GEOS F436/636 Beyond the Mouse

Christine (Chris) Waigl
University of Alaska Fairbanks – Fall 2018

Week 7: MATLAB plots, figures and maps 1 & 2

Topics for week 7

- Reminders from last week
- Understanding the basic components of a plot/figure
- 2D and 3D plots; plotting data vs. plotting functions
- Multiple plots in one figure
- Visualizing rasters (matrices) and maps ← requires Mapping Toolbox
- Finishing up MATLAB topics:
    - Strings vs. character arrays
    - basics of dealing with date and time

Reminders from last week: accessing data

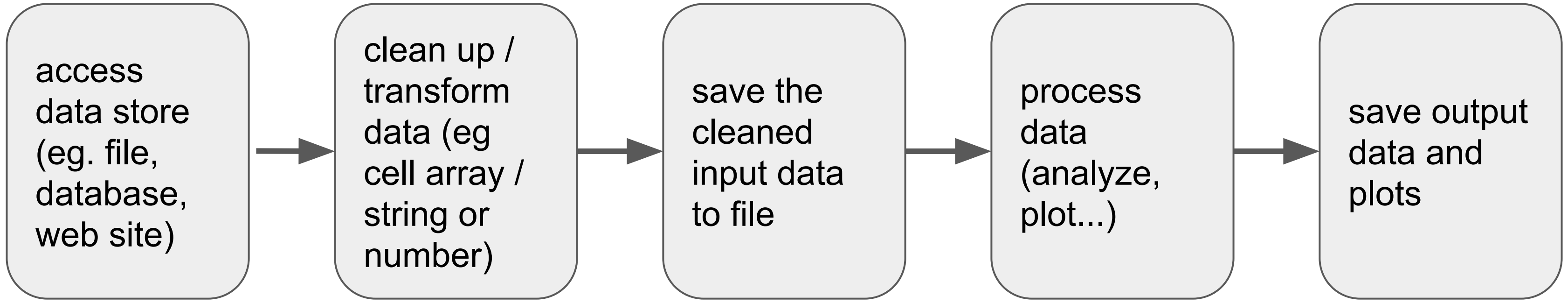
- MATLAB offers a number of built-in functions to access data stored in files:
    - `fgetl` to read text files line by line, `dlmread` or `csvread` (with corresponding `dlmwrite/csvwrite`) to read numerical tabular data into a matrix, textscan to read tabular data that may contain strings into a cell array. (Use `fprintf` for writing data line-by-line.)
    - For binary file formats functions exists as well: `xlsread/xlswrite`, `h5read/hdf5write...`
- Reading data from a URL with `webread` is similar to reading from a file
- Make sure to read the documentation on the functions / commands you are using and understand the examples. What are the arguments of the function (ie, what variables do you have to pass into the function to make it work?) It takes time and practice!

If you want to have data to play around with, you can load the sea ice climatology data for this week:

```
[doy, avg_extent, std, perc10, perc25, perc50, perc75, perc90] = loadseaicedata();
```

Project organization and planning are key!

- The basic workflow is usually:


access data store (eg. file, database, web site)
clean up / transform data (eg cell array / string or number)
save the cleaned input data to file
process data (analyze, plot...)
save output data and plots

- Think of these steps *before* you start fiddling with the data and create a directory layout. Don't forget that storing your code in GitHub will help managing changes. (But add large data files to your .gitignore file.

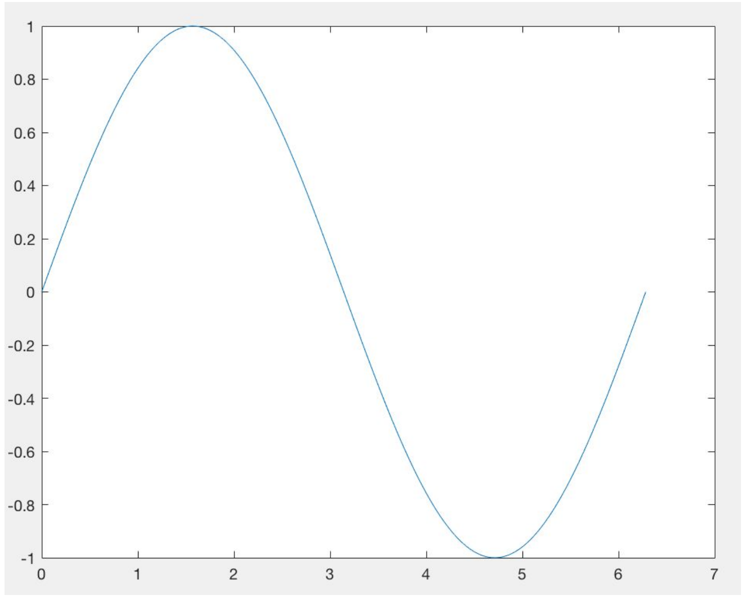
Making plots and figures is an integral part of data analysis and processing

The function of plots is **visual communication.**
(We will come back to this point.)

In MATLAB, 2D-plotting is a simple process:

1. generate x-data
2. generate y-data
3. plot
4. (... add visual improvements ...)

```
>> x = 0:pi/100:2*pi;
>> y = sin(x);
>> plot(x, y);
```



There are functions other than plot to achieve various types of plots. Try some of these:

```
>> x = 0:pi/10:2*pi;                      ← vary the number of x-values!
>> y = sin(x)+0.1*rand(1,length(x));  ← what does the second term do?
>> plot(x, y);
>> stem(x, y);
>> bar(x, y);
```

Also, try some other plots:

```
>> x = 0:0.2:10;
>> y1 = x.*x;
>> hist(x, y1);
>> semilogy(x, y1);
>> semilogx(x, y1);
>> loglog(x, y1);
```

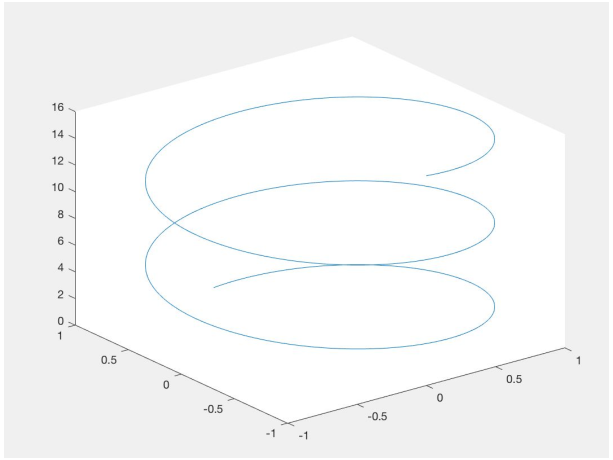
What is the difference between plotting data and plotting a function?

ANS = There really is none: You plot a function by generating x-values, applying the function to get y-values, and then plotting the data

Plotting 3D-data comes in two cases: line plots in 3D-space, and plots over a 2D-surface

3D line plots are parameterized curves:

```
>> t = 0:pi/100:5*pi;
>> x = sin(t); >> y = cos(t);
>> plot3(x, y, t);
```



Surface plots have a 2D-domain and a 1-D value space. We generate the input domain using the meshgrid function.

```
>> x = linspace(1,10); y = x;
>> [X, Y] = meshgrid(x, y);
>> Z = sin(X) + cos(Y);
>> surf(X, Y, Z);
>> contour(X, Y, Z);
>> mesh(X, Y, Z);
```

Changing the appearance of a plot

Line style, marker style and color are changed by passing a third "LineSpec" argument to the plot function. Look at **doc plot** for all options!

```
>> x = 0:pi/10:2*pi; y = sin(x) + 0.1*rand(1, length(x));
>> plot(x, y, 'rx');
>> plot(x, y, 'bo');
>> plot(x, y, 'g-x');
```

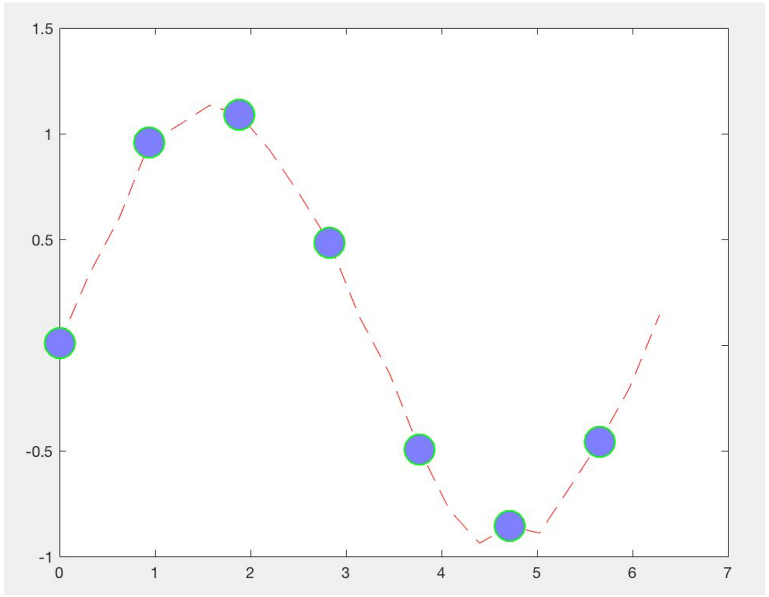
You can plot several lines into the same plot ("axes object") by saying "hold on":

```
>> hold on
>> plot(x, y, 'rx');
```

Don't forget to switch hold off again.

More appearance attributes are changed by adding key-value attribute pairs

```
plot(x,y,'ro--',...
'MarkerIndices',1:3:length(y), ...
'MarkerFaceColor', [0.5, 0.5, 1], ...
'MarkerEdgeColor', 'g', ...
'MarkerSize', 20);
```



Also add:

```
title('...'), xlabel('...'),
ylabel('...'), grid
text(xpos, ypos, 'some_string');
text(xpos, ypos, sprintf('Station: %s', stations{num}));
```

For more advanced changes, you'll need the **gca** object. gca stands for "get current axes"

For example: change the x- and y-limits (show only part of the data)

```
>> set(gca, 'XLim', [xmin xmax]); ← x-axis only
>> set(gca, 'YLim', [ymin ymax]); ← y-axis only
>> set(gca, 'XLim', [xmin xmax], 'YLim', [ymin ymax]); ← both axes

>> get(gca, 'XTick')
>> set(gca, 'XTick', 1:3:15)
>> set(gca, 'XTickLabel', {50, 'Fred',  'Tuesday', 75.5, 999})
```

Plotting multiple "things": The MATLAB graphics object hierarchy:

- Screen
  - Figure1
    - Axes1 (xlabel, ylabel, title, tick marks, tick labels)
      - Graph1 (linestyle, legendlabel)
      - Graph2
      - …
    - Axes2
      - Graph1
      - …
  - Figure2
    - Axes1
      - Graph1
      - Graph2
    - Axes2
      - Graph1

The **figure** object

To create a new figure with no axes: `>> figure;`

To highlight a figure that is already displayed (if it doesn't already exist, it will be created): `>> figure(2)`

To get a particular property associated with a figure:

```
>> get(figure(1), 'Position')
[420 528 560 420]
```

To modify a particular property associated with a figure:

```
>> set(figure(1), 'Position', [100 100 560 420])
```

This particular example will just move where figure(1) is plotted on the screen.

gcf is a 'handle' for the current active figure window:

>> figureposition = get(gcf, 'Position')

The **axes** object

New figures are created without a set of axes. To get a ‘handle’ for the current active set of axes use gca (get current axes). Example: get a list of all properties associated with current axes: `>> get(gca)`   `>> get(gca, ‘position’)`
This will return the screen position of the current active figure window, which by default is: [0.13 0.11 0.775 0.815] (The Format here is [xorigin yorigin xwidth yheight] in fractions of the figure window width.)
To modify the position of the current axes within a figure:

```
>> set(gca, 'position', [0.2 0.3 0.6 0.4])
```

The axes would start 20% of the way across the screen, 30% of the way up, and be 60% the screen width, and 40% the screen height.

Subplots (= several plots in one figure)

Subplots means: One figure contains multiple sets of axes.

```
figure(2)
ax1 = subplot(2,1,1);
x = linspace(0,10);
y1 = sin(x);
plot(x,y1)
title('Subplot 1: sin(x)')
```

```
subplot(2,1,2)
y2 = sin(2*x);
plot(x,y2)
title('Subplot 2: sin(2x)')
```

`subplot(n, m, k)` ← subplot number k in an n by m grid. Count goes rows first.

More on axes

You can use the axes command: `>> axes('position', [0.2 0.3 0.6 0.4]);`
MATLAB default plot properties (appearance) is fine for the screen, but the features are *too small* for presentations and publications. You will have to adjust these features. For example:

`set(`**`gca`**`,'FontName','Corbel','FontSize',12,'FontWeight','Bold',  'LineWidth',2);` ← if I haven't given a name to the current axes, I use "gca"

`set(`**`ax1`**`,'FontName','Calibri','FontSize',12,'FontWeight','Bold',  'LineWidth',4);` ← if I know the name of the current axes ("ax1")

The legend command adds descriptive labels to each plotted data series (in order of plotting), either to the current axes or specific axes:

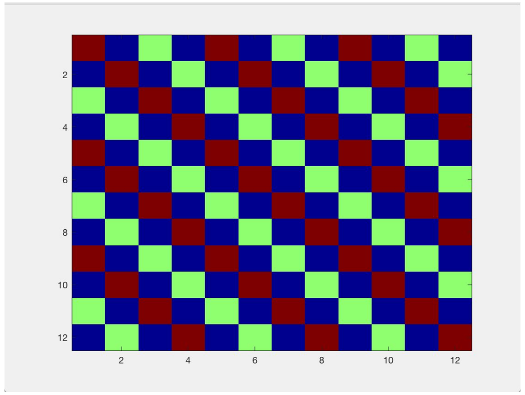
`legend('label1', 'label2', ...)   or legend(ax1, {'label1', 'label2', ...}`

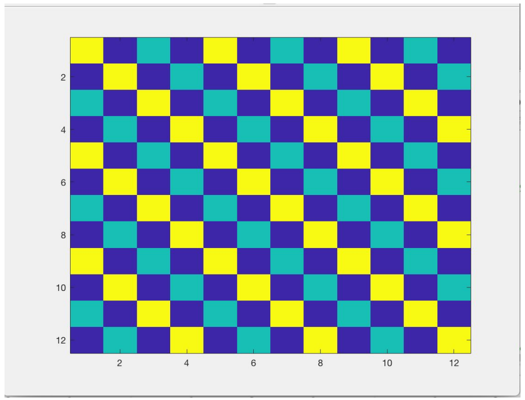
Visualizing raster data (images)

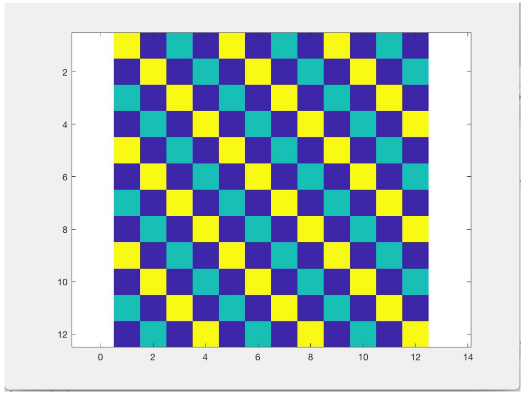
This is similar to 3D plotting: The domain of the data to be plotted is an x/y plane, and the colors represent z-values. The data can be imagined as a matrix, or, for RGB-images, three stacked matrices (n by m by 3). Example, in steps (note: `eye` is the identity matrix with 1 on the diagonal and 0 otherwise. `repmat` repeats a matrix):

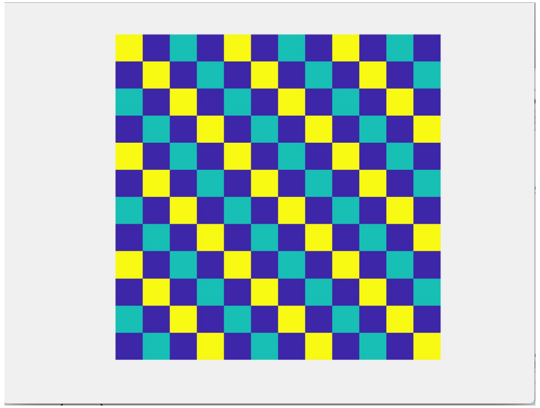
```
>> A = repmat(eye(4), 3) + repmat(eye(2), 6);
>> imagesc(A)           ← display an image, with a scaled colormap
>> colormap default       equivalent to: image(A, 'CDataMapping','scaled')
>> axis equal
>> axis tight
>> axis off
```

... and these look like:







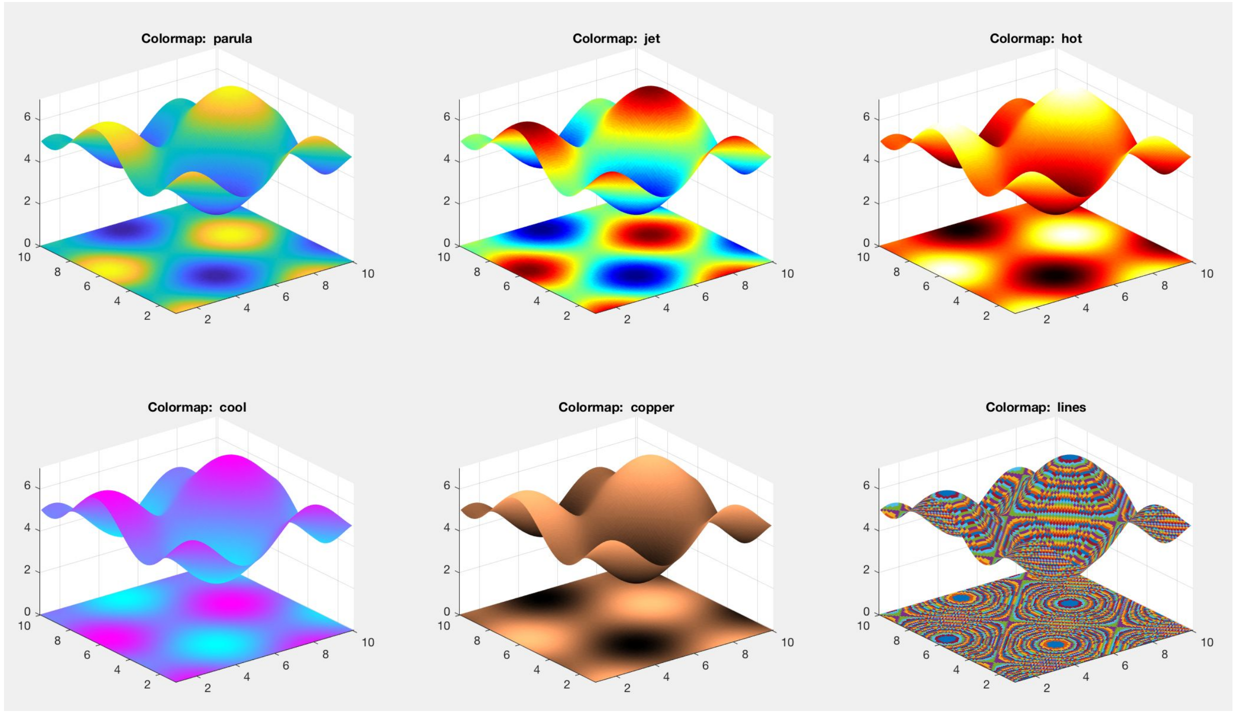


The order of commands is important!

If you use:
`axis off`

You don't need :
`axis tight`

Carefully select a color map: Ideally, the brightness value should rise or fall from one end to the other. The old default "jet" is visually more confusing than the new default "parula". "hot" can be used for thermal images. Categorical maps like "lines" will make your data non-continuous: use only when appropriate.



The code to generate the previous plot illustrate creating sub-plots in a loop.

```
colormapnames = {'parula', 'jet', 'hot', 'cool', 'copper', 'lines' };
x = linspace(1,10); y = x;
[X, Y] = meshgrid(x, y);
Z = sin(X) + cos(Y) + 5;
figure(7)
for ii=1:length(colormapnames)
    subplot(2, 3, ii)
    plot1 = surf(X, Y, Z);
    set(plot1,'linestyle','none')
    hold on
    imagesc(x, y, Z)
    colormap(gca, colormapnames{ii})
    axis tight
    title(['Colormap:  ' colormapnames{ii}])
end
```

Saving plots to a file

Two options: `print()` and `saveas()`.

`>> saveas(figure(2), 'fig2.png')` ← doc saveas for formats

`>> print -f1 -dpng myplotfilename.png` ← script form

`>> print('-f1', '-dpng', '-r200', 'f1.png')` ← functional form

For example in a loop:

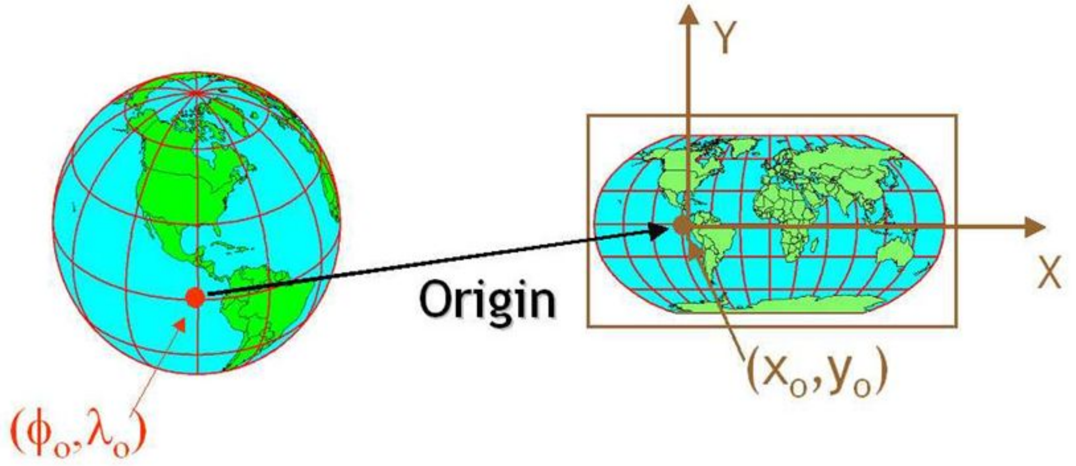
```
for ii = 1:Nplots
    print('-depsc2', sprintf('-f%d',ii), '-r70', sprintf('f%d.eps',ii));
end
```

Making maps in MATLAB

Maps are a special case of raster plot. They are a form of 3D plot: A color value is plotted over an X/Y grid. The task is made more difficult by the need to deal with (flat) coordinate systems to represent a section of the (ellipsoid) earth.

You need the Mapping Toolbox installed. Get started by typing

```
>> mapdemos
```



What makes maps much harder than other raster plots is the need to project the curved earth surface on a flat plane.

- A *projection* maps a point on the curved surface onto the plane. A *datum* is the model of the earth surface used (sphere, ellipsoid, geoid...)
- Use `projlist` to find out which projections your version of MATLAB offers.
- Use the geographically informed functions axesm and surfm (etc.) to generate geographically informed figures.
- However, you'll need lat/lon coordinates of every pixel in your data. If your data comes in a GeoTIFF file, for example, use geotiffread to access the data and geotiffinfo to extract information about the file's projection and other data properties.

Strings vs. character arrays

Strings were introduced to MATLAB in 2016. They improve handling of text data

```
>> mychararray = 'A horse! A horse! My kingdom for a horse!';
>> mystr = "A horse! A horse! My kingdom for a horse!";
```

Character arrays are simply numerical arrays of the character codes of the letters.
Character arrays sometimes have unexpected behavior:

```
>> mychararray + 1 ←  [ 66  33  105  112  115  116  102   34  ...]
>> mystr + 1       ←  "A horse! A horse! My kingdom for a horse!1"
>> length(mychararray) ← 41
>> length(mystr)       ← 1
```

String functions are used to find, replace, edit, manipulate or test strings. See https://www.mathworks.com/help/matlab/characters-and-strings.html . For ex.

`>> strrep(mystr, '!', '!!!')` replaces every exclamation mark by three exclamation marks
(Works for both strings and character arrays.)

The basics of dealing with dates and times for arithmetic and time series plot labeling

- `datetime` creates an object that represents a point in time.
- The difference of two datetimes is a duration. You can add and subtract durations to and from each other and to and from datetimes: `datetime('now') - datetime(2018,7,1,14,2,22)` = time elapsed since 2018-07-01 14:02:22 in H:M:S .
- Useful alternative formats for points in time are datenum (for printing and plot labeling) and datevec (for data storage). datestr makes a formated string.
```
datevec(datetime(2018,7,1,14,2,22))  ← [2018 7 1 14 2  22]
datenum(datetime(2018,7,1,14,2,22))  ← 737242.5849768518
datestr(737242.5849768518)   ←  '01-Jul-2018 14:02:22'
```
- To transform a day-of-year (from 1 to 366) into a `datestr` and `datenum` using a dummy year, generate the plot, then use `datetick` to apply the datetimes to the x-axis labels.
    - dates = datestr(datenum(2016, 0, doy))
    - plot(dates, ...)
    - datetick