Week 3 - Storing Information

Data Structures

Tues Class: Outline

- Simple data structures: arrays, strings, doubles/floats, characters
- Storing data in memory: variable declaration; colon operator
- Regular indexing into arrays (RC); more complicated indexing
- Distinguish between array indices and stored values
- Complex data structures: cell arrays and structures
- Distinguish between cells/fields and contents of cells/fields

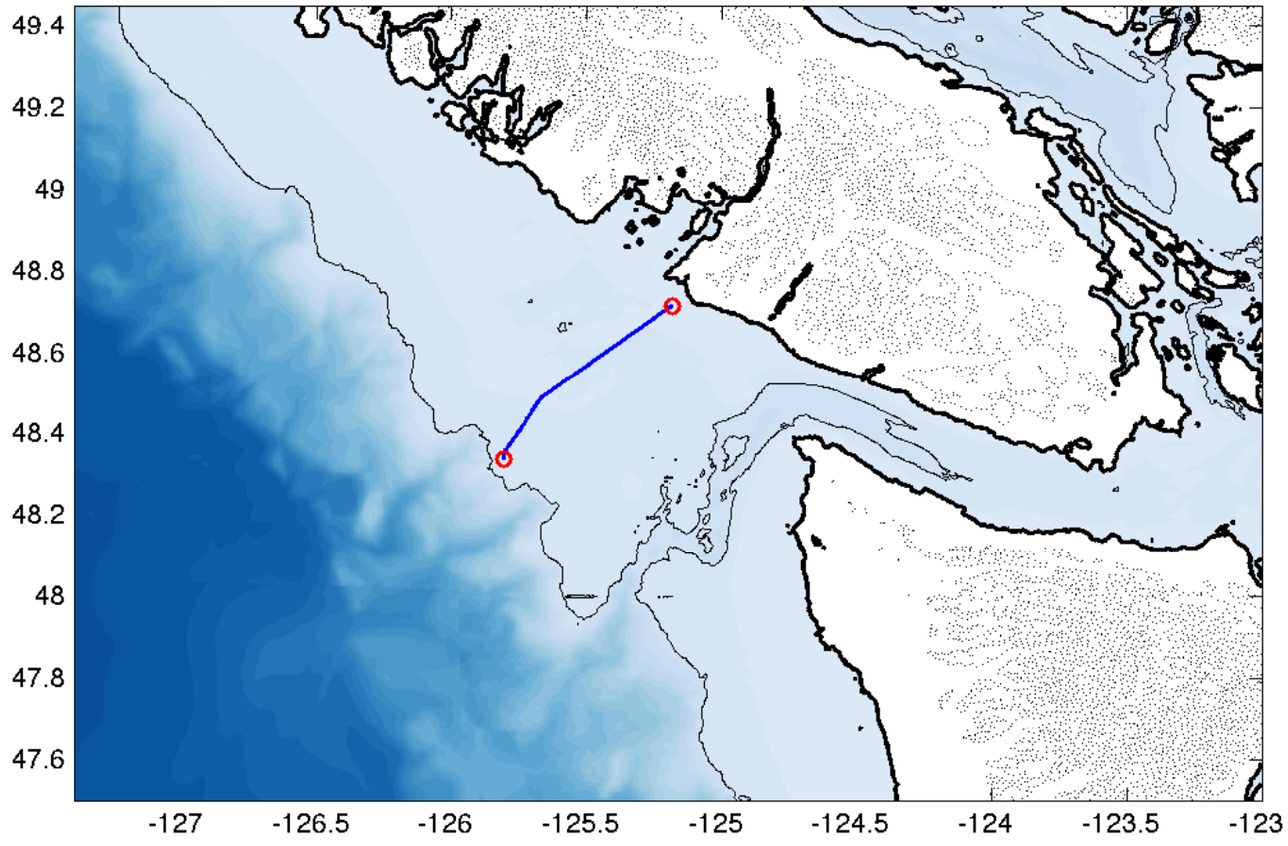
Quiz #1

In EOAS we often work with many measurements (ie. *lots of data*).

Example

- West Coast of VI 2013
- Transect: about 70 km
- Measurements:
    - every 1 m vertical
    - every 1 km lateral
- Avg Depth: 90 m
- Temperature, Salinity, Oxygen, Pressure, Latitude, Longitude, Depth

One of about 75 transects



90 x 70 = 6300 measurements per variable per transect.
7 variables and 75 transects, so 6300 x 7 x 75 =3,307,500 measurements!

How do you store and organize your data?

Primitive Data Structures: **Arrays**

Example, continued: The simplest way to store large amounts of data is in *arrays*. Consider the salinity measurements from the previous example - to plot this data, we require three arrays:

"2D Array", size MxN

2D Matrix

depth	saln										
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	…	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	…	NaN
…	…	…	…	…	…	…	…	…	…	…	…	…
15	32.65	32.26	32.13	32.31	32.52	32.51	32.58	32.77	32.76	32.42	…	32.53
16	32.67	32.30	32.13	32.36	32.55	32.59	32.69	32.85	32.81	32.49	…	32.58
17	32.66	32.35	32.15	32.40	32.64	32.61	32.64	32.82	32.87	32.61	…	32.57
18	32.67	32.43	32.23	32.46	32.70	32.67	32.66	32.83	32.93	32.68	…	32.59
19	32.74	32.51	32.30	32.56	32.77	32.75	32.72	32.85	32.99	32.72	…	32.72
20	32.78	32.56	32.36	32.69	32.85	32.82	32.77	32.88	33.02	32.78	…	32.74
21	32.86	32.61	32.44	32.74	32.89	32.88	32.87	32.97	33.07	32.91	…	32.76
…	…	…	…	…	…	…	…	…	…	…	…	…

2.2	3.1	4.0	4.9	5.8	6.7	7.6	8.5	9.4	10.3	…	12.1

distance

"1D Array", size Mx1

Column Vector

"1D Array", size Nx1

Row Vector

In Matlab:

```
>> whos
  Name            Size                Bytes  Class     Attributes

  depth         251x1                  2008  double
  distance        1x75                  600  double
  saln          251x75                150600  double
  tempr         251x75                150600  double

>>
```

Arrays are like tables that store values, with one value stored in each location, much like a spreadsheet in Excel.

They can be:

- 1 Dimensional (row or column)
- 2 Dimensional (square/rectangle)
- 3 Dimensional (cube)
- *N* Dimensional (hard to visualize!)

Creating Numeric Arrays

- Arrays are usually denoted using **square brackets [ ]**
- When creating an array, you usually assign it to a **variable name** using the '=' symbol (see previous week)
- Use a **semi-colon** (;) to separate rows, and a **comma** (,) or **white space** to separate columns

```
>> x = [5 8 6 9]

x =

     5     8     6     9
```

```
>> y = [5; 9; 1; 3]

y =

     5
     9
     1
     3
```

```
>> z = [1, 2; 3, 4]

z =

     1     2
     3     4
```

```
>> a = [4 9 7; 3 9]
```

- Use the **colon** (:) operator to create numeric series. This is very useful!

```
>> x = 1:5

x =

     1     2     3     4     5
```

```
>> y = [-5:0.2:-4.4]

y =

   -5.0000   -4.8000   -4.6000   -4.4000
```

```
>> z = 3:-2:-6

z =

     3     1    -1    -3    -5
```

Creating Character Arrays

- Character arrays are similar to numeric arrays, but contain characters
- Characters are denoted using single quotes: ‘a’ ‘E’ ‘v’
    - “Space” is a character and has to be explicitly inserted
    - Row vector of characters is known as a “string” of characters

```
>> x = 'hello'

x =

hello
```

```
>> y = ['h' 'ell' 'o']

y =

hello
```

```
>> z = ['hello' ' jello']

z =

hello jello
```

*notice the explicit space before ‘j’

```
>> xyz = ['hello'; 'jello']

xyz =

hello
jello
```

- *x* and *y* are identical
- *x*, *y*, and *z* are all 1D character arrays, or simply “strings”
- *xyz* is a 2D character array

Indexing into Arrays

- For accessing a subset of an array
- Think **RC** — row, column
- Regular indexing always uses the convention **(row, column)**
    - ie. “row-comma-column”

Forgot how to index?
Think “RC Cola”

Example: Given that the array `sal` looks like this:

32.65	32.26	32.13	32.31	32.52	32.51	32.58	32.77	32.76	32.42	32.53
32.67	32.30	32.13	32.36	32.55	32.59	32.69	32.85	32.81	32.49	32.58
32.66	32.35	32.15	32.40	32.64	32.61	32.64	32.82	32.87	32.61	32.57
32.67	32.43	32.23	32.46	32.70	32.67	32.66	32.83	32.93	32.68	32.59
32.74	32.51	32.30	32.56	32.77	32.75	32.72	32.85	32.99	32.72	32.72
32.78	32.56	32.36	32.69	32.85	32.82	32.77	32.88	33.02	32.78	32.74
32.86	32.61	32.44	32.74	32.89	32.88	32.87	32.97	33.07	32.91	32.76

`sal(2:3, 2:5)` `sal(end, :)` `sal(:,7)`

Here the colon (:) operator means “all”, and the reserved word “end” refers to the last element

Worksheet

Please find your groups

Start on exercises 1 and 2

Complex Data Structures

Arrays are limiting because they only store one type of data at a time and because they must have regular shapes (ie. rectangles). There are two types of data structures that can accommodate more than one type of data type:

Cell Arrays

Arrays of “cells” where each cell may contain any simple data structure (ie. a numeric or character array). Use curly braces {} to denote cell arrays.

```
>> Var1 = {'Farmer John'; [5 6; 7 8]; [1:4]}

Var1 =

    'Farmer John'
    [2x2 double]
    [1x4 double]

>> Var1{2}

ans =

     5     6
     7     8
```

Structures

Structures are variables that contain a series of fields, each holding a simple data structure. Use “dot notation” to access

```
>> Client.name = 'Farmer John';
>> Client.age = 39;
>> Client.assets = [326, 254, 784];
>> Client

Client =

      name: 'Farmer John'
       age: 39
    assets: [326 254 784]

>> Client.name

ans =

Farmer John
```

Practice with structures and cell arrays

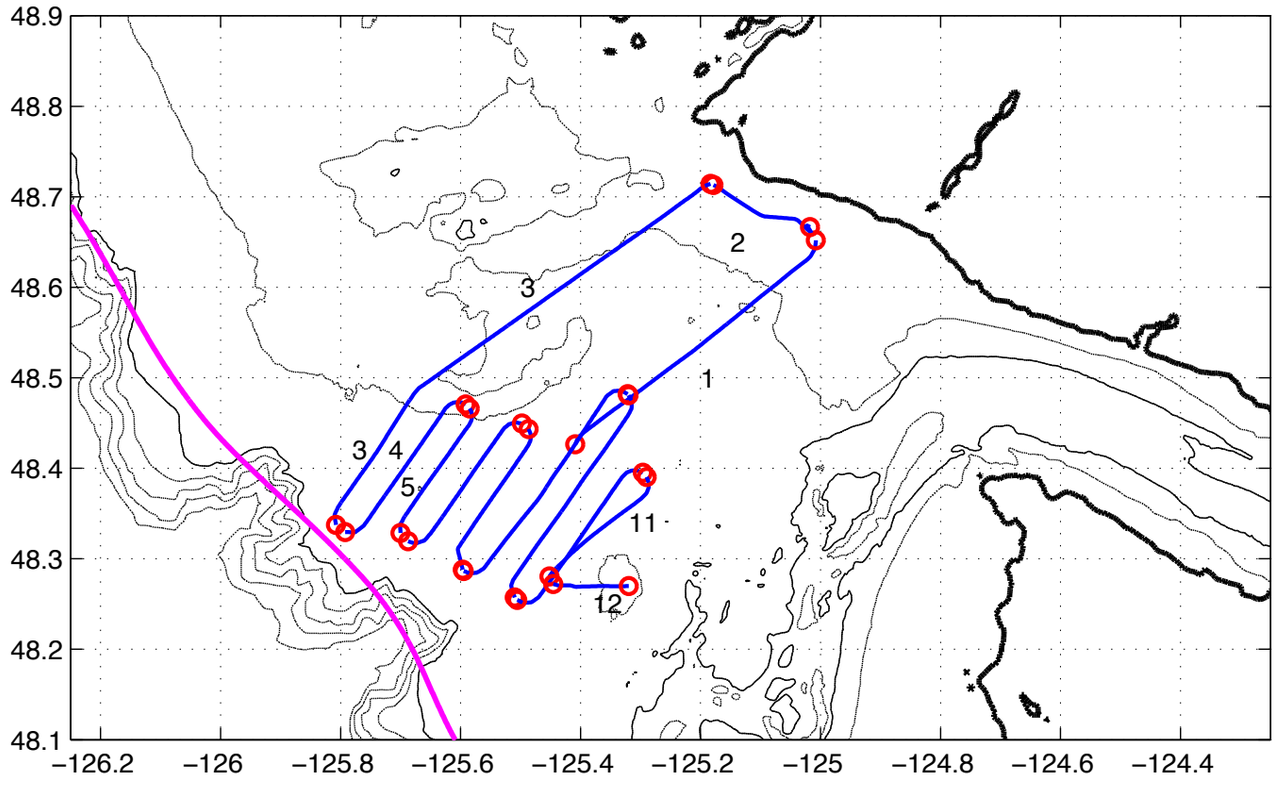
Worksheet exercise 3 and this week's lab

Cell Arrays: A practical example

Go back to our example of last year's Vancouver Island cruise. Say the cruise consisted of 12 individual legs (leg 3 is the one we looked at earlier), each measuring temperature (*T*), salinity (*S*), and distance (*x*), depth (*z*)

Problem: What is an efficient way to store *T*, *S*, *x,* and *z* for each leg?

Solution: You do NOT need 12x4=48 variables! Instead, create four cell arrays, each with 12 cells.



Use four variables, each a cell array with 12 cells (one for each leg)

```
>> whos
  Name          Size                 Bytes  Class

  depth         1x12                 25440  cell
  distance      1x12                  8544  cell
  saln          1x12               1808544  cell
  tempr         1x12               1808544  cell
```

*notice the "class" on the right

next slide: more practice to try on your own

More complicated indexing

Example: Given that the array `sal` looks like this:

32.65	32.26	32.13	32.31	32.52	32.51	32.58	32.77	32.76	32.42	32.53
32.67	32.30	32.13	32.36	32.55	32.59	32.69	32.85	32.81	32.49	32.58
32.66	32.35	32.15	32.40	32.64	32.61	32.64	32.82	32.87	32.61	32.57
32.67	32.43	32.23	32.46	32.70	32.67	32.66	32.83	32.93	32.68	32.59
32.74	32.51	32.30	32.56	32.77	32.75	32.72	32.85	32.99	32.72	32.72
32.78	32.56	32.36	32.69	32.85	32.82	32.77	32.88	33.02	32.78	32.74
32.86	32.61	32.44	32.74	32.89	32.88	32.87	32.97	33.07	32.91	32.76

1. How might you create the following array, *using only one line of code*?

32.30	32.13	32.36	32.55	32.59
32.43	32.23	32.46	32.70	32.67

2. Given `x=1:2:10` is stored in memory, what do you think the following will produce?

```
z = sal(1,x)
```