MATLAB Week 5

08 December 2009

Outline

- TeX and LaTeX in MATLAB
- Data Manipulation
  - Regression, integration, Fourier analysis
- Toolboxes
  - Mapping
  - Signal Processing
  - Statistics

TeX

- MATLAB has a basic built-in interpreter for plain TeX and basic LaTeX
  - Greek letters (upper and lower case), mathematic symbols can be added to text
  - Also, font type (bold, italic) and color can be changed

```
>> x = 1:0.1:2*pi;
>> y = sin(x);
>> figure(1);
>> plot(x,y,'b-');
>> axis([1 2*pi -1 1]);
>> xlabel('Greek symbols \alpha \beta \chi');
>> ylabel('superscript ^{1234} subscript _t');
>> title('\color{red}the \it\color{black}color \color{blue}can \color{yellow}\bfchange');
fx >>
```

TeX

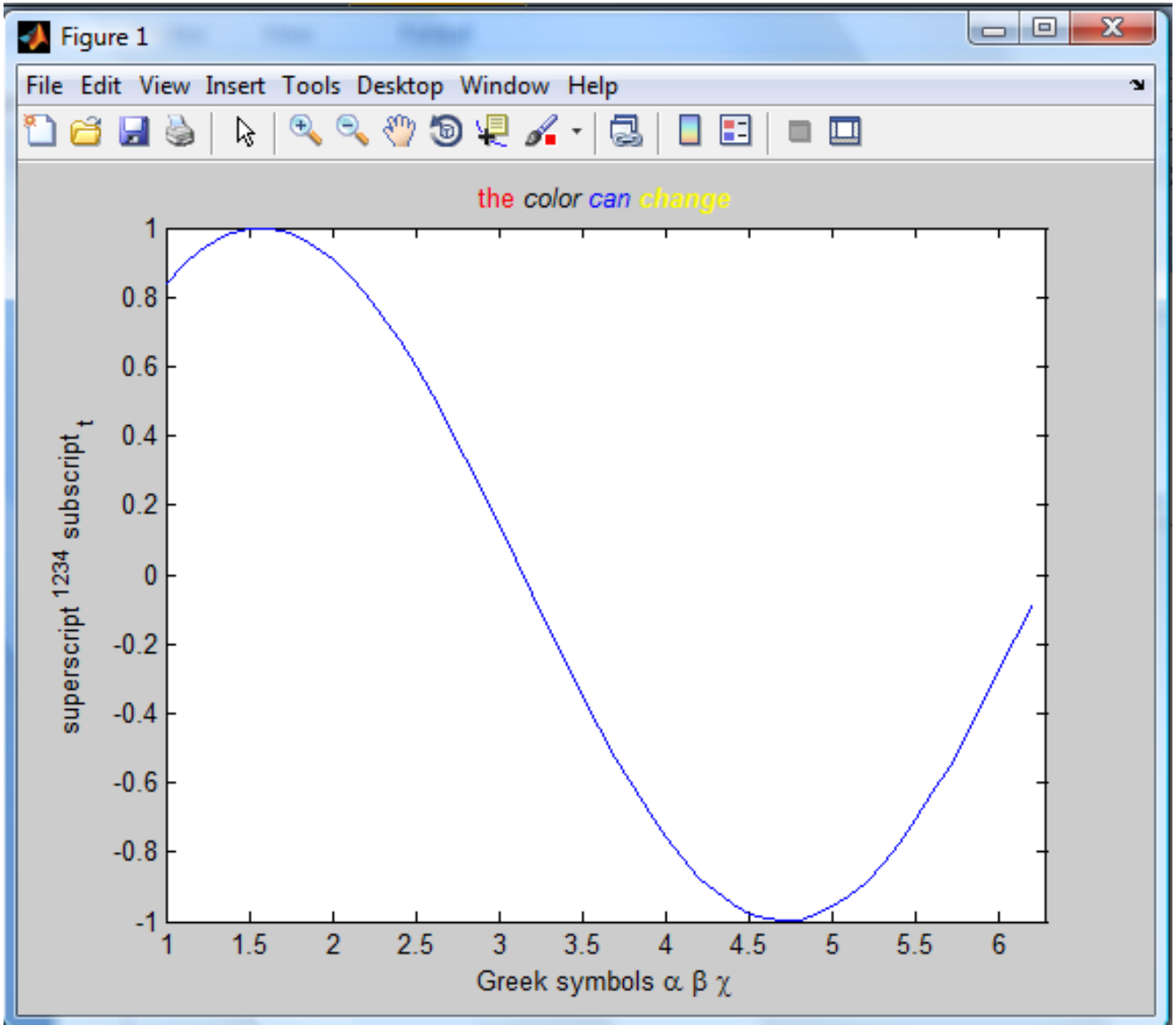


Figure 1
File Edit View Insert Tools Desktop Window Help
the color can change
superscript 1234 subscript t
Greek symbols α β χ
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
1
1.5
2
2.5
3
3.5
4
4.5
5
5.5
6

TeX

- MATLAB help describes the basic TeX commands you can use
- For more information on TeX and LaTeX
  - Donald E. Knuth, *The T_EXbook*, Addison Wesley, 1986.
  - The T_EX Users Group home page: http://www.tug.org
  - http://www.latex-project.org/

Integration

- **Z = trapz(Y)** computes an approximation of the integral of **Y** via the trapezoidal method (with unit spacing). To compute the integral for spacing other than one, multiply **Z** by the spacing increment. Input **Y** can be complex.
- **Z = cumtrapz(Y)** computes an approximation of the cumulative integral of **Y** via the trapezoidal method with unit spacing. Input Y can be complex here too.

Integration

- **q = quad(fun,a,b)** tries to approximate the integral of function fun from a to b to within an error of $1e^{-6}$ using recursive adaptive Simpson quadrature. **fun** is a function handle. Limits **a** and **b** must be finite. The function **y = fun(x)** should accept a vector argument **x** and return a vector result **y**, the integrand evaluated at each element of **x**.

Integration

Example

To compute the integral

$$\int_0^2 \frac{1}{x^3 - 2x - 5}\,dx$$

write an M-file function `myfun` that computes the integrand:

```
function y = myfun(x)
y = 1./(x.^3-2*x-5);
```

Then pass `@myfun`, a function handle to `myfun`, to `quad`, along with the limits of integration, `0` to `2`:

```
Q = quad(@myfun,0,2)

Q =

   -0.4605
```

Alternatively, you can pass the integrand to `quad` as an anonymous function handle `F`:

```
F = @(x)1./(x.^3-2*x-5);
Q = quad(F,0,2);
```

Differentiation

- **Y = diff(X)** calculates differences between adjacent elements of **X**
- **Y = diff(X,n)** applies diff recursively **n** times, resulting in the **n**th difference. Thus, **diff(X,2)** is the same as **diff(diff(X))**
- **L = del2(U)**
  L = del2(U,h)
  - Computes the discrete Laplacian of matrix **U**. **H** is an optional argument to specify spacing between calculation points

Differentiation

- **k = polyder(p)** returns the derivative of the polynomial **p**.
- **k = polyder(a,b)** returns the derivative of the product of the polynomials **a** and **b**.
- **FX = gradient(F)** where F is a vector returns the one-dimensional numerical gradient of **F**
- **[FX,FY] = gradient(F)** where **F** is a matrix returns the **x** and **y** components of the two-dimensional numerical gradient

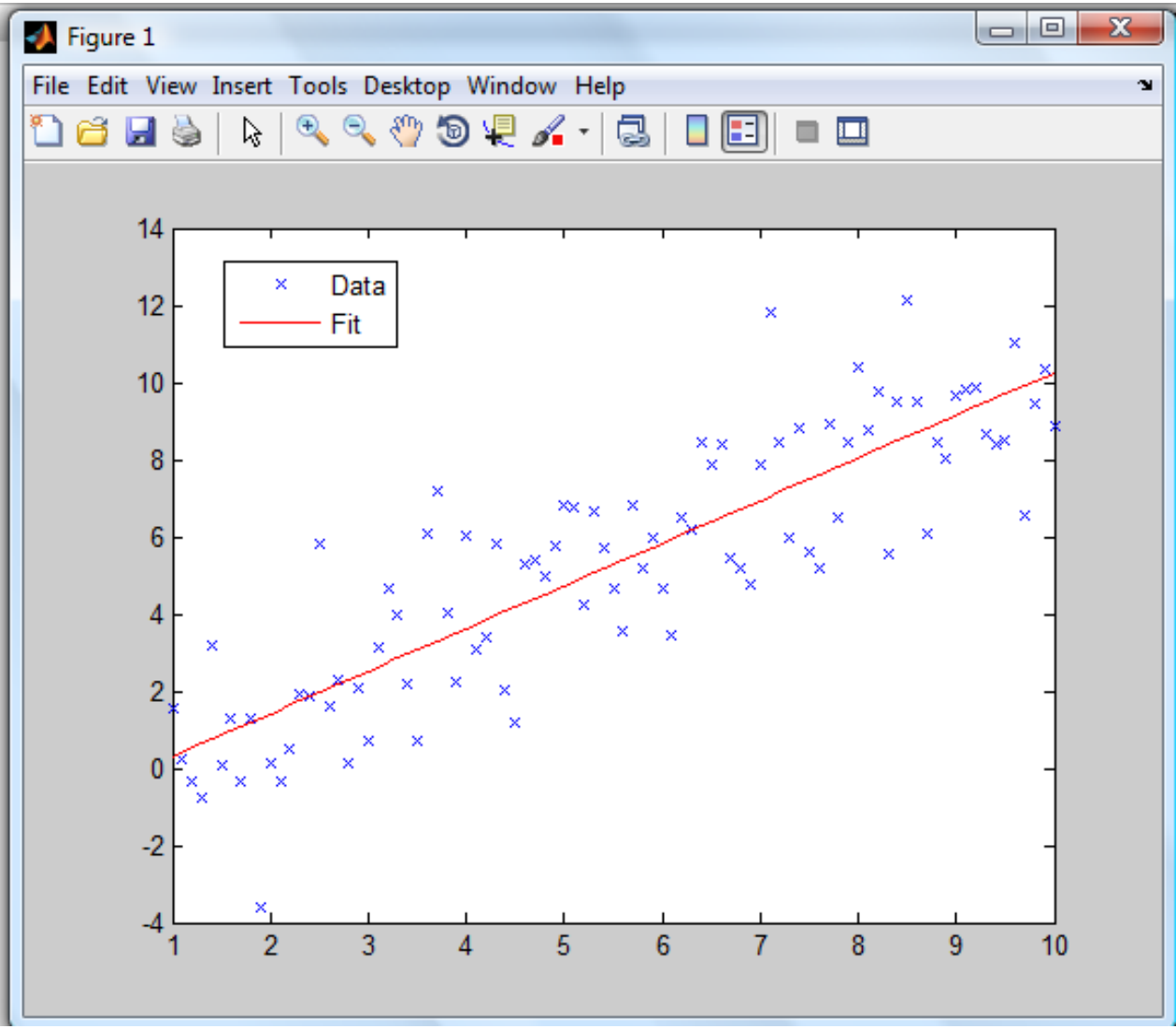
Regression

- **p = polyfit(x,y,n)** finds the coefficients of a polynomial **p(x)** of degree **n** that fits the data, **p(x(i))** to **y(i)**, in a least squares sense. The result **p** is a row vector of length **n+1** containing the polynomial coefficients in descending powers
- **y = polyval(p,x)** returns the value of a polynomial of degree n evaluated at **x**. The input argument **p** is a vector of length **n+1** whose elements are the coefficients in descending powers of the polynomial to be evaluated.

Regression

Command Window

```
>> x = 1:0.1:10;
>> points = normrnd(0,1.7,1,91)+x;
>> p = polyfit(x,points,1);
>> y = polyval(p,x);
>> figure(1);
>> plot(x,points,'bx',x,y,'r-');
>> legend('Data', 'Fit');
>>
```



Statistics Toolbox

- Normal probability, quantile-quantile, cumulative distribution plots
  - **normplot(x)**
  - **qqplot(x,y)**
  - **cdfplot(y)**
  - **probplot(y)**
    - Compares **y** to a normal distribution by default
    - **probplot('distribution', y)** compares **y** to **distribution**

Statistics Toolbox

- Statistics toolbox has many built-in distributions you can sample from
  - Beta, binomial, normal (Gaussian), lognormal, Weibull, Rayleigh, Poisson, exponential, gamma, etc
  - There are also functions to fit to those distributions to data, plot CDFs, PDFs
  - Some multivariate distributions are also included

Statistics Toolbox

- K-means clustering
  - **IDX = kmeans(X,k)**
  - This function will partition **X** into **k** clusters
  - **IDX** is a vector containing the cluster indices of each point
- Analysis of Variance (ANOVA)
  - Up to N-way ANOVA can be done
  - **anova1**, **anova2**, **anovan** are the function names

Statistics Toolbox

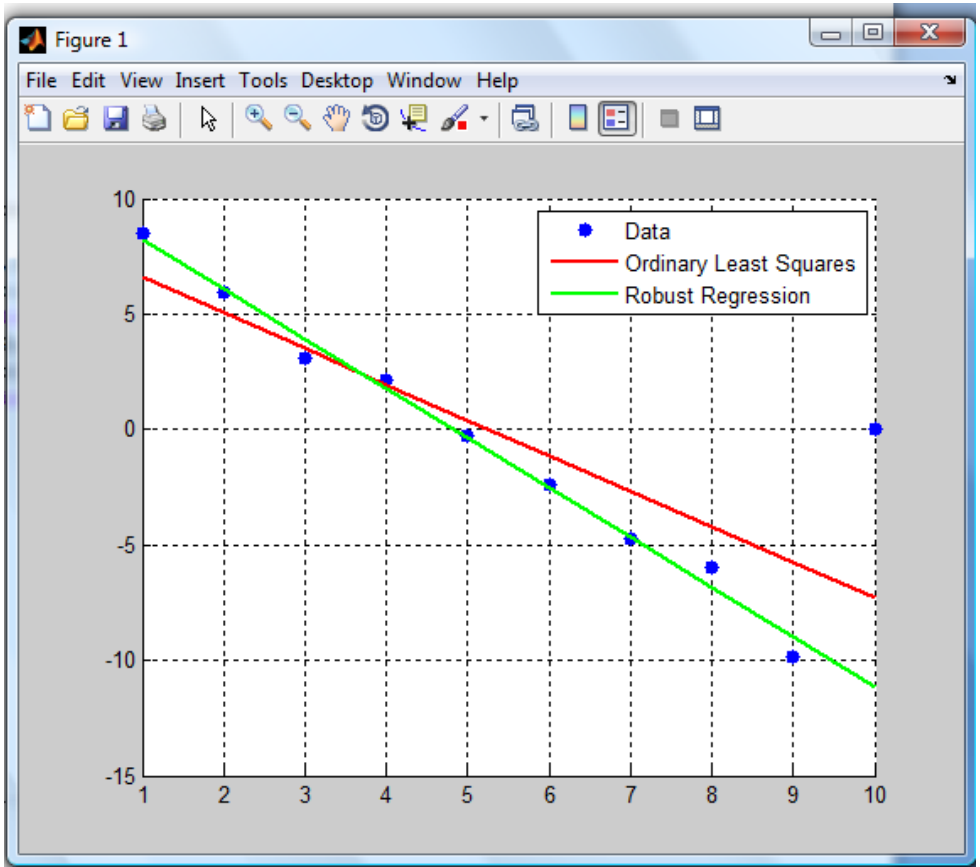
- The statistics toolbox also has:
  - Hypothesis testing
  - Multiple linear regression
  - Polynomial regression
  - Markov models
  - Etc

More Regression

- Statistics toolbox has robust regression
  - Robust regression uses an iterative method to tune the weighting function
  - Traditional least-squares method gives equal weight to each point
  - Robust method gives less weight to outliers

More Regression

```
>> x = (1:10)';
y = 10 - 2*x + randn(10,1);
y(10) = 0;
>> bls = regress(y,[ones(10,1) x]);
>> brob = robustfit(x,y);
>> scatter(x,y,'filled'); grid on; hold on
plot(x,bls(1)+bls(2)*x,'r','LineWidth',2);
plot(x,brob(1)+brob(2)*x,'g','LineWidth',2)
legend('Data','Ordinary Least Squares','Robust Regression')
```

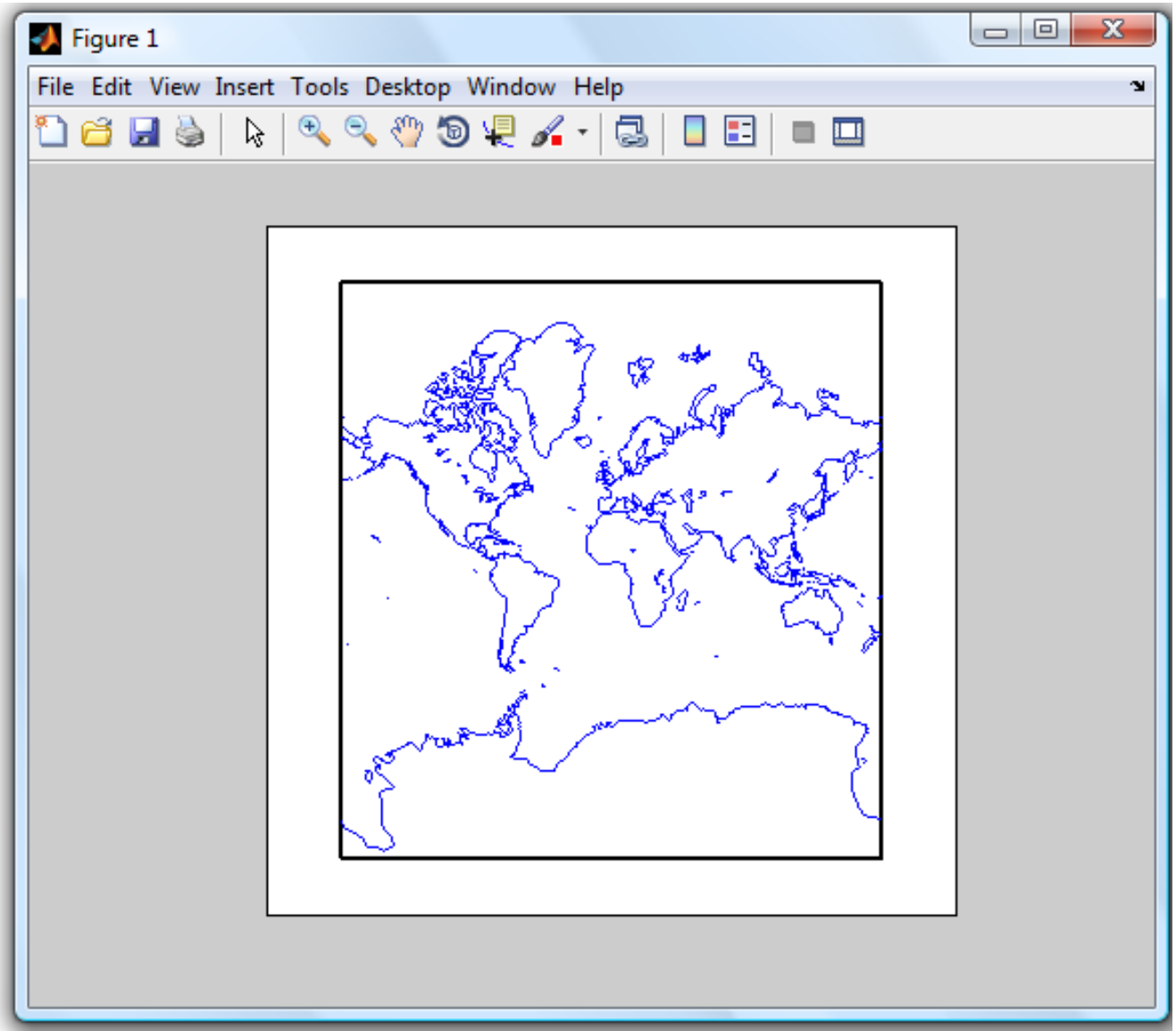


Mapping Toolbox

- Two kinds of map data handled by mapping toolbox
  - Vector and raster data
  - Vector data is a sequence of latitude, longitude pairs
    - Latitude and longitude would each be in a vector
  - Raster data is a matrix of values, each corresponding to a rectangular patch of area
    - Need to have lat, long matrices to specify where each patch belongs

Mapping Toolbox

```
>> clear all
>> load coast
>> axesm mercator
>> framem
>> plotm(lat,long)
>>
```

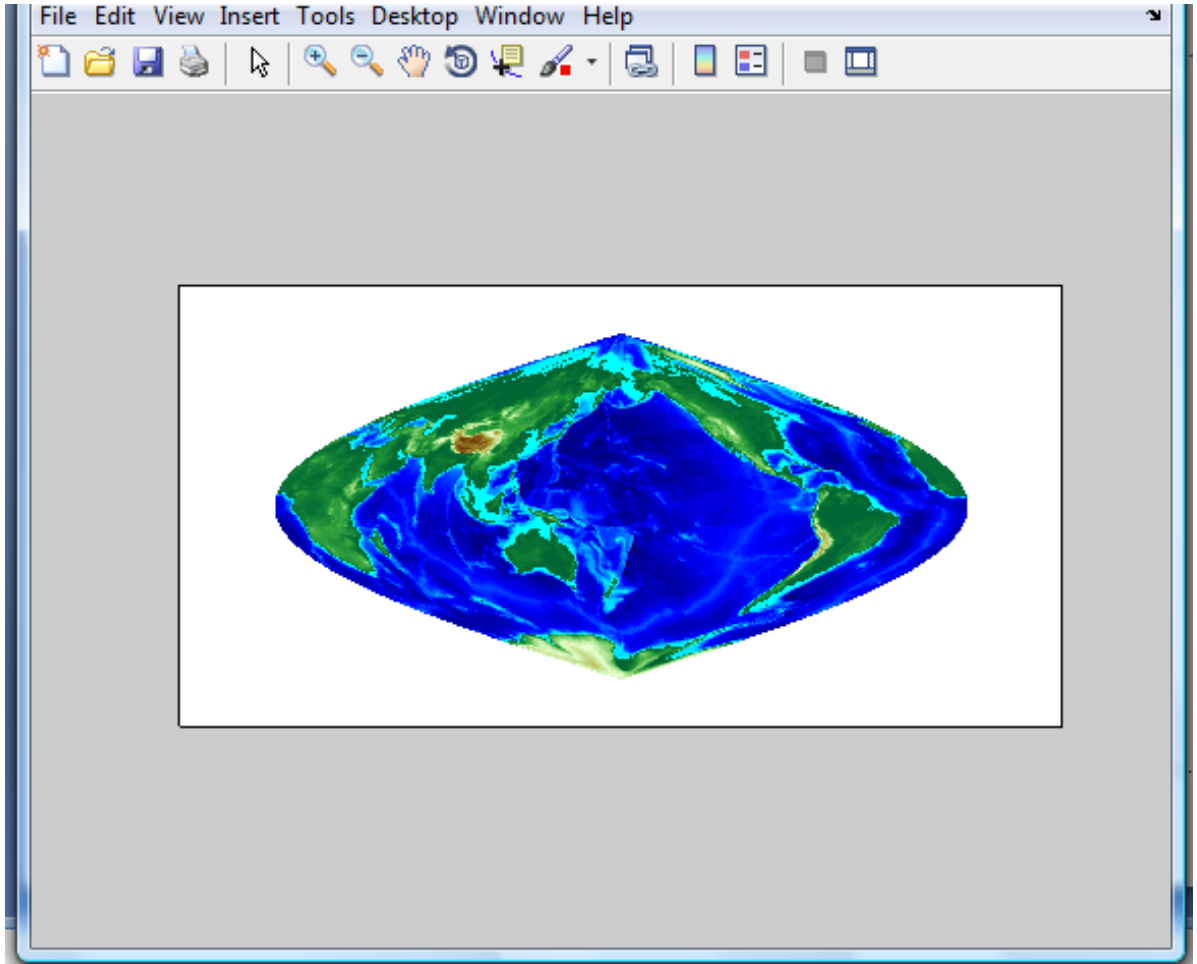


Mapping Toolbox

```
>> load topo topo
>> topoR = makerefmat('RasterSize', size(topo), ...
   'Latlim', [-90 90], 'Lonlim', [-180 180]);
>> axesm sinusoid
>> geoshow(topo,topoR,'DisplayType','texturemap')
demcmap(topo)
>> topoR

topoR =

         0    1.0000
    1.0000         0
 -180.5000  -90.5000
```



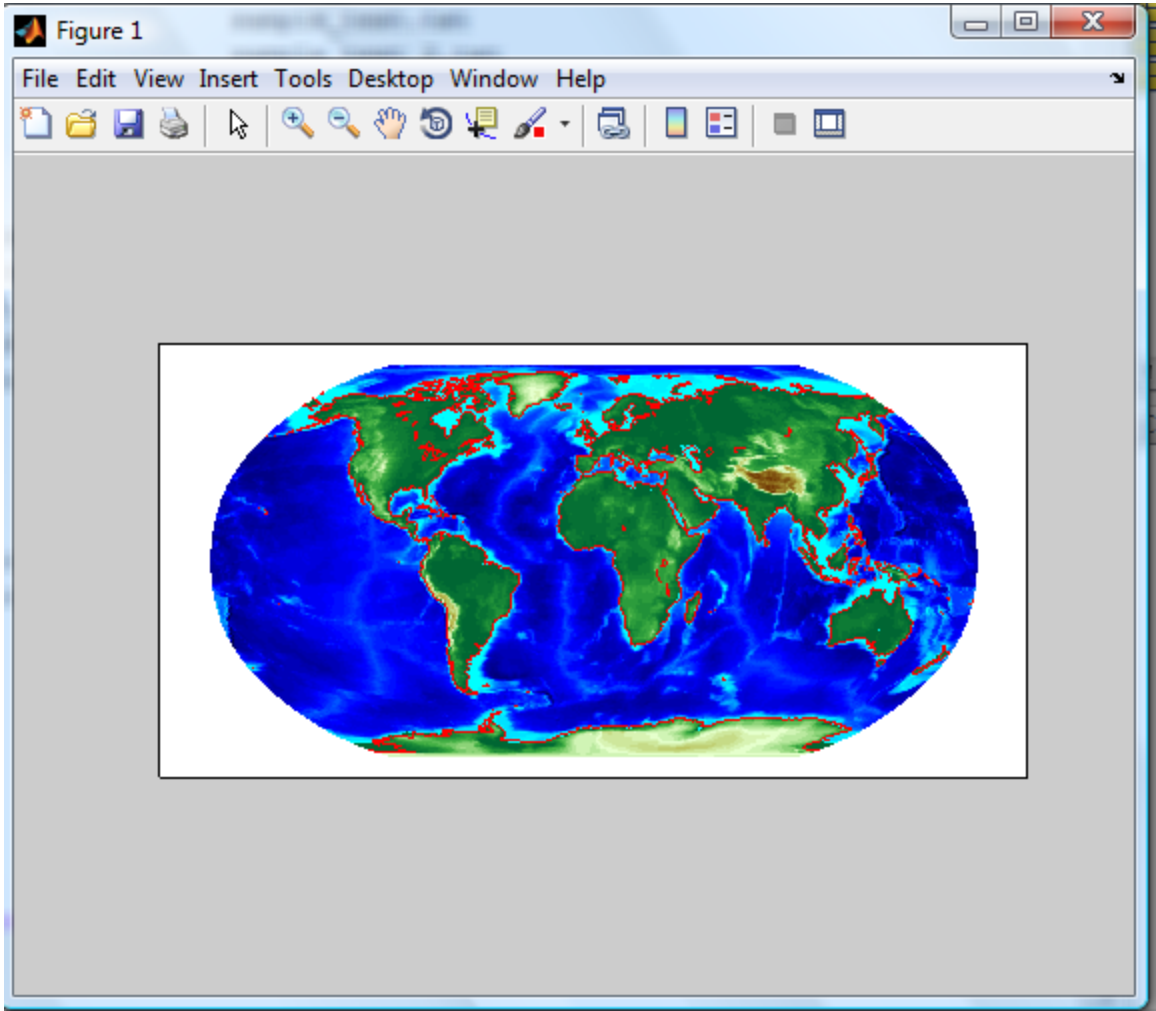
Mapping Toolbox

- **makerefmat** creates a referencing matrix for raster data
- **R = makerefmat(x11, y11, dx, dy)**
  R = makerefmat(lon11, lat11, dlon, dlat)
  R = makerefmat(param1, val1, param2, val2, ...)
  - **x11** and **y11** are the location of the first point (1,1)
  - **dx** and **dy** are the column and row resolutions respectively

Mapping Toolbox

```
>> clma
>> load coast
>> load topo
>> axesm robinson
>> geoshow(topo,topolegend,'DisplayType','texturemap')
demcmap(topo)
>> geoshow(lat,long,'Color','r')
>>
```

- Can display raster and vector data on same figure



Mapping Toolbox

- Projection types
  - Cylindrical: Mercator, Miller, Cassini, etc
  - Pseudocylindrical: Mollweide, Goode, Fournier, etc
  - Conic: Lambert conformal, Albers equal area, etc
  - Polyconic: Boone, Werner, etc
  - Azimuthal: Orthographic, Hammer, Stereographic, etc
  - 3-D globe: Displays Earth as a 3-D sphere

Mapping Toolbox

- **contourm(lat, lon, Z)** displays a contour plot of the geolocated M-by-N data grid, **Z**. **lat** and **lon** can be the size of **Z** or can specify the corresponding row and column dimensions for **Z**.
- **contourm(Z, R, n)** or **contourm(lat,lon,Z,n)** where **n** is a scalar, draws **n** contour levels.
- **contourm(Z, R, V)** or **contourm(lat, lon, Z, V)** where **V** is a vector, draws contours at the levels specified by the input vector **v**. Use **V = [v v]** to compute a single contour at level **v**.
- **contourm(..., linespec)** uses any valid LineSpec string to draw the contour lines.

Mapping Toolbox

- **surfm(lat,lon,Z)** constructs a surface to represent the data grid **Z** in the current map axes. The surface lies flat in the horizontal plane with its CData property set to **Z**. The 2-D arrays or vectors lat and lon define the latitude-longitude graticule mesh on which **Z** is displayed.

Mapping Toolbox

- **setm** is the equivalent of **set** for map axes
- **axesm(*PropertyName*,PropertyValue,...)** creates a map axes using the specified properties. Properties may be specified in any order, but the MapProjection property must be included.
  - Can just use basic syntax as seen in previous slides, but this way permits you to set up properties of the projection axes

Mapping Toolbox

- **demcmap(Z)** creates and assigns a colormap for elevation data grid **Z**. The colormap has the number of land and sea colors in the same proportions as the maximum elevations and depths in the data grid. With no output arguments, the colormap is applied to the current figure and the color axis is set so that the interface between the land and sea is in the right place.
- **demcmap(Z,ncolors)** makes a colormap with a length of **ncolors**. The default value is 64.

Mapping Toolbox

- Many other features I didn't show here
- There is a sample program on the website that goes into some detail using **axesm**, **contourm**, **surfacem** (very similar to **surfm**), and **setm** along with more options using **gca** and **gcf** and the **print** command

Signal Processsing

- **Y = fft(X)** returns the discrete Fourier transform (DFT) of vector **X**, computed with a fast Fourier transform (FFT) algorithm.
- **Y = fft2(X)** returns the two-dimensional discrete Fourier transform (DFT) of **X**, computed with a fast Fourier transform (FFT) algorithm. The result **Y** is the same size as **X**.
- **y = ifft(X)** returns the inverse discrete Fourier transform (DFT) of vector **X**, computed with a fast Fourier transform (FFT) algorithm. If **X** is a matrix, ifft returns the inverse DFT of each column of the matrix.

Signal Processing

- Signal processing toolbox has infinite impulse response (IIR) and finite impulse response (FIR) filter capabilities
- Can design filters based on classical filters like Butterworth or Bessel for IIR filters
  - FIR filters are also customizable

Signal Processing

- Waveform generation
  - **sawtooth(t)** generates a sawtooth wave with period 2π for the elements of time vector **t**. sawt**ooth(t)** is similar to **sin(t),** but creates a sawtooth wave with peaks of -1 and 1 instead of a sine wave
  - **x = square(t)** generates a square wave with period 2π for the elements of time vector **t**. **square(t)** is similar to **sin(t)**, but creates a square wave with peaks of ±1 instead of a sine wave

Signal Processing

- **C = conv(...,'shape')** returns a subsection of the convolution, as specified by the shape parameter:
  - Full: Returns the full convolution (default).
  - Same: Returns the central part of the convolution of the same size as **A**.
  - Valid: Returns only those parts of the convolution that are computed without the zero-padded edges. Using this option**, length(c)** is **max(length(a)-max(0,length(b)-1),0)**.

Wavelet Analysis

- MATLAB has a wavelet toolbox
  - Continuous and discrete wavelets
  - **coefs = cwt(x,scales,'*wname*')**
  - **wname** is the name of the wavelet transform and includes: Morlet, Daubechies, Haar, Mexican Hat, Gaussian, etc
  - **scales** is the range of scales the wavelet transform is stretched or compressed
  - **[cA,cD] = dwt(X,'*wname*')**
    - **dwt** is the discrete wavelet function

Wavelet Analysis

- Wavelets are good for many things
  - Detecting changes in signal frequency in time
  - Identifying signal breaks
  - De-noising signals
  - Image compression
- In meteorology, the first three points are especially relevant

Questions?