Matlab for Oceanography
Meeting 2

Outline:

- cell mode
- structures and cell arrays
- NetCDF data format (Argo data example)
- basic graphics recap
- graphics handles
- plotting Argo data

Brief demonstration of **cell mode**

- allows you to run subsections of a Matlab script quickly
- allows you to easily create documentation for your script

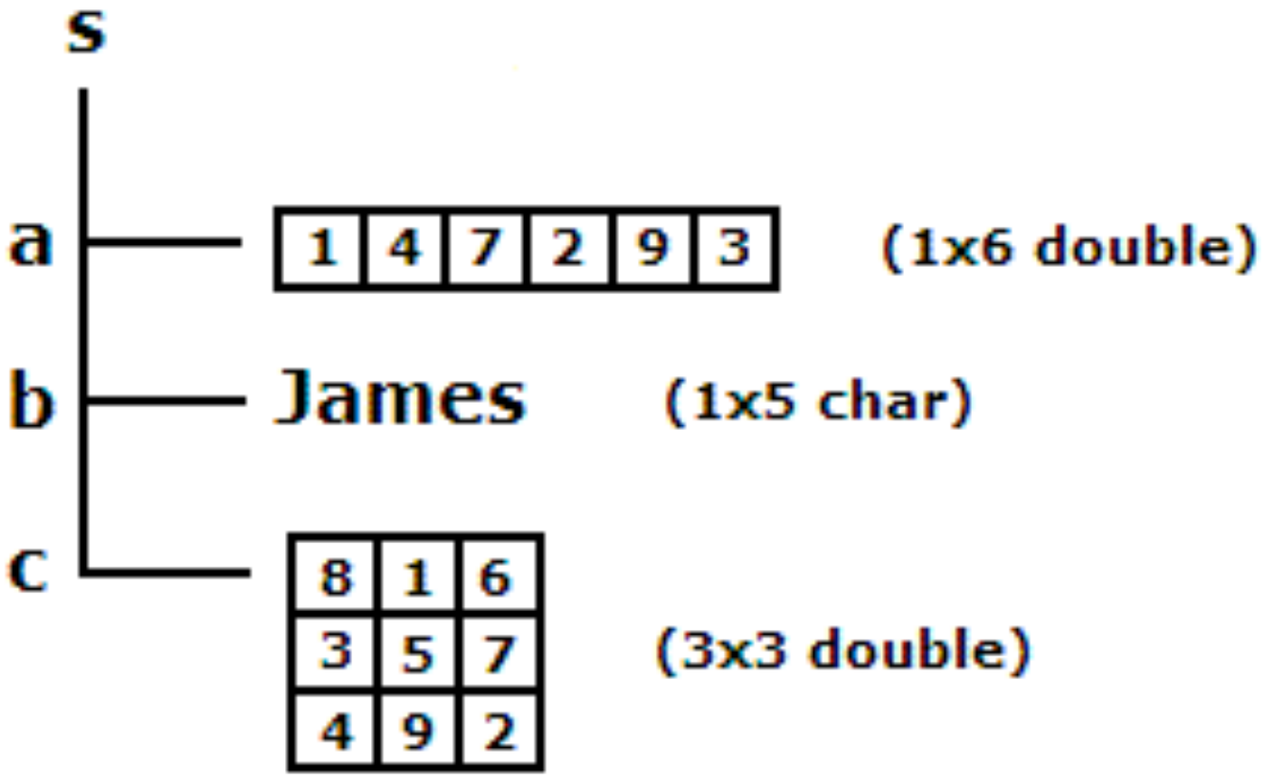
Let's look at example "predprey_driver_celldemo.m"

You may want to check out this brief video tutorial about cell mode: http://www.mathworks.com/support/2009b/matlab/7.9/demos/RapidCodeIterationUsingCells_viewlet_swf.html

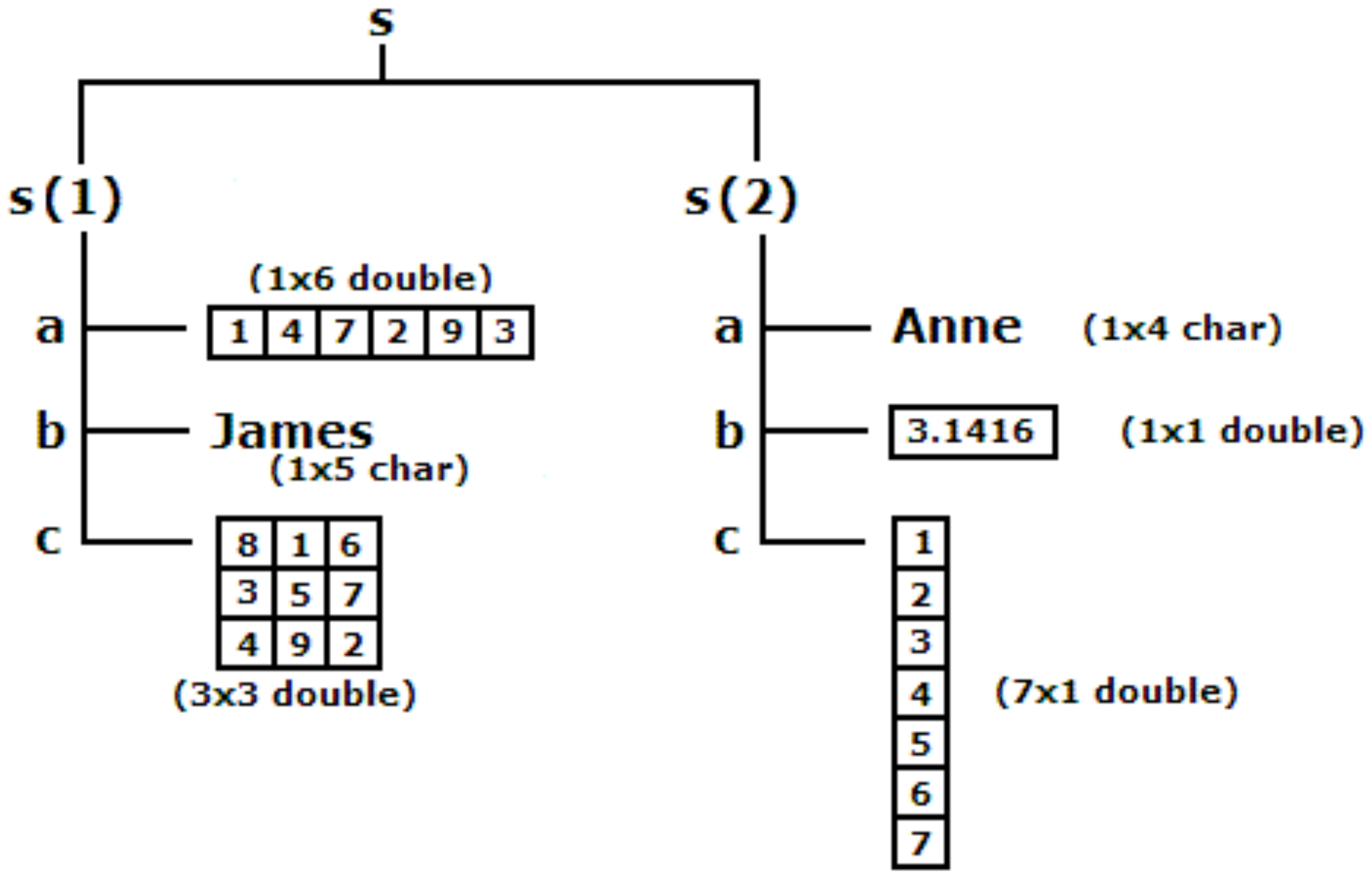
Limitations of **arrays**:
- can't mix characters and numbers
- can't handle strings of different lengths

Now, introduce **structures** and **cell arrays**, which are more flexible.

A **structure** allows you to put different types of data into its *fields*.



The fields are duplicated and referred to just as matrix elements.



http://www.mathworks.com/access/helpdesk/help/techdoc/matlab_prog/br04bw6-38.html

See example script “Creating_structure_example.m”

Reasons for using a structure:

- allows one to store arrays of mixed types and sizes in one entity

- functions can be applied directly to structures; they can also be passed to and from functions

- allows one to apply text labels

Cell arrays:

A ***cell*** can be thought of as the most general data object or 'data container' in Matlab.

A **cell array** is an array of these cells that can contain numbers, arrays, strings, structures and cell arrays itself.

See script "Creating_cell_array_example.m"

A cell is more general than a structure and the notations is different in some respects (which can be confusing).

Comparing Structures and Cell arrays:

Structures and cell arrays are similar in purpose in that both allow one to store heterogeneous data of different types and sizes in single entities.

The most obvious difference is that structures have *named fields*, while cell array uses *numerical indexing*.

Structures are useful where organization of data is important.
Cell arrays are useful when data are being processed by index in loops and for storing character strings of different lengths.

Note that varargin and varargout are cell arrays.

See “vararg_example.m”

For example, call as:

```
>> [a b c] = vararg_example(1,2,3,4,5)
a =
     2
b =
     4
c =
     6
```

NetCDF format

- is a self-contained binary data format where all metadata is included in the data file
- is rapidly becoming the standard for observational data sets and model output

The data in a NetCDF file is organized with the help of dimensions, variables and attributes.
The variables are essentially arrays that contain the data. Their sizes are defined with the help of dimensions.
One of these dimensions can be "unlimited", in other words, one can extend the dimension when adding more data to the file).

NetCDF format

Each variable comes with attributes that contain explanations, e.g. units.

A file can also contain global attributes explaining, e.g., who created the file and when, etc.

Let's look at an example from the Argo array
"4900093_trajectory_headder.txt"
and look at dimensions, variables and attributes.

Loading NetCDF data into Matlab -- Function overview

ncid = **netcdf.open**(filename, mode)

[ndims,nvars,ngatts,unlimdimid] = **netcdf.inq**(ncid)

[dimname, dimlen] = **netcdf.inqDim**(ncid,dimid)

[varname,xtype,dimids,natts] = **netcdf.inqVar**(ncid,varid)

attname = **netcdf.inqAttName**(ncid,varid,attnum)

attrvalue = **netcdf.getAtt**(ncid,varid,attname)

data = **netcdf.getVar**(ncid,varid)

netcdf.close(ncid)

There are many more functions, e.g. to create new variables and dimensions, etc.

See online help for those.

Let's look at "Loading_NetCDF_data.m"

Task for you:

Load data from two NetCDF files (4900093_profiles.nc, 5901467_profiles.nc), create a structure for these data and save it.

The variables we are interested in are:
- temperature,
- salinity,
- oxygen and, of course,
- location and
- time of the measurements.

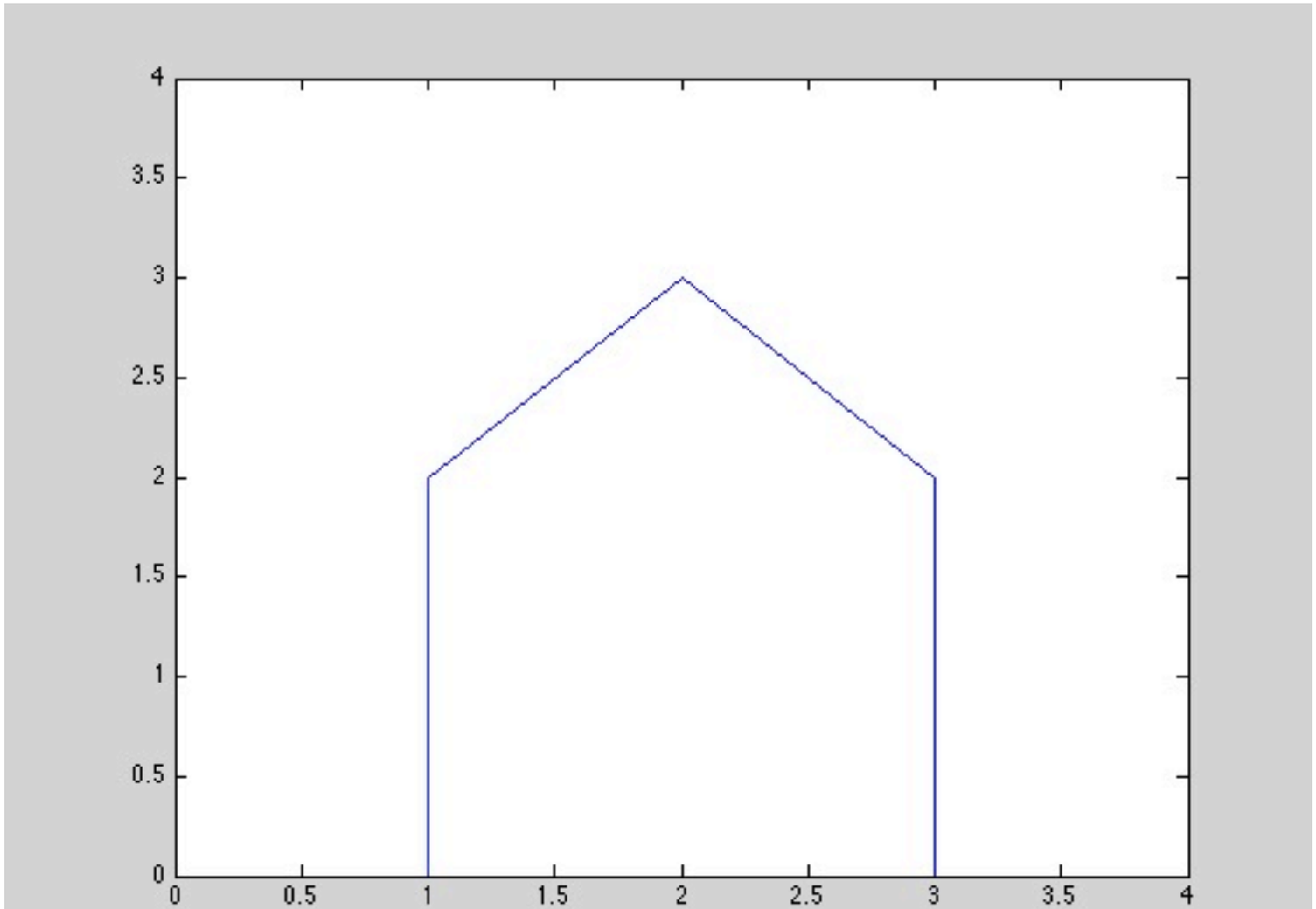
See my example in “Creating_Argo_data_structure.m”

For a refresher or crash course on basic graphics see:

"Graphics.m"

Exercises:

1) Recreate this plot:



2) Draw graphs of x^2, x^3, x^4 and $\exp(x^2)$ over the interval $0<=x<=4$ with semilogy.

3) Plot the function $u=y^2\exp(-x^2-0.3y^2)$ in the interval $-2<=x<=2$ and $-6<=y<=6$ using a grid resolution of at least 0.15 in both directions.

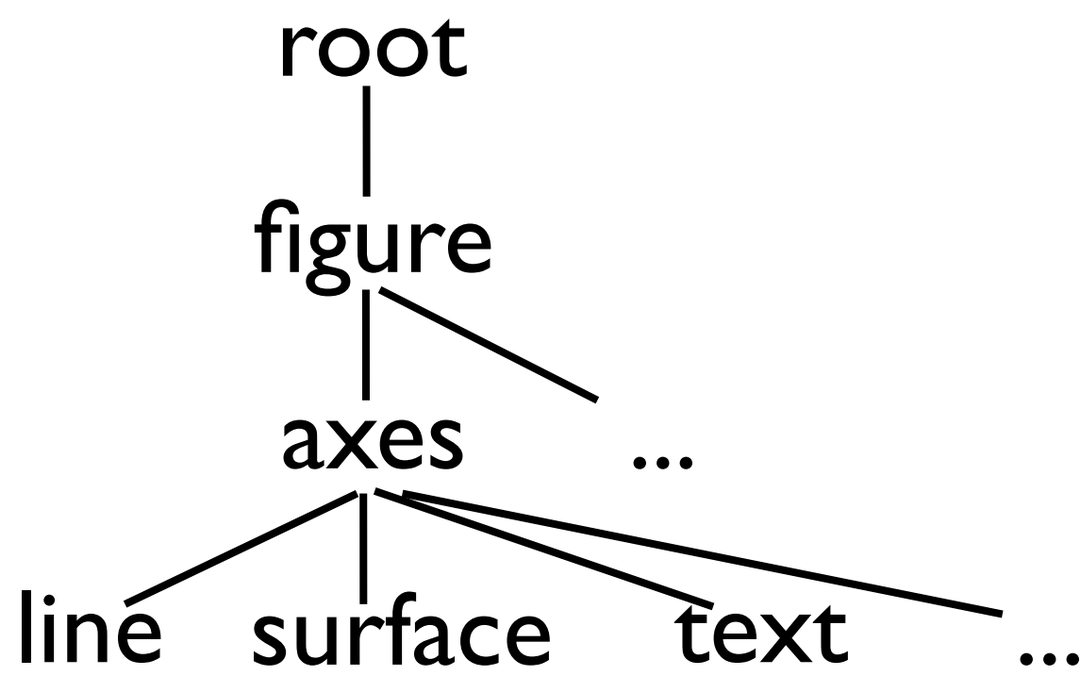
See “Graphics_exercises.m”

Graphics Handles

Matlab typically makes sensible choices for you when plotting graphics, such as defining the axis limits, tick locations, etc. However, sometimes you want more control. This is possible using the *Graphics Handles*.

A Graphics Handle can be thought of as a connection to a graphics object (handles are unique identifiers of graphics objects). The objects are arranged in a parent-child inheritance structure.

For example, line and text objects are children of axes objects. Axes define a region in the figure window and orient their children in this region.



http://www.mathworks.com/access/helpdesk/help/techdoc/creating_plots/f7-20419.html

Most functions that create graphics objects return handles to these (floating point numbers). Save this number in order to manipulate the object now or later.

Note, the root object has the handle 0. There is only one and it is created when Matlab starts up.

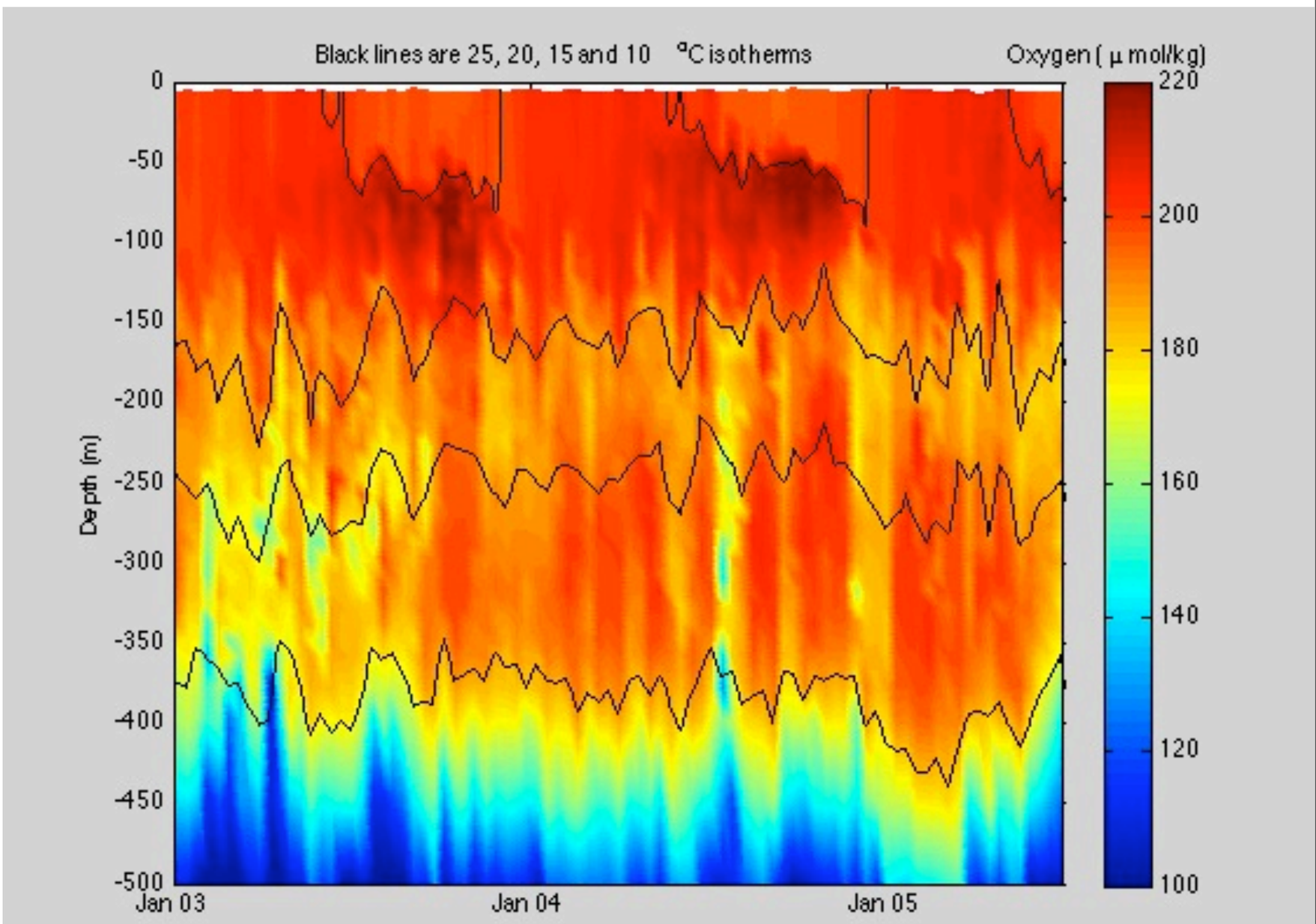
See “Graphics_handles.m”

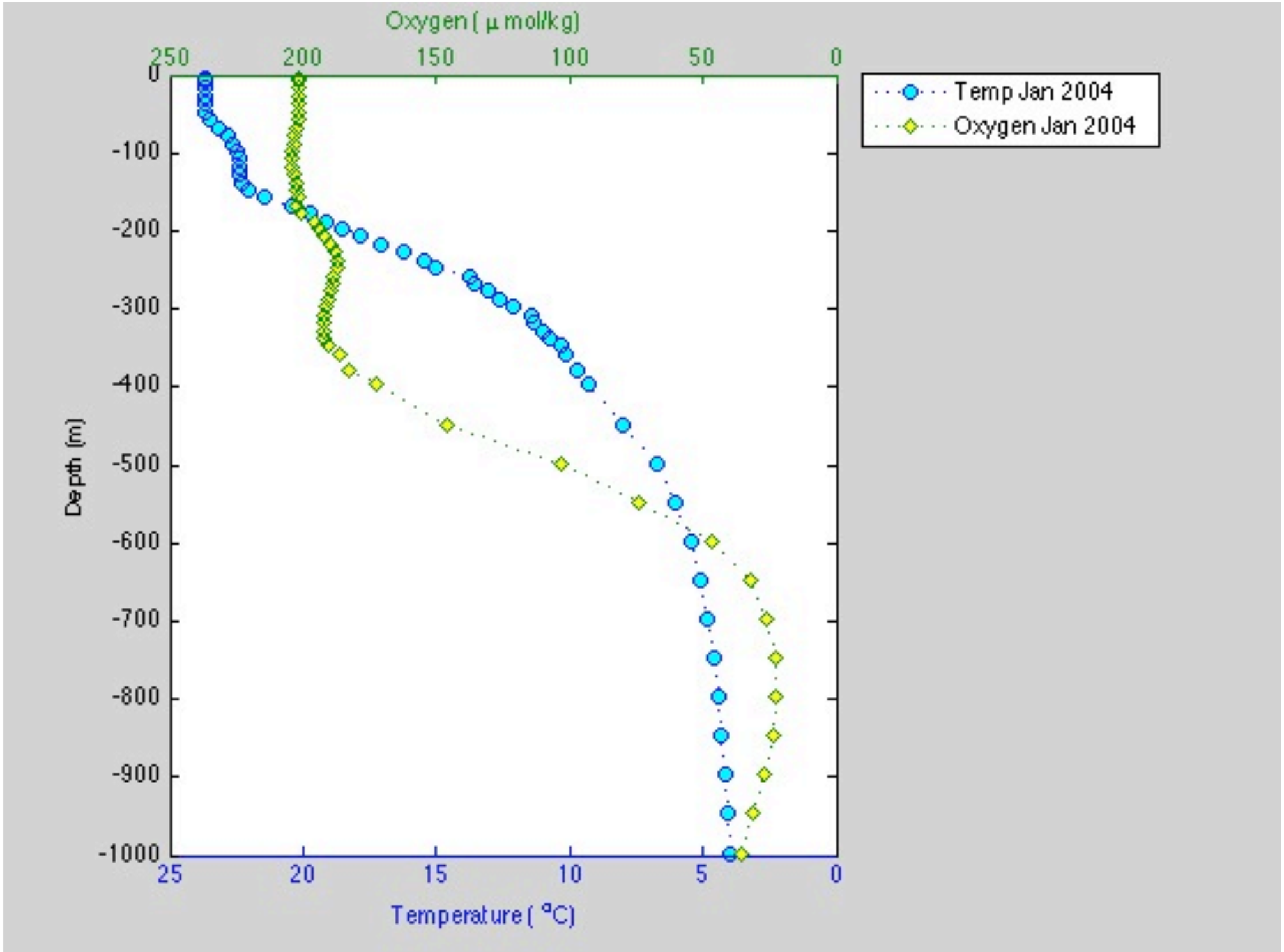
Exercise:

Create two plots using the Argo data structure,

1) 3d plot of oxygen with temperature contour lines (you may want to check out the function "datetick")

2) Temperature and oxygen profiles for early January of 2004 in the same plot




Oxygen (µ mol/kg)
250
200
150
100
50
0
0
-100
-200
-300
-400
-500
-600
-700
-800
-900
-1000
Depth (m)
25
20
15
10
5
0
Temperature (°C)
Temp Jan 2004
Oxygen Jan 2004

See “Plotting_argo_data.m”