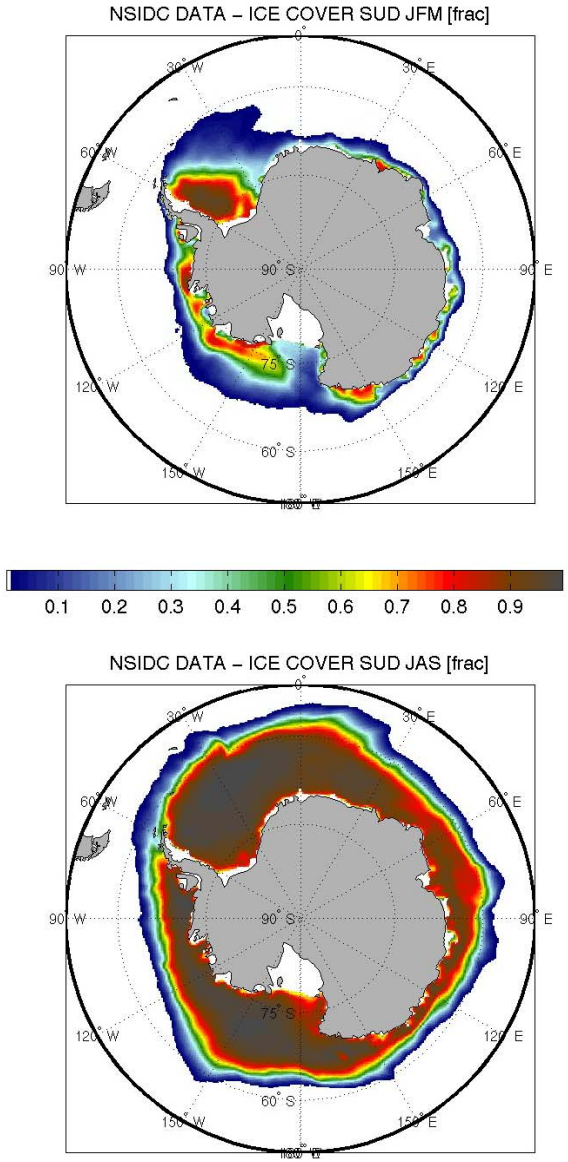


MATLAB & Practical Applications on Climate Variability Studies day 1

Istituto Nazionale di Geofisica e Vulcanologia

Fields Visualization

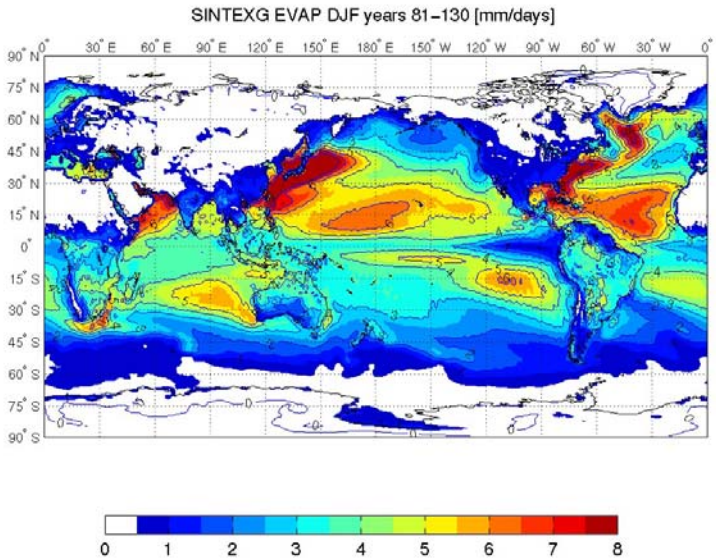


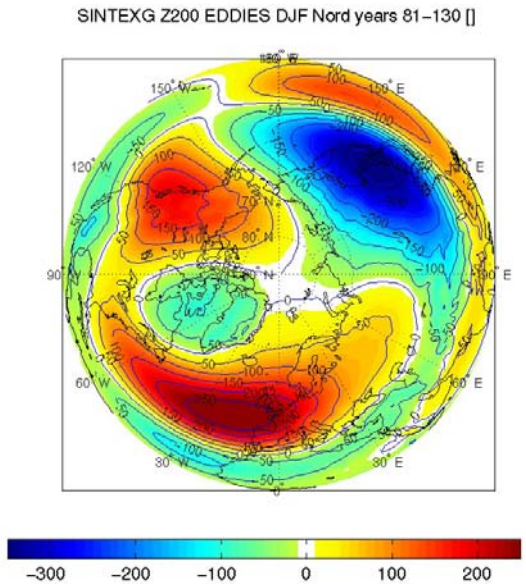


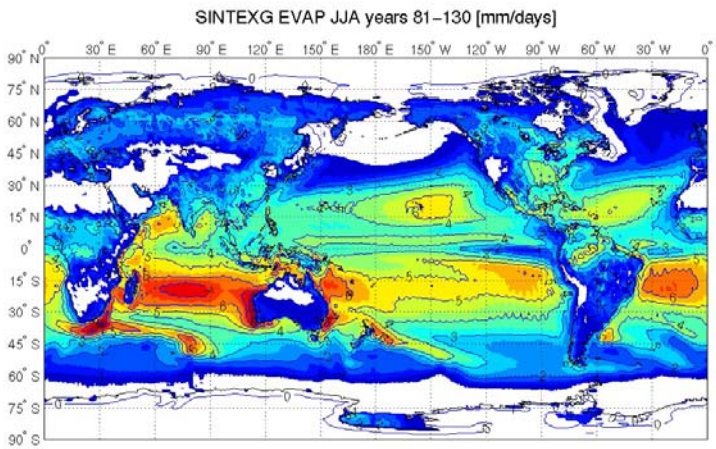
MATLAB & Practical Applications on Climate Variability Studies day 1

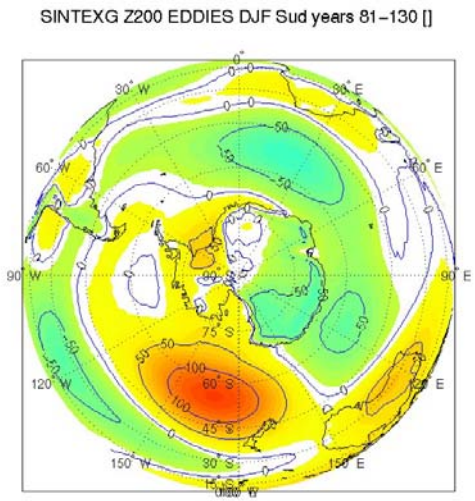
Istituto Nazionale di Geofisica e Vulcanologia

Fields Visualization





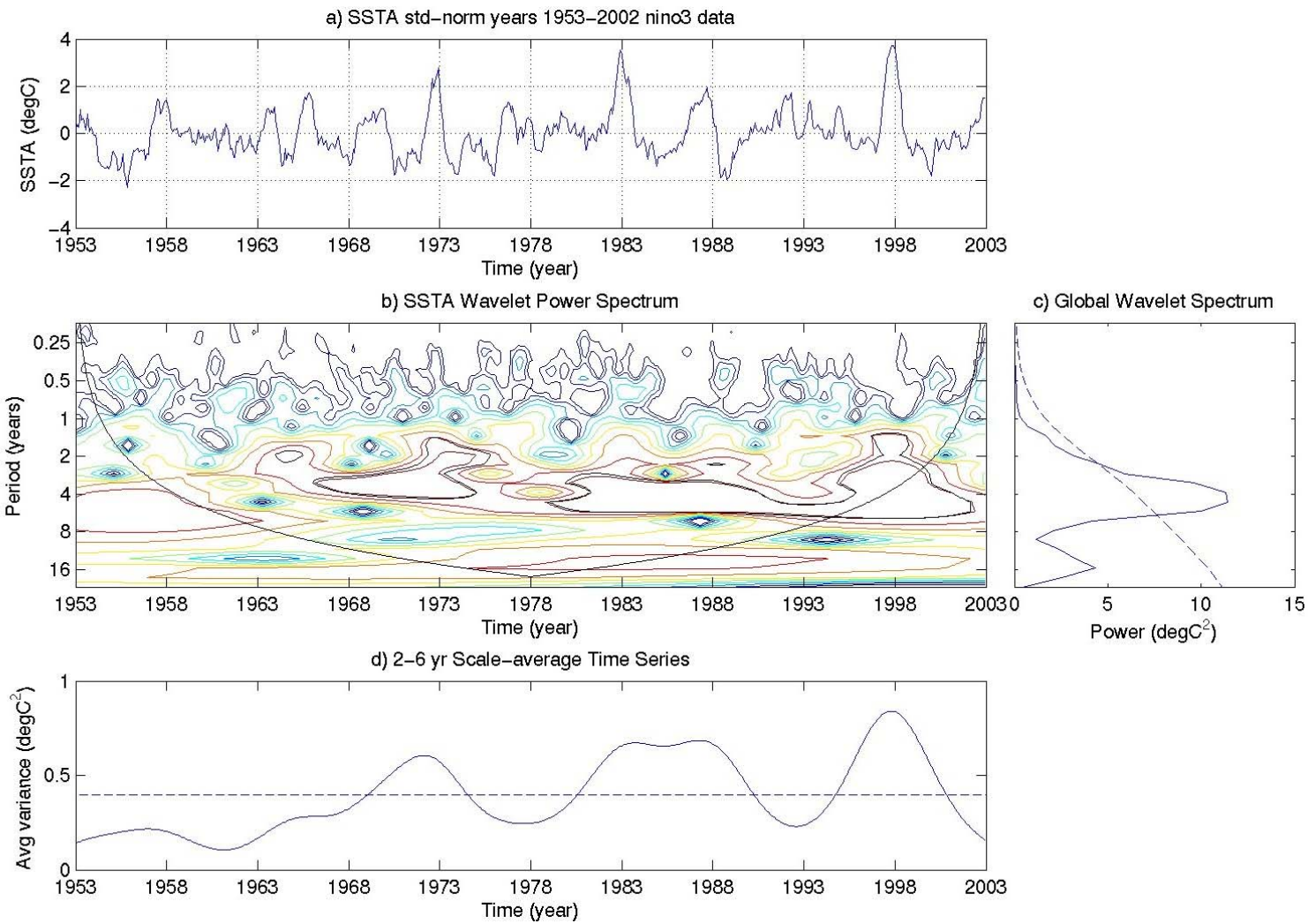




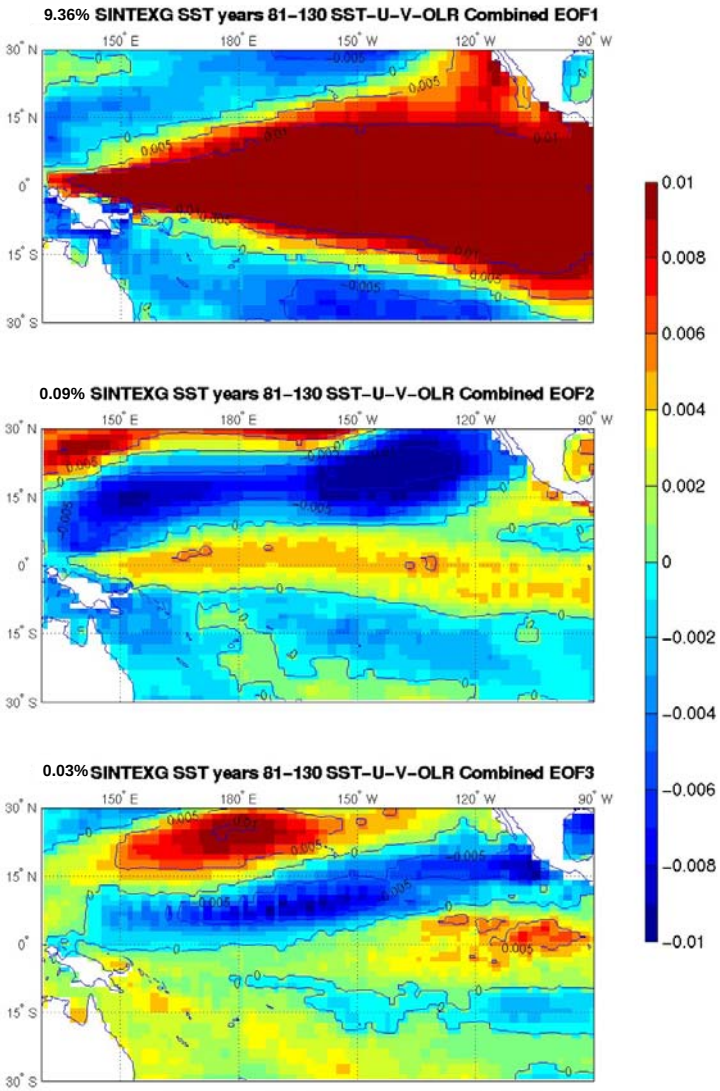

movie

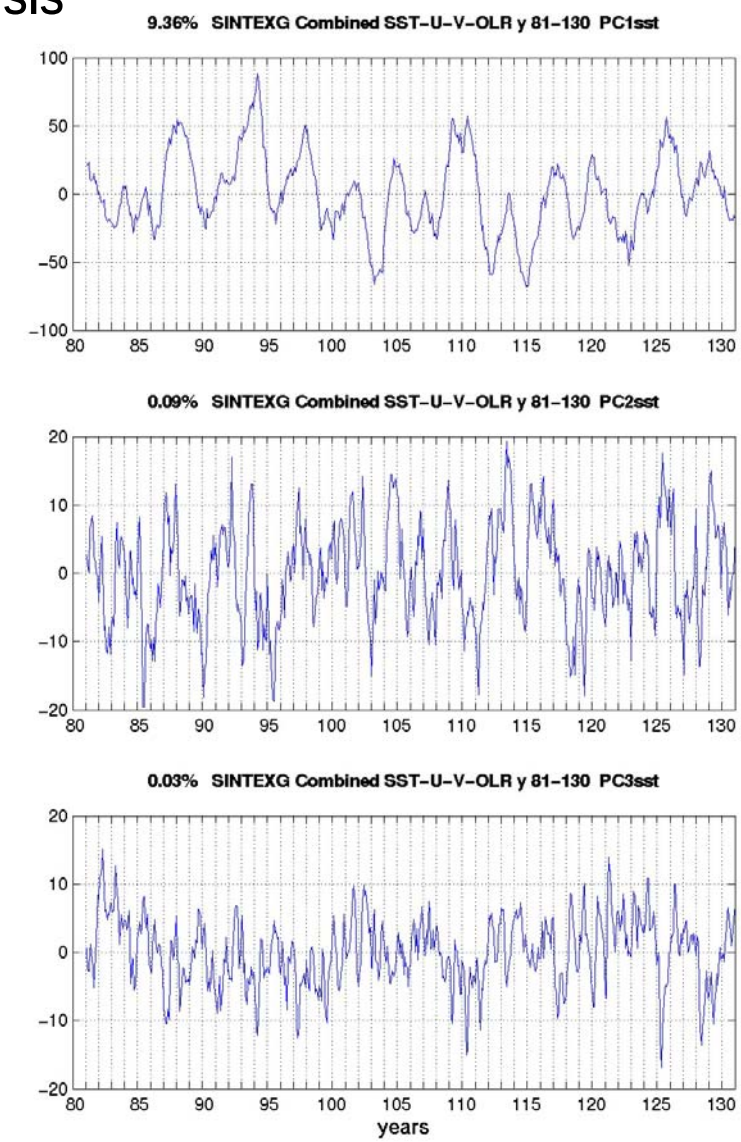
Istituto Nazionale di Geofisica e Vulcanologia

Spectral Analysis



EOFs analysis





MATLAB & Practical Applications on Climate Variability Studies day 1

Istituto Nazionale di Geofisica e Vulcanologia

Tutorial Program

1. Basics

- Working environment
- Dealing with matrices
- Useful functions
- Logical operators
- Saving and loading
- Data management
- Exercises

day 1

2. Programming

- *Basics graphics settings - ex*
- Functions & scripts
- Vectorization
- Exercises

day 2

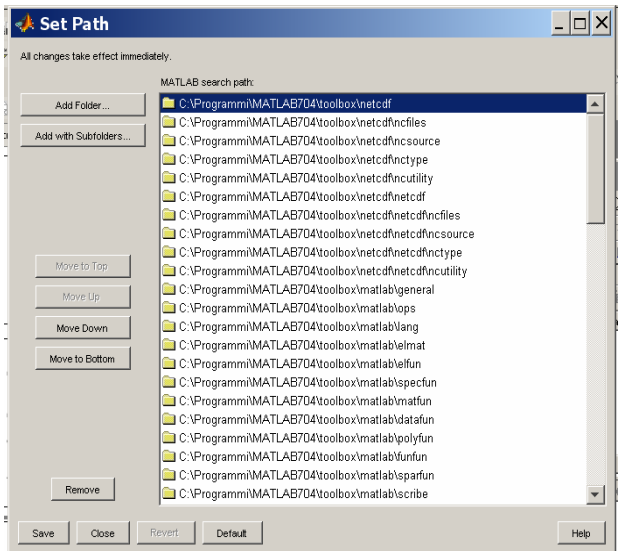
3. Applications

- Generic scripts
  - Means, STD
  - Anomalies
  - Correlations
  - Hovmoeller (ACW ..)
  - Time series (SSTA..):
    - detrending
    - filtering
- Interpolation: regular & irregular grids
- EOFs REOFs CEOFs
- Spectral analysis
- Wavelets
- Probability Density Functions

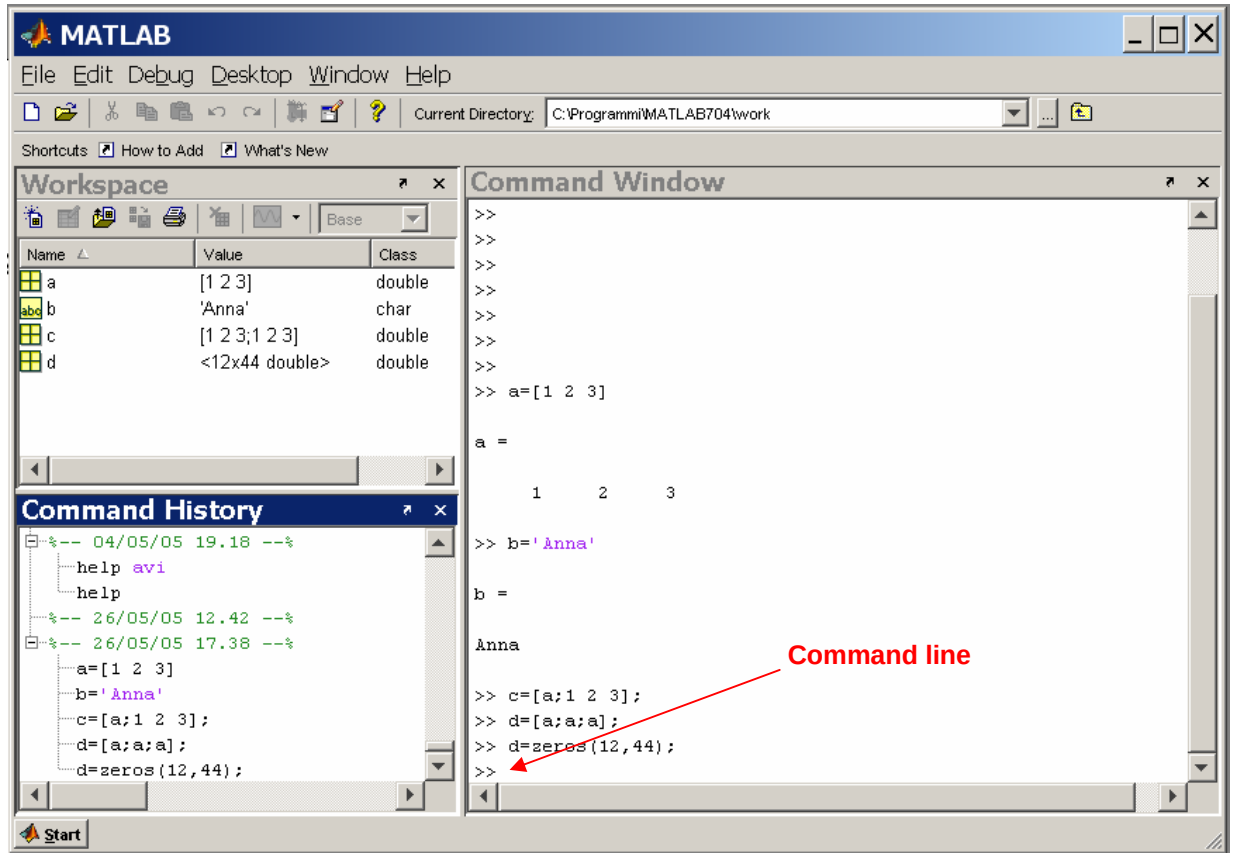
days 3, 4 & 5

Basics

- Working environment:



or:
>> addpath c:\Programmi\MATLAB704\toolbox\netcdf



You can use some external unix basics commands (**ls pwd**, ……) inside Matlab, in Windows environment too, but if you want to run extended unix command (in unix env.), you must to use the "**!**" symbol before.

It is important to set the matlab path as you prefer, before starting to work.

Basics

- Format :

```
>> x = [4/3 1.2345e-6]

>>format short
1.3333 0.0000

>>format short e
1.3333e+000 1.2345e-006

>>format short g
1.3333 1.2345e-006

>>format long
1.33333333333333 0.00000123450000

>>format long e
1.333333333333333e+000 1.234500000000000e-006

>>format long g
1.33333333333333 1.2345e-006

>>format bank
1.33 0.00

>>format rat
4/3 1/810045

>>format hex
3ff5555555555555 3eb4b6231abfd271

>>format compact
suppresses many of the blank lines that appear in the output.
```

Basics

- Dealing with matrices :

First, let's create a simple vector with 9 elements called a.

```
>>a = [1 2 3 4 6 4 3 4 5]
    a = 1 2 3 4 6 4 3 4 5
```

You can view individual entries in this vector. For example to view the first entry just type:

```
>> a(5)
    ans = 6
```

Now let's add 2 to each element of our vector, a, and store the result in a new vector.
Notice how MATLAB requires no special handling of vector or matrix math.

```
>>b = a + 2
    b = 3 4 5 6 8 6 5 6 7
```

Creating a matrix is as easy as making a vector, using semicolons (;) to separate the rows of a matrix.

```
>>A = [1 2 0; 2 5 -1; 4 10 -1]
    A =  1  2   0
         2  5  -1
         4 10  -1
```

We can easily find the transpose of the matrix A.

```
>>B = A'
    B =  1  2   4
         2  5  10
         0 -1  -1
```

Now let's multiply these two matrices together.
Note again that MATLAB doesn't require you to deal with matrices as a collection of numbers.
MATLAB knows when you are dealing with matrices and adjusts your calculations accordingly.

```
>>C = A * B
    C =  5 12  24
        12 30  59
        24 59 117
```

Basics

- Dealing with matrices :

Instead of doing a matrix multiply, we can multiply the corresponding elements of two matrices or vectors using the **.*** operator.

```
>>A = [1 2 0; 2 5 -1; 4 10 -1]
    A =   1  2   0
          2  5  -1
          4 10  -1

>>B = A'
    B =   1  2   4
          2  5  10
          0 -1  -1

>>C = A .* B
    C = 1   4   0
        4  25 -10
        0 -10   1
```

You can **delete rows and columns** from a matrix using just a pair of square Brackets:

```
>>A = [1 2 0; 2 5 -1; 4 10 -1]
    A =   1  2   0
          2  5  -1
          4 10  -1

>> A(:,2)=[ ]
   A =   1   0
         2  -1
         4  -1
```

The same for the **./** and **.** operators.

NaN *concept*:

NaN is the IEEE arithmetic representation for Not-a-Number.
A NaN is obtained as a result of mathematically undefined operations like 0.0/0.0 and inf-inf.
For examples we will use it in treating the land-sea masks of the numerical models.

Basics

- Useful functions for beginners:

HELP: On-line help, display text at command line.
LOOKFOR: Search all M-files for keyword.
WHOS: List current variables, long form.
MAX: Largest component.
MIN: Smallest component.
ROUND, **CEIL**, **FLOOR**, **FIX:** Rounding.
SQUEEZE: Remove singleton dimensions.
FIND: Find indices of nonzero elements.
MEAN: Average or mean value.
ISNAN: True for Not-a-Number.
FLIPUD: Flip matrix in up/down direction.
FLIPDIM: Flip matrix along specified dimension.
RESHAPE: Change size.
PERMUTE: Permute array dimensions.
REPMAT: Replicate and tile an array.
EVAL: Execute string with MATLAB expression.

```
Command Window

 To get started, select MATLAB Help or Demos from the Help menu.

>> help flipdim
 FLIPDIM Flip matrix along specified dimension.
    FLIPDIM(X,DIM) returns X with dimension DIM flipped.
    For example, FLIPDIM(X,1) where

        X = 1 4  produces  3 6
            2 5            2 5
            3 6            1 4


    Class support for input X:
       float: double, single

    See also fliplr, flipud, rot90, permute.


    Reference page in Help browser
       doc flipdim



>> |
```

```
Command Window
A =

     1     2     3
     4     5     6
     7     8     9

>> whos
  Name           Size                    Bytes  Class

  A              3x3                        72  double arr
  ans            1x3                        24  double arr
  f              0x0                         0  double arr
  filename       1x24                       48  char array

Grand total is 36 elements using 144 bytes

>> max(A)

ans =

     7     8     9

>> min(A)

ans =

     1     2     3

>>
```

Basics

- Useful functions for beginners:

ROUND, CEIL, FLOOR, FIX

```
Command Window
>> help fix

 FIX    Round towards zero.
    FIX(X) rounds the elements of X to the nearest integers
    towards zero.

    See also FLOOR, ROUND, CEIL.

>> help floor

 FLOOR  Round towards minus infinity.
    FLOOR(X) rounds the elements of X to the nearest integers
    towards minus infinity.

    See also ROUND, CEIL, FIX.

>> help round

 ROUND  Round towards nearest integer.
    ROUND(X) rounds the elements of X to the nearest integers.

    See also FLOOR, CEIL, FIX.

>> help ceil

 CEIL   Round towards plus infinity.
    CEIL(X) rounds the elements of X to the nearest integers
    towards infinity.

    See also FLOOR, ROUND, FIX.
```

```
Command Window
>> fix(-7.3)

ans =

    -7

>> floor(-7.3)

ans =

    -8

>> round(-7.3)

ans =

    -7

>> ceil(-7.3)

ans =

    -7

>>
```

Basics

- Logical operators:

A = [0 1 1 0 1];

B = [1 1 0 0 1];

Operator	Description	Example
&	Returns 1 for every element location that is true (nonzero) in both arrays, and 0 for all other elements.	A & B = 01001
|	Returns 1 for every element location that is true (nonzero) in either one or the other, or both arrays, and 0 for all other elements.	A | B = 11101
~	Complements each element of the input array, A.	~A = 10010
xor	Returns 1 for every element location that is true (nonzero) in only one array, and 0 for all other elements.	xor(A,B)=10100

Logical Operation	Equivalent Function
A & B	and(A, B)
A | B	or(A, B)
~A	not(A)

Istituto Nazionale di Geofisica e Vulcanologia

- Logical expressions using the Find function:

```
>> A = magic(4)
A =
   16    2    3   13
    5   11   10    8
    9    7    6   12
    4   14   15    1

>> i = find(A > 8)
>> A(i) = 100

A =
  100    2    3  100
    5  100  100    8
  100    7    6  100
    4  100  100    1
```

A(A > 8) = 100;

```
>> [row, col] = find(A > 12)
row =        col =
    1            1
    4            2
    4            3
    1            4
```

Basics

- Saving and loading:

First, let's create two simple vectors called a and b.

```
>>a = [1 2 3 4 6 4 3 4 5]
    a = 1 2 3 4 6 4 3 4 5
>>b = [3 4 5; 6 8 6;  5 6 7]
    b = 3 4 5
        6 8 6
        5 6 7
```

Now let's save these variables in a filename.mat file.

```
>> save filename.mat a b;     % you can save a and b on the same filename.mat
>>ls *.mat
   filename.mat
>>clear                       % to clear the working environment

>> load filename.mat
>>whos
            a      1x9            72  double array
            b      3x3            72  double array
Or:
>>clear
>> load filename.mat b
>>whos
            b      3x3            72  double array
```

Istituto Nazionale di Geofisica e Vulcanologia

- …Saving and loading:

The load function reads binary files containing matrices generated by earlier MATLAB sessions (see previous slide), or reads text files containing numeric data. The text file should be organized only as a rectangular table of numbers.

But if you need to read binary files not generated by MATLAB (exe: input.dat) it is necessary to use the fopen function:

```
>>filenamein='input.dat'                                  % filenamein is a char array

>>fid=fopen(filenamein,PERMISSION,MACHINEFORMAT)   % now the variable fid is the MATLAB ID of the file.
```

PERMISSION can be:

- **'r'** read
- **'w'** write (create if necessary)
- **'a'** append (create if necessary)
- **'r+'** read and write (do not create)
- **'w+'** truncate or create for read and write
- **'a+'** read and append (create if necessary)
- **'W'** write without automatic flushing
- **'A'** append without automatic flushing

MACHINEFORMAT can be:

- **'native'** or 'n' - local machine format - the default
- **'ieee-le'** or 'l' - IEEE floating point with little-endian byte ordering
- **'ieee-be'** or 'b' - IEEE floating point with big-endian byte ordering
- **'vaxd'** or 'd' - VAX D floating point and VAX ordering
- **'vaxg'** or 'g' - VAX G floating point and VAX ordering
- **'cray'** or 'c' - Cray floating point with big-endian byte ordering
- **'ieee-le.l64'** or 'a' - IEEE floating point with little-endian byte ordering and 64 bit long data type
- **'ieee-be.l64'** or 's' - IEEE floating point with big-endian byte ordering and 64 bit long data type.

Istituto Nazionale di Geofisica e Vulcanologia

- …Saving and loading:

…. Now, we have the fid variable that is the MATLAB ID of the input file and we are ready to get the input file in our working environment using the fread function

```
>> var = fread(fid,SIZE,PRECISION)
```

SIZE can be:

N	read N elements into a column vector.
inf	read to the end of the file.
[M,N]	read elements to fill an M-by-N matrix, in column order.

PRECISION can be:

'uchar'	unsigned character, 8 bits.
'schar'	signed character, 8 bits.
'int8'	integer, 8 bits.
'int16'	integer, 16 bits.
'int32'	integer, 32 bits.
'int64'	integer, 64 bits.
'uint8'	unsigned integer, 8 bits.
'uint16'	unsigned integer, 16 bits.
'uint32'	unsigned integer, 32 bits.
'uint64'	unsigned integer, 64 bits.
'single'	floating point, 32 bits.
'float32'	floating point, 32 bits.
'double'	floating point, 64 bits.
'float64'	floating point, 64 bits.

var contains the values getted from the input.dat file.

Istituto Nazionale di Geofisica e Vulcanologia

- …Saving and loading:

Netcdf format:

The ncload function read *netcdf* files as load function read *.mat* files.

```
>> ncload filename.nc
>> whos
        a      1x9          72  double array
        b      3x3          72  double array

>> clear
>> ncload filename.nc  a
>> whos
        a      1x9          72  double arrayv
```

To work with big netcdf files:

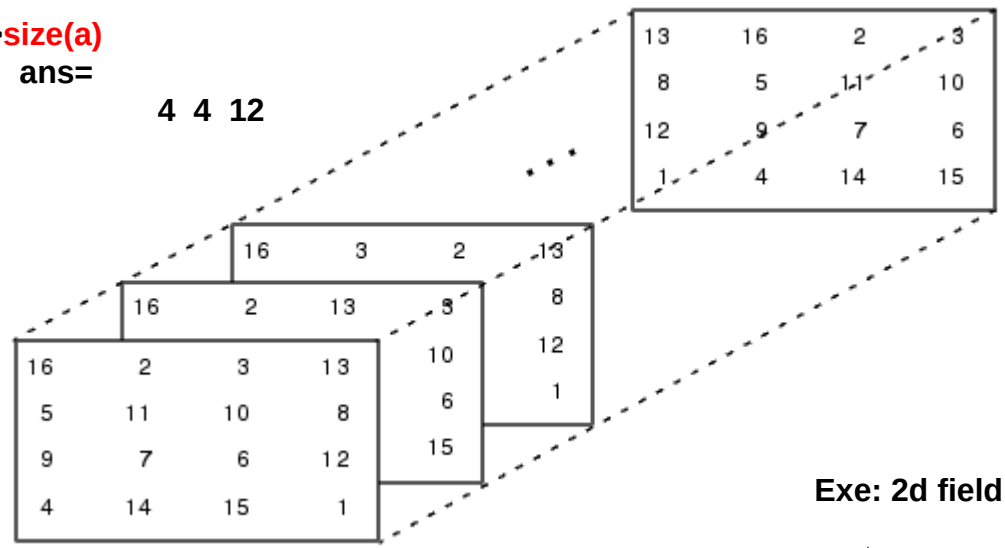
```
>>filenamein = 'filename.nc';          % filenamein is a char array
>>nomevar =  'sst' ;                   % nomevar is a char array
>>f = netcdf(filenamein,PERMISSION);   % now the variable f is the netcdf object.
>>var1 = f{nomevar};                   % var1 is the ncvar object.
>>var2 ( : , : ) = var1 (1 , : , :);
>> whos
        f            4d          2368      netcdf object
        filenamein   1x11        25        char array
        nomevar      1x3          5        char array
        var1         12x149x182   5365     ncvar object
        var2         149x182      45712    double array
```

Basics

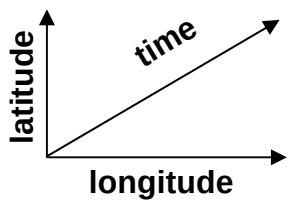
- Data Management:

Multidimensional arrays:

```
>>size(a)
ans=
        4  4  12
```



Exe: 2d field time series



- … Data Management:

Structures:

Structures are multidimensional MATLAB arrays with elements accessed by textual field designators. For example,

```
>> H.name = 'Guillermo';
>> H.intensity = 3;
>> H.mean_pressure_eye = 997;
```

creates a scalar structure with three fields:

```
H =
    name: 'Guillermo'
   intensity: 3
   mean_pressure_eye: 997
```

Like everything else in MATLAB, structures are arrays, so you can insert additional elements. In this case, each element of the array is a structure with several fields. The fields can be added one at a time,

```
>> H(2).name = 'Katrina';
>> H(2).intensity = 5;
>> H(2).mean_pressure_eye= 990;
```

or an entire element can be added with a single statement:

```
>> H(3) = struct('name','George',...
          'intensity',2,'mean_pressure_eye',985)
```

```
>> H =
1x3 struct array with fields:
   name
   intensity
   mean_pressure_eye
```

Istituto Nazionale di Geofisica e Vulcanologia

Basics

- … Data Management:

Cell arrays:

```
>>a = cell(8,1);
for n = 1:8
   a{n} = magic(n);
end
>>a
```

64	2	3	61	60	6	7	57
9	55	54	12	13	51	50	16
17	47	46	20	21	43	42	24
40	26	27	37	36	30	31	33
32	34	35	29	28	38	39	25
41	23	22	44	45	19	18	48
49	15	14	52	53	11	10	56
8	58	59	5	4	62	63	1

16	2	3	13
5	11	10	8
9	7	6	12
4	14	15	1

8	1	6
3	5	7
4	9	2

1	3
4	2

1

A cell array is an array whose entries can be data of any type!

```
a =
    [        1]
    [ 2x2  double]
    [ 3x3  double]
    [ 4x4  double]
    [ 5x5  double]
    [ 6x6  double]
    [ 7x7  double]
    [ 8x8  double]
```

To retrieve the contents of one of the cells, use subscripts in curly braces. For example, a{1} is 1.

Cell arrays contain copies of other arrays, not pointers to those arrays.
If you subsequently change A, nothing happens to C.

```
>> C = {A sum(A) prod(prod(A))}
C =
   [4x4 double]    [1x4 double]    [20922789888000]
```

Istituto Nazionale di Geofisica e Vulcanologia

- …Data Management:

… Cell arrays:

A cell array is an array whose entries can be data of any type!

```
>> a{1}='hurricanes pdi time serie';

>> a{2}=[7.3 6.0 6.4 7.1 7.2 6.3];

>> H.name='guillermo';

>> a{3}=H;

>> a

a =

   [1x25 char]    [1x6 double]    [1x1 struct]
```

```
>> a{1}
ans =
hurricanes pdi time serie

>> a{2}
ans =
   7.3000    6.0000    6.4000    7.1000    7.2000    6.3000
```

```
>> a{3}
ans =
   name: 'guillermo‘

>> a{3}.name
ans =
guillermo
```

MATLAB & Practical Applications on Climate Variability Studies day 1

Istituto Nazionale di
Geofisica e Vulcanologia

Exercises

- Exe 1:

Read an SST field in netcdf format, subsample and save in matlab format.

- Exe 2:

Read an SST field, create nino3 time series and save it in binary format.

- Exe 3:

Read an SST field, mask and compute global mean.

- Exe 4:

Load monthly SST fields in structured array, mean in time and concatenate annual averages.

B.Aires, 20-24/02/06
Centro de Investigaciones del Mar y la Atmosfera and
Department of Atmospheric and Oceanic Sciences (UBA)
E.Scoccimarro, A.F.Carril