Data Analysis in Geophysics
ESCI 7205

Class 18

Bob Smalley

More Matlab.

Matlab

Linear Algebra (a la Matlab) Review

But first - Random numbers

When generated on the computer - how random are they?

Compare – run multiple times
Restart Matlab – run multiple times
Mac vs PC

(seed)

Computers use "pseudorandom" numbers.

From the Matlab/Mathworks web page

Pseudorandom numbers are generated by deterministic algorithms.

They are "random" in the sense that, on average, they pass statistical tests regarding their distribution and correlation.

They differ from true random numbers in that they are generated by an algorithm, rather than a truly random process.

From the Matlab/Mathworks web page

Random number generators (RNGs), like those in MATLAB are algorithms for generating pseudorandom numbers with a specified distribution.

A given number may be repeated many times during the sequence, but the entire sequence is not repeated.

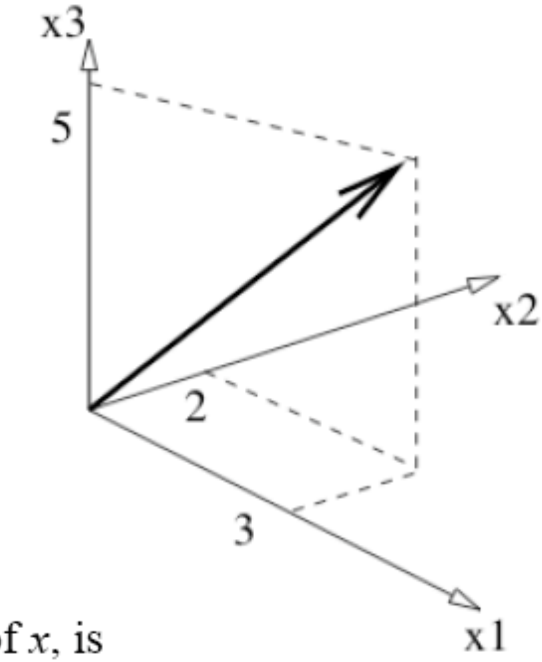
Acknowledgement
This lecture borrows heavily from online lectures/ ppt files posted by

David Jacobs at Univ. of Maryland
Tim Marks at UCSD
Joseph Bradley at Carnegie Mellon

Vectors

```
>> a=[2 3 5];
>> norm(a)
6.1644
>> norm(a)^2
38.000
```

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{pmatrix} \quad \text{e.g.} \quad \boldsymbol{x} = \begin{pmatrix} 3 \\ 2 \\ 5 \end{pmatrix}$$



The **length** of x, a.k.a. the **norm** or **2-norm** of x, is

$$\|x\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

e.g.,

$$\|x\| = \sqrt{3^2 + 2^2 + 5^2} = \sqrt{38}$$

Matrices

$$\mathbf{U} = [u_{mn}] = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ u_{21} & u_{22} & \cdots & u_{2n} \\ & & \cdots & \\ u_{m1} & u_{m2} & \cdots & u_{mn} \end{bmatrix}$$

The general matrix consists of *m* rows and *n* columns. It is also known as an *m* x *n* (read *m* by *n*) array.

Each individual number, u_{ij}, of the array is called the *element*

Elements u_{ij} where m=n is called the principal diagonal

Transpose of a Matrix

```
>> a=[6 1;2 5]
a =
     6     1
     2     5
>> a'
ans =
     6     2
     1     5
```

Transpose:

$$C_{m\times n} = A^T{}_{n\times m} \qquad (A+B)^T = A^T + B^T$$

$$c_{ij} = a_{ji} \qquad (AB)^T = B^T A^T$$

Examples:

$$\begin{bmatrix} 6 & 2 \\ 1 & 5 \end{bmatrix}^T = \begin{bmatrix} 6 & 1 \\ 2 & 5 \end{bmatrix} \qquad \begin{bmatrix} 6 & 2 \\ 1 & 5 \\ 3 & 8 \end{bmatrix}^T = \begin{bmatrix} 6 & 1 & 3 \\ 2 & 5 & 8 \end{bmatrix}$$

If $A^T = A$, we say *A* is **symmetric**.

Notice how Matlab looks like math.

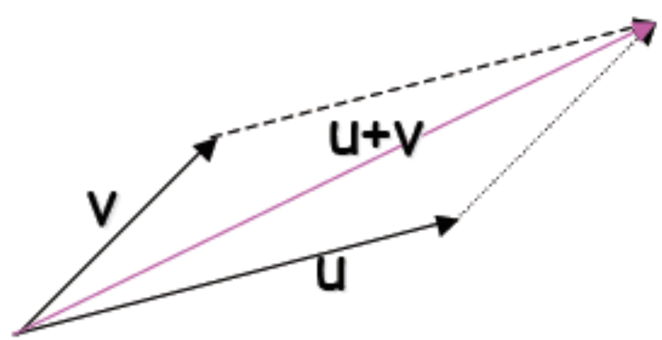
Matrix & Vector Addition

```
>> a=[1; 2]
a =
      1
      2
>> b=[3; 4]
b =
      3
      4
>> c=a+b
c =
      4
      6
>>
```

Vector/Matrix addition is associative and commutative

(A+B)+C=A+(B C);   A+B=B+A

$$u + v = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} u_1 + v_1 \\ u_2 + v_2 \end{bmatrix}$$



NO LOOPS

Looks like Math – just add them.

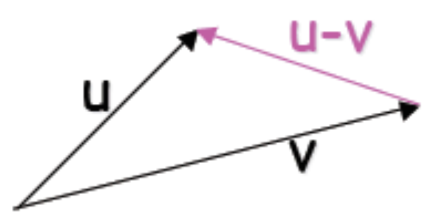
Matrix and Vector Subtraction

Same as addition

Vector/Matrix subtraction is also associative and commutative

(A-B)-C =A-(B- C);   A-B=B-A

$$u - v = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} - \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} u_1 - v_1 \\ u_2 - v_2 \end{bmatrix}$$



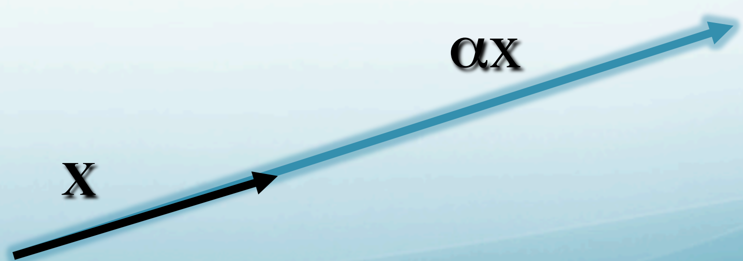
Matrix and Vector Scaling

$$\boldsymbol{z} = \alpha \boldsymbol{x}$$

for a scalar α then

$$\boldsymbol{z} = \alpha \begin{pmatrix} 3 \\ 2 \\ 5 \end{pmatrix} = \begin{pmatrix} 3\alpha \\ 5\alpha \\ 2\alpha \end{pmatrix}$$

```
>> x=[1 2 3]
x =
     1     2     3
>> y=3*x
y =
     3     6     9
>>
```



For addition and subtraction, the size of the matrices must be the same

$$A_{nm} + B_{nm} = C_{nm}$$

For scalar multiplication, the size of A_{nm} does not matter

All three of these operations do not differ from their ordinary number counterparts

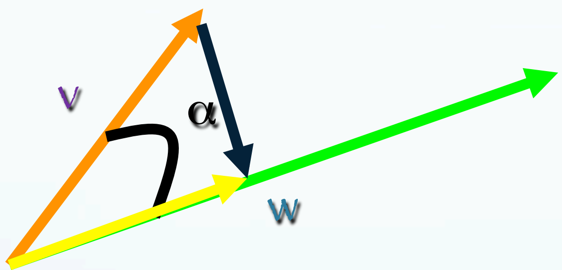
The operators work element-by-element through the array, $a_{mn}+b_{mn}=c_{mn}$

Vector Multiplication: inner or dot product

The inner product of vector multiplication is a SCALAR

$$v \cdot w = (x_1, x_2) \cdot (y_1, y_2) = \| v \| \cdot \| w \| \cos \alpha$$

$$v \cdot w = (x_1, x_2) \cdot (y_1, y_2) = x_1 y_1 + x_2 y_2$$



Projection of one vector (orange) onto another (green, result - projection – yellow).

Dot product is zero for perpendicular vectors.

The inner/dot product can be represented as row matrix multiplied by a column matrix. A row matrix can be multiplied by a column matrix, in that order, only if they each have the same number of elements!

```
>> x=[1 2]
x =
     1     2
>> y=[2 1]
y =
     2     1
>> x*y'
ans =
     4
>> y=[-2 1]
y =
    -2     1
```

```
>> x*y'
ans =
     0
>> y=[2 -1]
y =
     2    -1
>> x*y'
ans =
     0
>>
```

$$a = \begin{bmatrix} 6 \\ 2 \\ -3 \end{bmatrix} \qquad b = \begin{bmatrix} 4 \\ 1 \\ 5 \end{bmatrix}$$

$$\begin{aligned} a \cdot b &= a^T b \\ &= \begin{bmatrix} 6 & 2 & -3 \end{bmatrix} \begin{bmatrix} 4 \\ 1 \\ 5 \end{bmatrix} \\ &= 6 \cdot 4 + 2 \cdot 1 + (-3) \cdot 5 \\ &= 11 \end{aligned}$$

Several ways to properly calculate the dot product of two vectors

```
>> sum(a.*b)
```

element by element multiplication (.*), then sum the results – based on definition.

Or making it look like matrix multiplication

```
>> whos
  Name      Size            Bytes  Class     Attributes
  a         1x3                24  double
  b         1x3                24  double
>> a*b'
```

Or using Matlab function

```
>> c=dot(a,b)
```

The outer product

A column vector multiplied by a row vector. The outer product of vector multiplication is a MATRIX.

```
>> a=[6 2 -3]
a =
     6     2    -3
>> b=[4;1;5]
b =
     4
     1
     5
>> c=b*a
c =
    24     8   -12
     6     2    -3
    30    10   -15
>>
```

Matrix Multiplication

Two matrices can be multiplied together
if and only if
the number of columns in the first equals
the number of rows in the second.

$$C_{n \times p} = A_{n \times m} B_{m \times p}$$

$$c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}$$

In MATLAB, the * symbol represents matrix multiplication :

```
>> a=[1 2 3;3 2 1]
a =
    1     2     3
    3     2     1
>> b=[4 5;10 2;2 10]
b =
    4     5
   10     2
    2    10
>> c=a*b
c =
   30    39
   34    29
>>
```

```
>> a(1,:)
ans =
    1     2     3
>> b(:,1)
ans =
    4
   10
    2
>> a(1,:)*b(:,1)
ans =
   30
>> a(1,:)*b(:,2)
ans =
   39
>> a(2,:)*b(:,2)
ans =
   29
>> a(2,:)*b(:,1)
ans =
   34
```

- Matrix multiplication is not commutative!

$$A_{n\times n}B_{n\times n} \neq B_{n\times n}A_{n\times n}$$

```
>> c=a*b
c =
   30    39
   34    29
>> c=b*a
c =
   19    18    17
   16    24    32
   32    24    16
```

- Matrix multiplication is distributive and associative

$$A(B+C) = AB + BC$$

$$(AB)C = A(BC)$$

Matrices can represent sets of equations

$$
\begin{aligned}
&a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\
&a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\
&\ldots \\
&a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n
\end{aligned}
$$

What's the matrix representation?

$$
A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ & \ldots & & \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ . \\ x_n \end{bmatrix} \quad b = \begin{bmatrix} b_1 \\ b_2 \\ . \\ b_n \end{bmatrix}
$$

$$Ax = b$$

Determinant of a Matrix

```
>> a=magic(3)
a =
     8     1     6
     3     5     7
     4     9     2
>> det(a)
Ans
  -360
>>
```

Determinant: A must be square

$$\det\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{21}a_{12}$$

Example: $\det\begin{bmatrix} 2 & 5 \\ 3 & 1 \end{bmatrix} = 2 - 15 = -13$

$$\det\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = a_{11}\begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12}\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13}\begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}$$

Inverse of a Matrix

```
>> a=rand(3)
a =
    0.9649    0.9572    0.1419
    0.1576    0.4854    0.4218
    0.9706    0.8003    0.9157
>> b=inv(a)
b =
    0.3473   -2.4778    1.0874
    0.8607    2.4223   -1.2490
   -1.1203    0.5093    1.0310
>> a*b
ans =
    1.0000    0.0000   -0.0000
   -0.0000    1.0000   -0.0000
         0    0.0000    1.0000
>> b*a
ans =
    1.0000    0.0000   -0.0000
         0    1.0000    0.0000
   -0.0000   -0.0000    1.0000
>>
```

If A is a square matrix, the **inverse** of A, called A^{-1}, satisfies

$$AA^{-1} = \mathrm{I} \quad \text{and} \quad A^{-1}A = \mathrm{I},$$

Where I, the **identity matrix**, is a diagonal matrix with all 1's on the diagonal.

$$I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

If square matrix invertible, has same right and left inverse.

For a 2-D matrix

if

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$A^{-1} = \frac{\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}}{|A|}$$

the off diagonal elements change sign

the determinant

Example:

$$\begin{bmatrix} 6 & 2 \\ 1 & 5 \end{bmatrix}^{-1} = \frac{1}{28}\begin{bmatrix} 5 & -2 \\ -1 & 6 \end{bmatrix}$$

$$\begin{bmatrix} 6 & 2 \\ 1 & 5 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 6 & 2 \\ 1 & 5 \end{bmatrix} = \frac{1}{28}\begin{bmatrix} 5 & -2 \\ -1 & 6 \end{bmatrix} \cdot \begin{bmatrix} 6 & 2 \\ 1 & 5 \end{bmatrix} = \frac{1}{28}\begin{bmatrix} 28 & 0 \\ 0 & 28 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Square matrices with inverses are said to be nonsingular

Not all square matrices have an inverse. These are said to be singular.

Square matrices with determinants =0 are singular. (determinant in denominator)

Rectangular matrices are always singular.

Right- and Left- Inverse

If a matrix **G** exists such that $\mathbf{GA} = \mathbf{I}$, than **G** is a left-inverse of **A**

If a matrix **H** exists such that $\mathbf{AH} = \mathbf{I}$, than **H** is a right-inverse of **A**

Rectangular matrices may have right- or left-inverses, but they are still singular.

For

$$A = m\ x\ n,\ m > n: \quad \text{we have a left inverse } \left(A^T A\right)^{-1} A^T A = I_n,$$

$$A_{left}^{-1} = \left(A^T A\right)^{-1} A^T$$

$$A = m\ x\ n,\ n > m: \quad \text{we have a left inverse } AA^T\left(AA^T\right)^{-1} = I_m,$$

$$A_{right}^{-1} = A^T\left(AA^T\right)^{-1}$$

```
>> a=[1 2 3;4 5 6]
a =
    1     2     3
    4     5     6
>> det(a*a')
ans =
   54
>> ainv=a'*inv(a*a')
ainv =
  -0.9444    0.4444
  -0.1111    0.1111
   0.7222   -0.2222
```

```
>> a*ainv
ans =
    1.0000   -0.0000
   -0.0000    1.0000
>> det(a'*a)
ans =
     0
```

Right inverse exists,
but left doesn't.

Matrix Division in Matlab

In ordinary math, division `(a/b)` can be thought of as `a*(1/b) or a*b`$^{-1}$.

There really is no such thing as matrix division in any simple sense.

A unique inverse matrix of **b**, **b**$^{-1}$, only potentially exists if **b** is square.

Also, matrix multiplication is not communicative, unlike ordinary multiplication.

Matrix Division in Matlab

`/` : `B/A` (right division) is roughly the same as `B*inv(A)`.

`A` and `B` must have the same number of columns for right division.

More precisely, `B/A = (A'\B')'`.

Matrix Division in Matlab

\ : If **`A`** is a square matrix, **`A\B`** (left division) is roughly the same as **`inv(A)*B`**, except it is computed in a different way.

`A` and **`B`** must have the same number of rows for left division.

```
a =
    1     2
    3     4
>> b
b =
    1
    2
>> c=a\b
c =
         0
    0.5000
```

```
>> ainv=inv(a)
ainv =
   -2.0000    1.0000
    1.5000   -0.5000
>> ainv*b
ans =
         0
    0.5000
>> a*c
ans =
    1
    2
```

Matrix Division in Matlab

\ : If **A** is an **m-by-n** matrix (not square) and **b** is a matrix of **m** rows, **Ax=b** is solved by least squares using **A\b** (left division).

```
>> A
A =
1.0000       0         0
1.0000  1.0000    0.5000
1.0000  2.0000    2.0000
1.0000  3.0000    4.5000
1.0000  4.0000    8.0000
1.0000  5.0000   12.5000
>> b
b =
        1001
        1093
        1177
        1245
        1305
        1349
```

```
>> x_=A\b
x_ =
   1.0e+03 *
    1.0004
    0.0998
   -0.0120
>> (A*x_-b)/mean(b)
ans =
   -0.0005
    0.0011
   -0.0008
    0.0008
   -0.0011
    0.0005
```

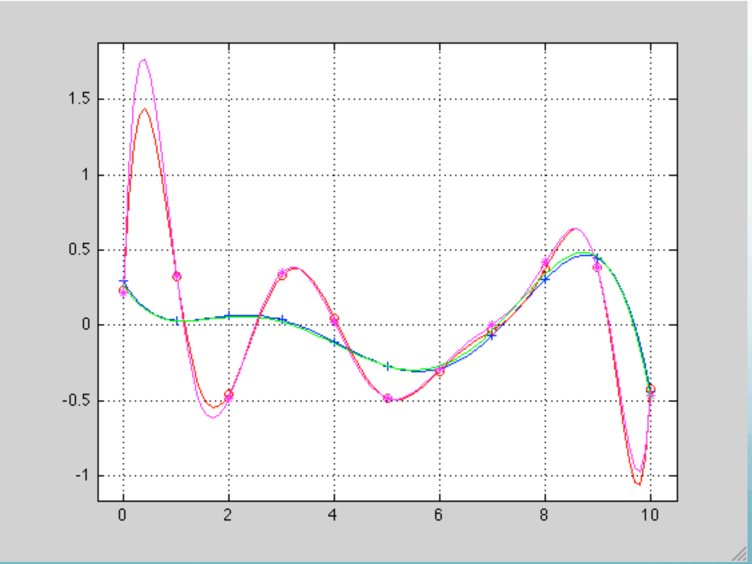
Matlab also has routines to do polynomial fits

(positive powers only).

Data – 11 points (red circles)

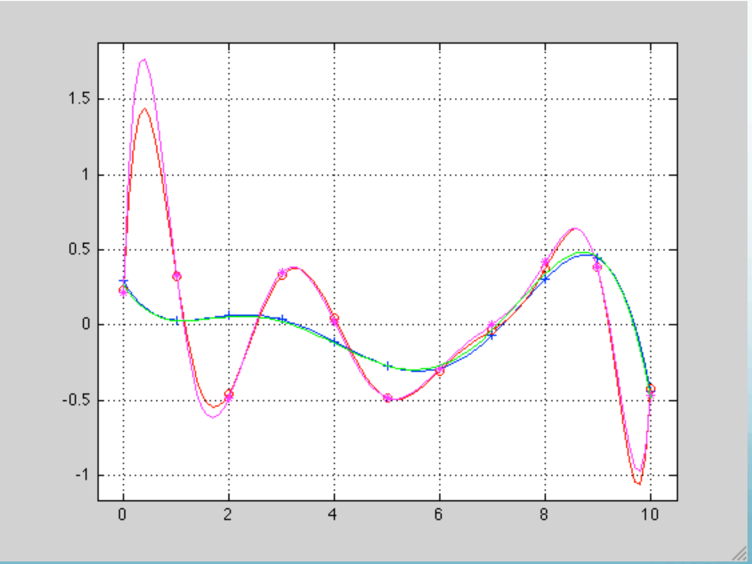
5th order polynomial fit to 11 points – fewer parameters than data – get LS fit (blue line, blue '+') – does not go through data points

(but misfit "minimized").



Data – 11 points (red circles)

10^{th} order polynomial fit to 11 points – same number parameters as data – get exact solution (red line). Goes though each point exactly.

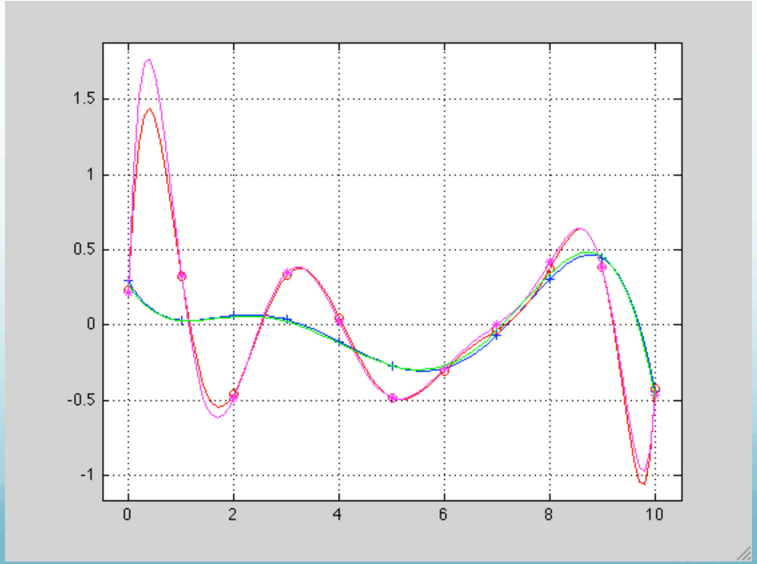


Now add some noise.

Data – 11 points (red circles)

Magenta and green – 5^{th} and 10^{th} order polynomials fit to data with 10% noise.

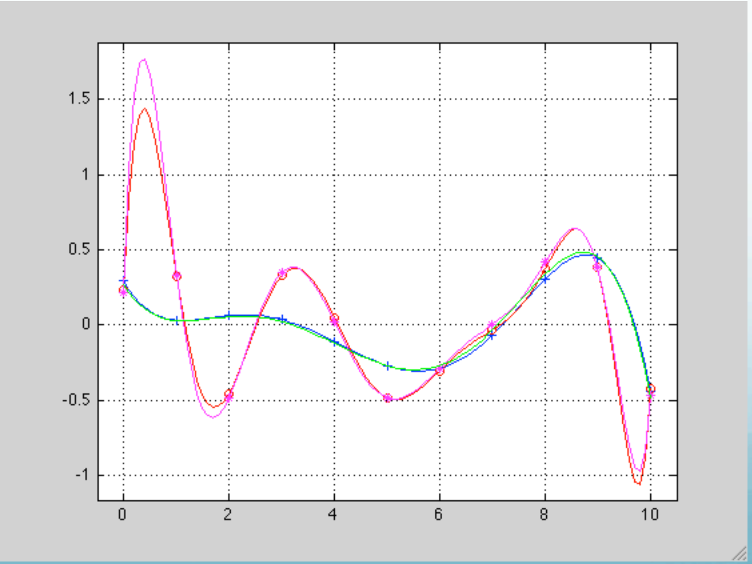
The fits to the noisy and perfect data look pretty much the same when plotted.



```
>> poly_demo
p5 =
   -0.0010    0.0215   -0.1629    0.4980   -0.6166    0.2926

pn5 =
   -0.0009    0.0209   -0.1559    0.4696   -0.5767    0.2733

p10 =
  Columns 1 through 11
    0.0000   -0.0003    0.0051   -0.0474    0.2251   -0.3195   -1.7232    8.6334  -14.0467    7.3647    0.2333
pn10 = ?     ?
  Columns 1 through 11
    0.0000   -0.0002    0.0024   -0.0145   -0.0298    0.9338   -5.6002   15.8159  -21.1548   10.1628    0.2137
```

The models (the values for polynomial coefficients), however, are quite different.

Compare "stability" of the solutions.



Array Operators (review)

\+ Addition

\- Subtraction

.* Element-by-element multiplication

./ Element-by-element division.

(A./B: divides A by B by element)

.\ Element-by-element left division

(A.\B divides B by A by element)

.^ Element-by-element power

.' Unconjugated array transpose

(does not take complex conjugate, unlike a regular [no dot] matrix transpose)

Some Special Matrices

<u>Square matrix</u>: **m** (# rows) = **n** (# columns)

<u>Symmetric matrix</u>: subset of square matrices where $A^T = A$

<u>Diagonal matrix</u>: subset of square matrices where elements off the principal diagonal are zero, $a_{ij} = 0$ if $i \neq j$

<u>Identity or unit matrix</u>: special diagonal matrix where all principal diagonal elements are 1

```
>> a=[1 2 3;4 5 6;7 8 9]
a =
    1     2     3
    4     5     6
    7     8     9
>> c=trace(a)
c =
   15
>> a=[.96 -.28; .28 .96]
a =
   0.9600   -0.2800
   0.2800    0.9600
>> inv(a)
ans =
   0.9600    0.2800
  -0.2800    0.9600
>> a'*a
ans =
    1     0
    0     1
>>
```

trace of a Matrix is Tr(A) =

$$\sum_{i=1}^{N} a_{ii}$$

(the sum of the diagonal entries)

In matlab use trace(A)

a matrix A is **orthonormal** if

$$A^T = A^{-1}$$

and in this case

$$AA^T = I$$

Misc stuff

MATLAB

`lookfor` command – to look for commands based on "keyword"

(searches all m files in path, including your files, for the keyword).

`what` command – lists Matlab related files (returns structure with fields for m, mat, mex, mdl, classes, and packages files)., Help window (pull down menu).

Helpdesk (internet)

http://www.mathworks.com/access/helpdesk/help/helpdesk.shtml

workspace

```
>> help workspace
 WORKSPACE Open Workspace browser to manage workspace WORKSPACE
Opens the Workspace browser with a view of the variables in the
current Workspace.  Displayed variables may be viewed,
manipulated, saved, and cleared.
```

path

```
>> help path
 PATH Get/set search path.
    PATH, by itself, prettyprints MATLAB's current search path.
The initial search path list is set by PATHDEF, and is perhaps
individualized by STARTUP.
    P = PATH returns a string containing the path in P. PATH(P)
changes the path to P.  PATH(PATH) refreshes MATLAB's view of the
directories on the path, ensuring that any changes to non-toolbox
directories are visible.
    PATH(P1,P2) changes the path to the concatenation of the two
path strings P1 and P2.  Thus PATH(PATH,P) appends a new
directory to th current path and PATH(P,PATH) prepends a new
directory.  If P is already on the path, then PATH(PATH,P) moves
P to the end of the path, and similarly, PATH(P,PATH) moves P to
the beginning of the path.
    For example, the following statements add another directory
to MATLAB’s search path on various operating systems:
      Unix:      path(path,'/home/myfriend/goodstuff')
      Windows:   path(path,'c:\tools\goodstuff’)
```

format command

```
>> help format
. . .
FORMAT Set output format.
. . .
FORMAT does not affect how MATLAB computations are done.
```

To separate multiple commands on one line use "`;`" for no output, and '`,`' for output

Command line editing

`arrows`: move cursor by character

`ctrl arrows l and r`: move cursor by word

`ctrl a, e`: move cursor to beginning, end line

`ctrl u, d, h, k`: clear line, delete char at cursor, delete char before cursor, delete to end of line.

Running an **m** file from the command line (should not be an interactive **m** file)

```
>> Matlab < somefile.m
```

Don’t show output on terminal (send to bit bucket), run in background to not lock up terminal

```
>> Matlab -nosplash < eig_mov.m > /nl &
```

```
>> a=[1 2 3;4 5 6;7 8 9]
a =
   1     2     3
   4     5     6
   7     8     9
>> c=trace(a)
c =
  15
```

```
>> a=[.96 -.28; .28 .96]
a =
  0.9600   -0.2800
  0.2800    0.9600
>> inv(a)
ans =
  0.9600    0.2800
 -0.2800    0.9600
>> a'*a
ans =
   1     0
   0     1
>>
```

trace of a Matrix is Tr(A) =

$$\sum_{i=1}^{N} a_{ii}$$

(the sum of the diagonal entries)

In matlab use trace(A)

a matrix A is **orthonormal** if

$$A^T = A^{-1}$$

and in this case

$$AA^T = I$$

Block matrices

```
>> b=[2 2;1 3]
b =
     2     2
     1     3
>> c=[0 2 3; 5 4 7]
c =
     0     2     3
     5     4     7
>> d=[1 0]
d =
     1     0
>> e=[-1 6 0]
e =
    -1     6     0
>> a=[b c;d e]
a =
     2     2     0     2     3
     1     3     5     4     7
     1     0    -1     6     0
```

Linear Dependence

- A set of vectors is **linearly dependent** if one of the vectors can be expressed as a linear combination of the other vectors.

Example:

$$\begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}2\\1\\0\end{bmatrix}$$

a b c

2a + 1b = c

Linear Independence

- A set of vectors is **linearly independent** if none of the vectors can be expressed as a linear combination of the other vectors.

Example:

$$\underset{a}{\begin{bmatrix}1\\0\\0\end{bmatrix}}, \underset{b}{\begin{bmatrix}0\\1\\0\end{bmatrix}}, \underset{c}{\begin{bmatrix}2\\1\\3\end{bmatrix}}$$

There is no simple, linear combination of **a** and **b** what will produce **c**.

Rank of a matrix

- The **rank** of a matrix is the number of linearly independent columns of the matrix.

  Examples:

  $$\begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{has rank 2}$$

  $$\begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{has rank 3}$$

- Note: the rank of a matrix is also the number of linearly independent *rows* of the matrix.

```
>> a=[1 2; 3 4]
a =
     1     2
     3     4
>> rank(a)
ans =
     2
>> rref(a)
ans =
     1     0
     0     1
>> help rref
 rref   Reduced row echelon form.
    R = rref(A) produces the reduced row echelon form of A.
```

```
>> a=[1 2 3;4 5 6;7 8 9]
a =
     1     2     3
     4     5     6
     7     8     9
>> b=inv(a)
Warning: Matrix is close to singular or badly scaled. Results may
be inaccurate. RCOND =  1.541976e-18.
b =
   1.0e+16 *
   -0.4504    0.9007   -0.4504
    0.9007   -1.8014    0.9007
   -0.4504    0.9007   -0.4504
>> a*b
ans =
     2     0     2
     8     0     0
    16     0     8
>> b*a
ans =
     4     0     0
     0     8     0
     4     0     0
```

```
>> a=[1 2 3;4 5 6;7 8 9]
a =
     1     2     3
     4     5     6
     7     8     9
>> det(a)
ans =
   6.6613e-16
>> 1*(5*9-6*8)-2*(4*9-7*6)+3*(4*8-7*5)
ans =
     0
>> c=a(:,1:2)\a(:,3)
c =
   -1.0000
    2.0000
>> rank(a)
ans =
     2
>> rref(a)
ans =
     1     0    -1
     0     1     2
     0     0     0
>>
```

Computers versus Math: Determinant is actually zero, inverse does not exist.

There are a lot of matrix math functions

```
>> help matfun
  Matrix functions - numerical linear algebra.

  Matrix analysis.
    norm        - Matrix or vector norm.
    normest     - Estimate the matrix 2-norm.
    rank        - Matrix rank.
    det         - Determinant.
    trace       - Sum of diagonal elements.
    null        - Null space.
    orth        - Orthogonalization.
    rref        - Reduced row echelon form.
    subspace    - Angle between two subspaces.

  Linear equations.
    / and /     - Linear equation solution; use "help slash".
    linsolve    - Linear equation solution with extra control.
    inv         - Matrix inverse.
    rcond       - LAPACK reciprocal condition estimator
    cond        - Condition number with respect to inversion.
    condest     - 1-norm condition number estimate.
    normest1    - 1-norm estimate.
```

```
  cholinc   - Incomplete Cholesky factorization.
  ldl       - Block LDL' factorization.
  lu        - LU factorization.
  luinc     - Incomplete LU factorization.
  qr        - Orthogonal-triangular decomposition.
  lsqnonneg - Linear least squares with nonnegativity
                                             constraints.
  pinv      - Pseudoinverse.
  lscov     - Least squares with known covariance.

Eigenvalues and singular values.
  eig       - Eigenvalues and eigenvectors.
  svd       - Singular value decomposition.
  gsvd      - Generalized singular value decomposition.
  eigs      - A few eigenvalues.
  svds      - A few singular values.
  poly      - Characteristic polynomial.
  polyeig   - Polynomial eigenvalue problem.
  condeig   - Condition number with respect to eigenvalues.
  hess      - Hessenberg form.
  schur     - Schur decomposition.
  qz        - QZ factorization for generalized eigenvalues.
```

```
   ordschur  - Reordering of eigenvalues in Schur decomposition.
   ordqz     - Reordering of eigenvalues in QZ factorization.
   ordeig    - Eigenvalues of quasitriangular matrices.

 Matrix functions.
   expm      - Matrix exponential.
   logm      - Matrix logarithm.
   sqrtm     - Matrix square root.
   funm      - Evaluate general matrix function.

 Factorization utilities
   qrdelete  - Delete a column or row from QR factorization.
   qrinsert  - Insert a column or row into QR factorization.
   rsf2csf   - Real block diagonal form to complex diagonal
                                                      form.
   cdf2rdf   - Complex diagonal form to real block diagonal
                                                      form.
   balance   - Diagonal scaling to improve eigenvalue accuracy.
   planerot  - Givens plane rotation.
   cholupdate - rank 1 update to Cholesky factorization.
   qrupdate   - rank 1 update to QR factorization.

>>
```

MATLAB

Data Analysis

Flow Charts

2 tasks

Understanding How a Process Works
(if you don't know how to do it, you can't write a program to do it)

Communicating How a Process Works
(translating it into computer code, communicating to the computer.)

A flow chart can therefore be used to:

Define and analyze processes;

Build a step-by-step picture of the process for analysis, discussion, communication, and coding;

and

Define, standardize or find areas for improvement in a process.

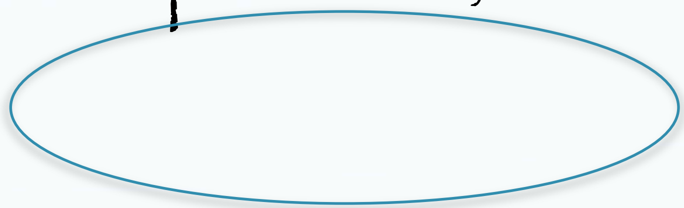
Also

by conveying the information or processes in a step-by-step flow, you can then concentrate more intently on each individual step,

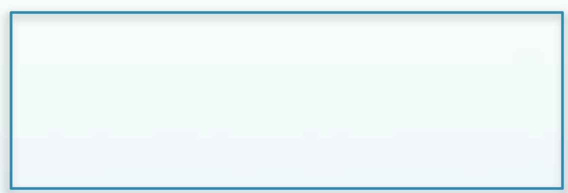
without feeling overwhelmed by the bigger picture.

Most flow charts are made up of three main types of symbol:

Elongated circles, signify start or end of a process;



Rectangles, show instructions or actions;

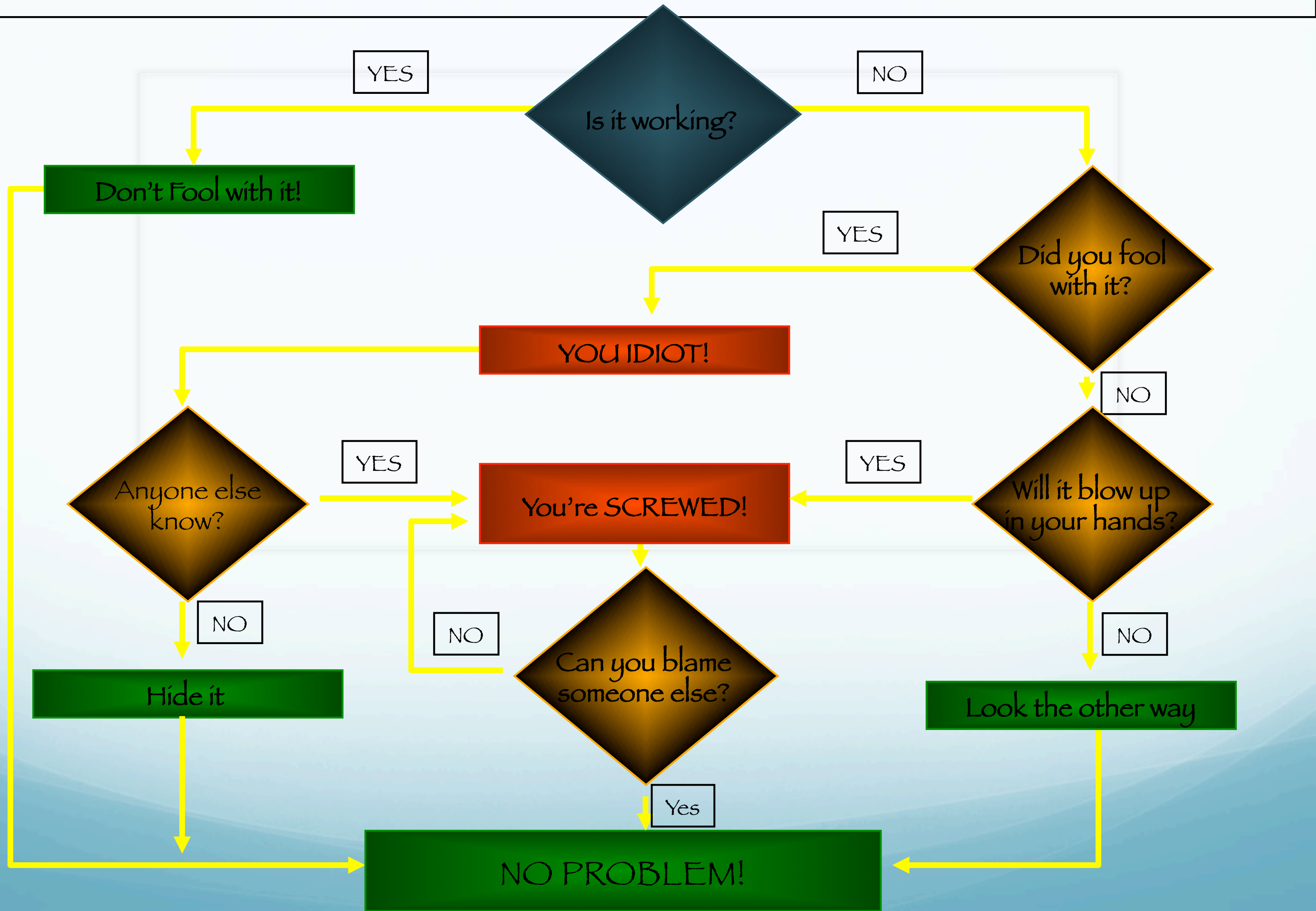


Diamonds, show decisions to be made

Within each symbol, write down what the symbol represents. This could be the start or finish of the process, the action to be taken, or the decision to be made.

Symbols are connected one to the other by arrows, showing the flow of the process.

Worlds most famous Flowchart: General Flowchart For Problem Resolution –



Coding and Flow Charts

Today's presentation will focus on understanding Chuck's Matlab script for polarization analysis using 3 component recordings of body and/or surface waves

Chucks example codes:

http://www.ceri.memphis.edu/people/langston/Matlab/programming.html

GOAL: solve for polarization using 3 component seismic data

Starting data: 3 component single station SAC formatted data

Result: Identify the azimuth(s) of the primary wave(s) recorded in the data

How to we get from A to B?

Principal Component Analysis

Principal component analysis (PCA) is a vector space transform often used to reduce multidimensional data sets to lower dimensions for analysis.

Principal Component Analysis

PCA involves the calculation of the eigenvalue decomposition of a data covariance matrix or singular value decomposition of a data matrix, usually after mean centering the data for each attribute.

Its operation can be thought of as revealing the internal structure of the data in a way which best explains the variance in the data.

What we are looking for:

New set of axis (basis) that maximizes the correlation of HT(Z) with R, and minimizes the correlations between both HT(Z) and R with T.

We are not using the full power of PCA, since we already have some model for the result of the analysis

(and have therefore preprocessed the data by taking the Hilbert transform of the z component).

What is the idea?

Seismic waves are polarized

P wave longitudinal (V and R)

S wave transverse with SH and SV polarizations (T, V and R)

Rayleigh waves (V and R)

Love waves (T).

If we take a short time period we can think of each component as a vector of **n** terms.

If we take the dot product of each vector with itself and with the other two components we can find the "angle" between them.

We can also make these dot products by making a **3xn** array using each seismogram as a row.

Multiplying this array with its transpose (an array where each seismogram is a column) results in a **3x3** matrix with the various dot products in the elements of the matrix.

(there are only 6 unique ones, 3 are redundant, since the resultant matrix is symmetric since $a \bullet b = b \bullet a$)

Now find the eigenvectors and eigenvalues of this matrix.

From the eigenvectors we can make a rotation matrix that will rotate our matrix to a diagonal matrix.

The off diagonal elements are now all zero and from the geometric interpretation of the dot product this means that the two vectors used to make that dot product are perpendicular.

So we can rotate the original horizontal components into a new set of seismograms rotated to the principal directions defined by the eigenvectors.

The dot products of the off diagonal terms will now be zero, indicating the vectors are perpendicular.

GOAL: Solve for polarization

Data Preparation

- load SAC data
- remove mean
- plot waveforms
- filter waveforms to highlight waves of interest
- plot filtered waveforms

Main Code

- principal component analysis using 3-component data
- note, this technique requires zero-mean data

Results

- plot azimuths of eigenvectors
- plot azimuths exceeding 50% of maximum value

Step 1: Data Preparation

Load SAC data

3 files (Z,E,N)…. suggests using a subroutine

Provide station name

Remove the mean

Plot waveforms vs time

Filter waveforms to highlight waves of interest

Design a filter; allow flexible input of corner info

Work on 3 files…suggests a subroutine

Plot filtered waveforms

Create function 'polarize'

```
function polarize(station,delt,ttot,twin,hilb,flp,fhi)
%
%  function polarize(station,delt,ttot,twin,hilb,flp,fhi)
%
%  Program to read in 3 component waveform data
%  Create the covariance matrix for a moving time window
%  Find the principal components and infer polarization
%
%   input:
%          station = station name for sacfile prefix
%          delt = sampling interval
%          ttot = total number of seconds to analyze in traces
%          twin = time window length, each time shift will be 1/2 of the
%                  window length
%          hilb = 0, no hilbert transform of vertical component
%               = 1, hilbert transform
%          flp = low passband corner frequency of a 2nd order butterworth
%                filter used to filter the data, if 0, then no filtering
%          fhi = hi passband corner frequency of the filter
```

Loading SAC data

Many Matlab scripts exist to read in SAC data.

I modified one version so it is

- not sensitive to byte order
-returns the data, plus: npts, delta, and begin point of the SAC file

- data is a column vector

Read the data

```
[e,npts,delt,date,hour,minu,seco,fname]=get_sac_fn('../
      2007.308.20.37.16.3856.IU.SBA.00.BHE.R.SAC');

[n,npts,delt,date,hour,minu,seco,fname]=get_sac_fn('../
      2007.308.20.37.16.3856.IU.SBA.00.BHN.R.SAC');

[z,npts,delt,date,hour,minu,seco,fname]=get_sac_fn('../
      2007.308.20.37.16.3856.IU.SBA.00.BHZ.R.SAC');
```

Removing the data mean

We need to remove the mean of the data for principal component analysis (PCA).

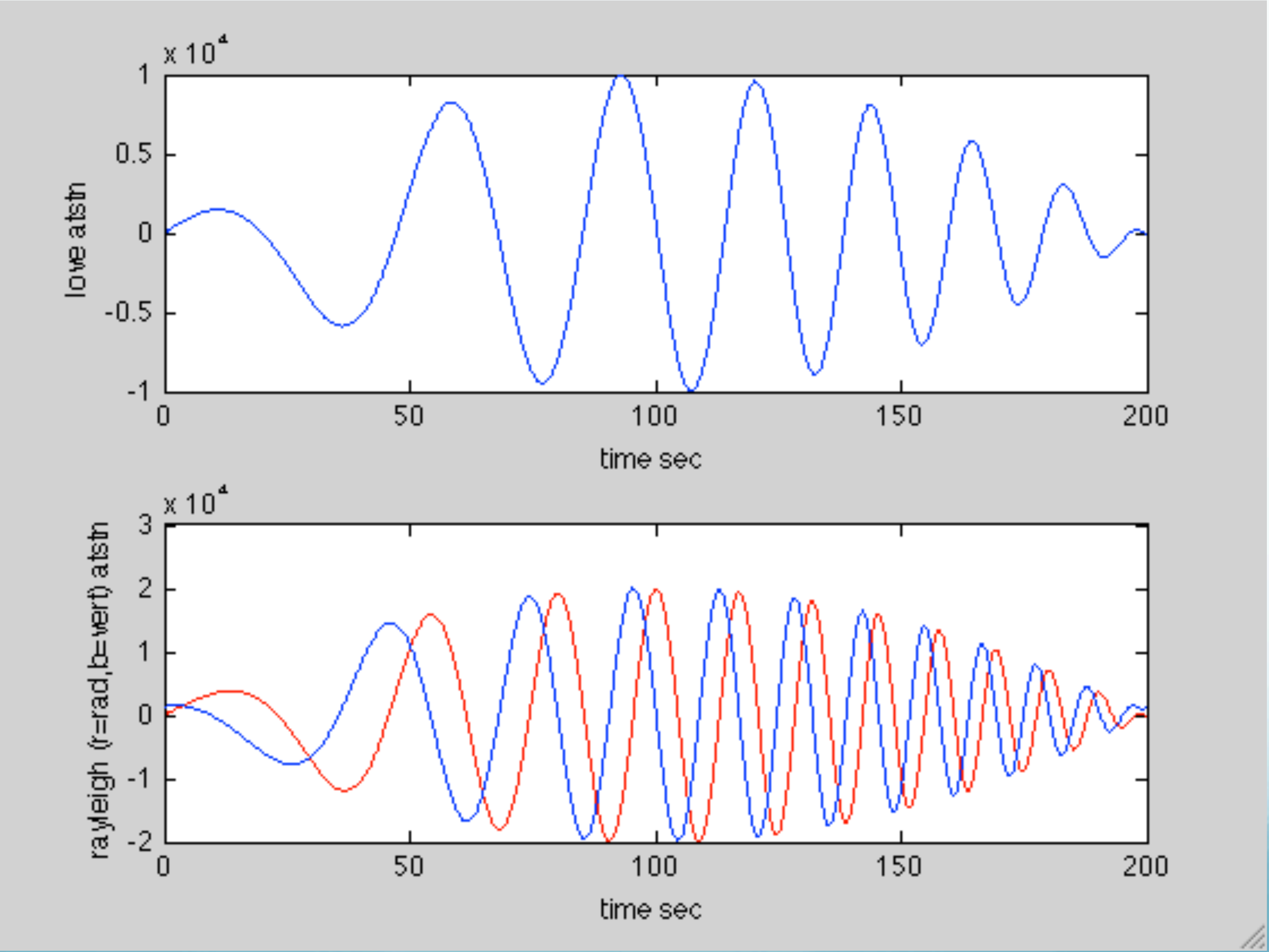
We also need to transpose the column vector data into row vectors.

```
e=dmean(e');            % remove the mean from each
n=dmean(n');            % and transpose the data
z=dmean(z');
```

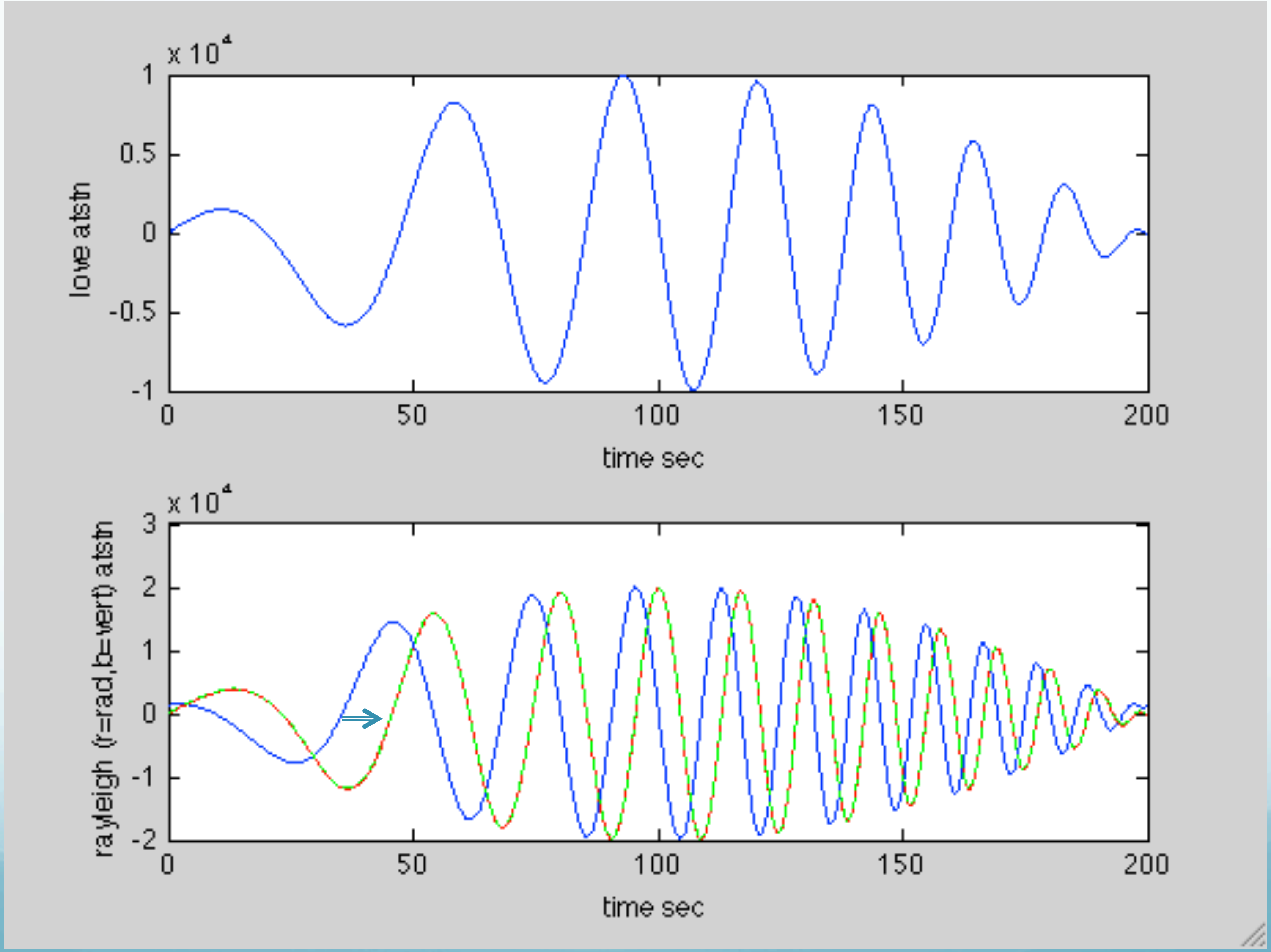
subroutine: dmean

```
function [a]=dmean(b)
%
% [a]=dmean(b)
%
%     Remove the mean from a row vector
m=mean(b);
a=b-m;
return;
```

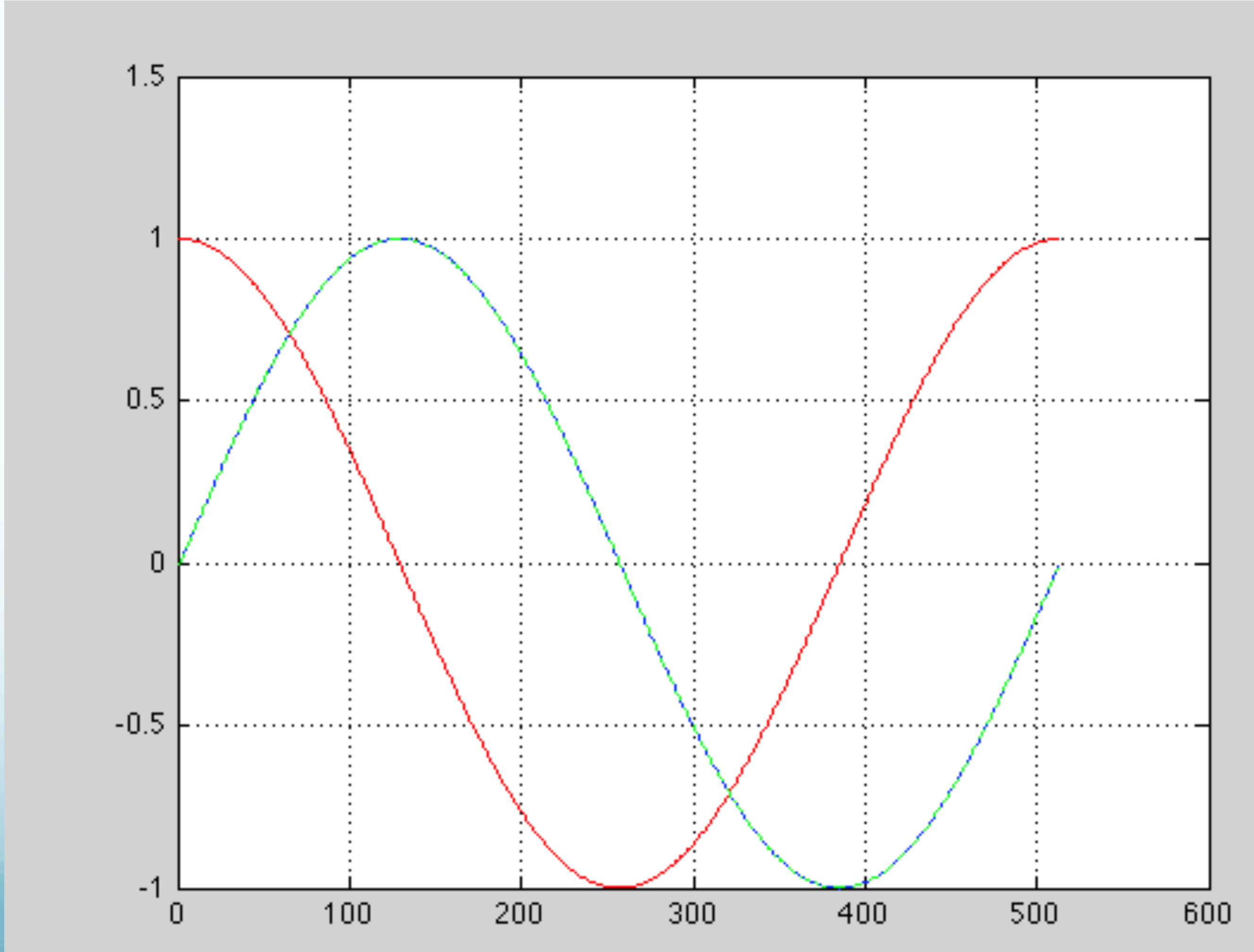
Make Love and Rayleigh waves (Z, R and T)



Rayleigh R and Z related by Hilbert X-form (90° phase shift, blue trace is Hilbert Transformed to green trace, then overlays red trace.).



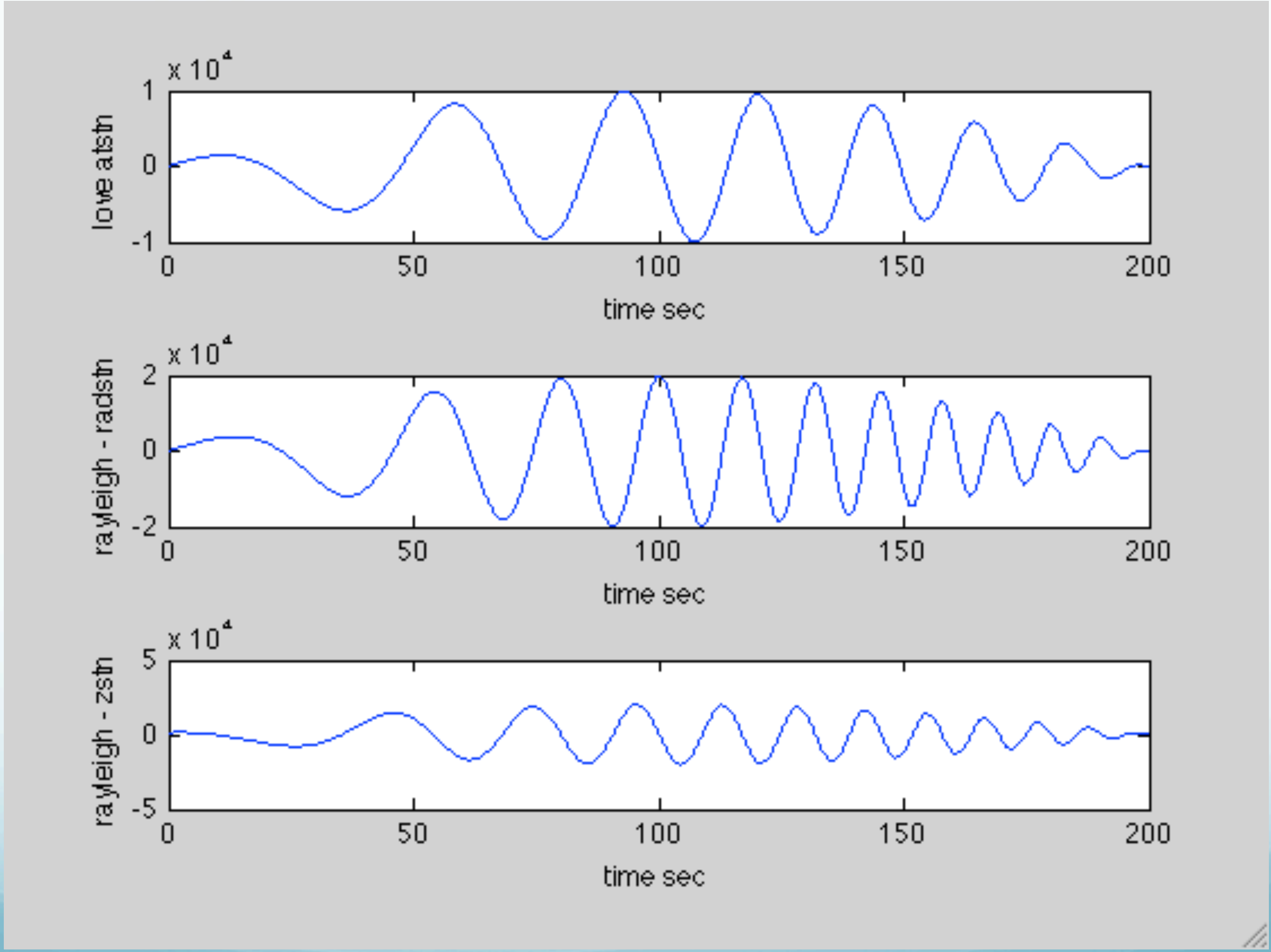
```
n=512;
a=sin(2*pi*[0:(n-1)]/n);
b=hilbert(a);
clf
plot(a)
hold
plot(-imag(b),'r')
plot(real(b),'g--')
grid
```



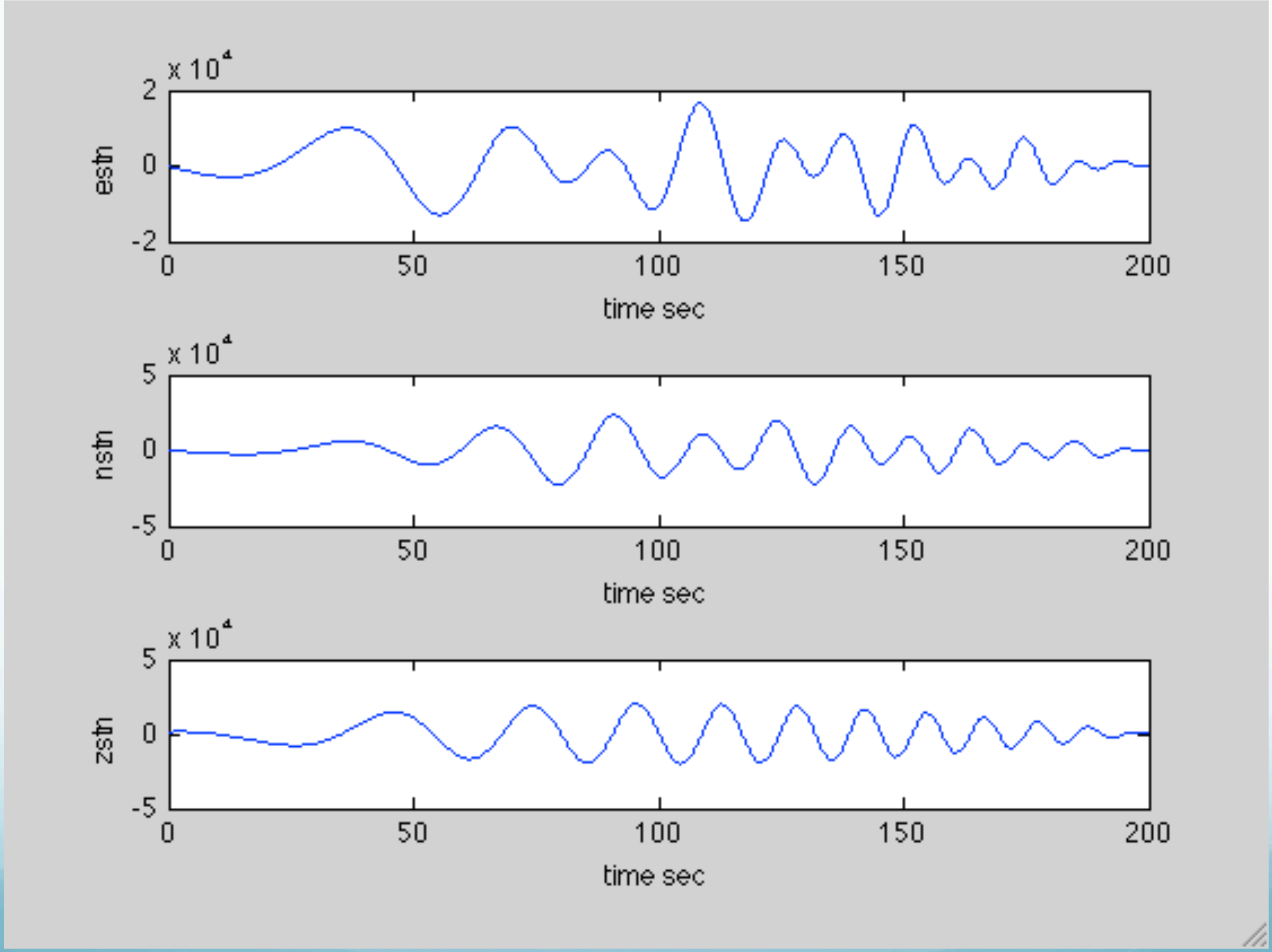
Hilbert Transform

```
if hilb ==1;          % hilbert transform the vertical component
 zh=hilbert(z);       % to make Rayleigh wave in phase (rather than
                      90° different) on vert and horz
 z=-imag(zh);         % if present (z constructed from HT, so
                      used +imag to make overlay for last
                      figure)
 else;
end;
```

Make Love and Rayleigh waves (Z, R and T)



Rotate horizontals into seismograms @ 30°.



Plot the data

```
% plot the raw data
f1=figure('name','DATA SEISMOGRAMS');
subplot(3,1,1);
plot(t,e);
xlabel('time sec');
ylabel(strcat('EW Comp at ',station));
subplot(3,1,2);
plot(t,n);
xlabel('time sec');
ylabel(strcat('NS Comp at ',station));
subplot(3,1,3);
plot(t,z);
xlabel('time sec');
ylabel(strcat('Z comp at ',station));
```

Filtering

Filtering is a two step process in Matlab

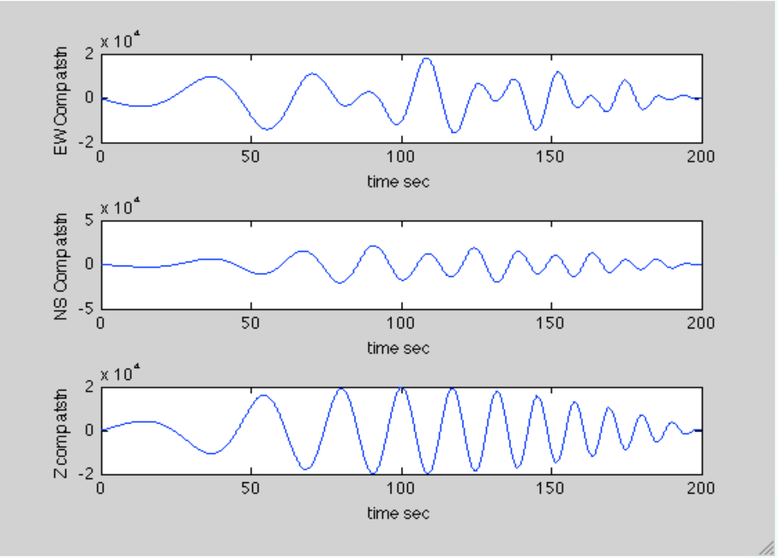
Design the filter
Apply the filter

There is a filter design GUI you can use to design the perfect filter called <u>fdatool</u>

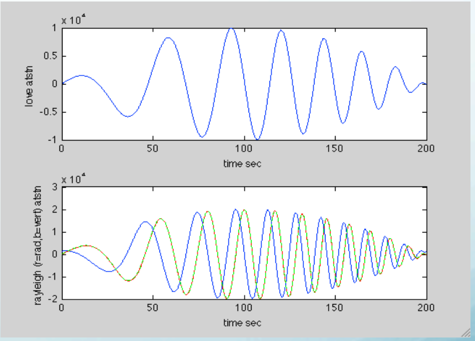
Or you can design filters using pre-built filter types (Butterworth, Bessel, etc.)

```
function [d]=bandpass(c,flp,fhi,npts,delt)
%
% [d]=bandpass(c,flp)
%
% bandpass a time series with a 2nd order butterworth filter
%
% c = input time series
% flp = lowpass corner frequency of filter
% fhi = highpass corner frequency
% npts = samples in data
% delt = sampling interval of data
%
n=2;                % 2nd order butterworth filter
fnq=1/(2*delt);        % Nyquist frequency
Wn=[flp/fnq fhi/fnq];    % non-dimensionalize the corner
frequencies
[b,a]=butter(n,Wn);      % butterworth bandpass non-dimensional
frequency
d=filtfilt(b,a,c);              % apply the filter: use zero
phase filter (p=2)
return;
```

```
Filter & plot the filtered data
% filter the data
%
if flp > 0;
   e1=bandpass(e,flp,fhi,npts,delt);
   n1=bandpass(n,flp,fhi,npts,delt);
   z1=bandpass(z,flp,fhi,npts,delt);
   e=e1;
   n=n1;
   z=z1;
   %
   % plot the filtered data
   f2=figure('name','FILTERED SEISMOGRAMS');
   subplot(3,1,1);
   ......   removed for clarity
   else;
end;
```



*The vertical channel has also had a Hilbert transform applied so that the Rayleigh wave is in phase on the NS and Z components



GOAL: Solve for polarization

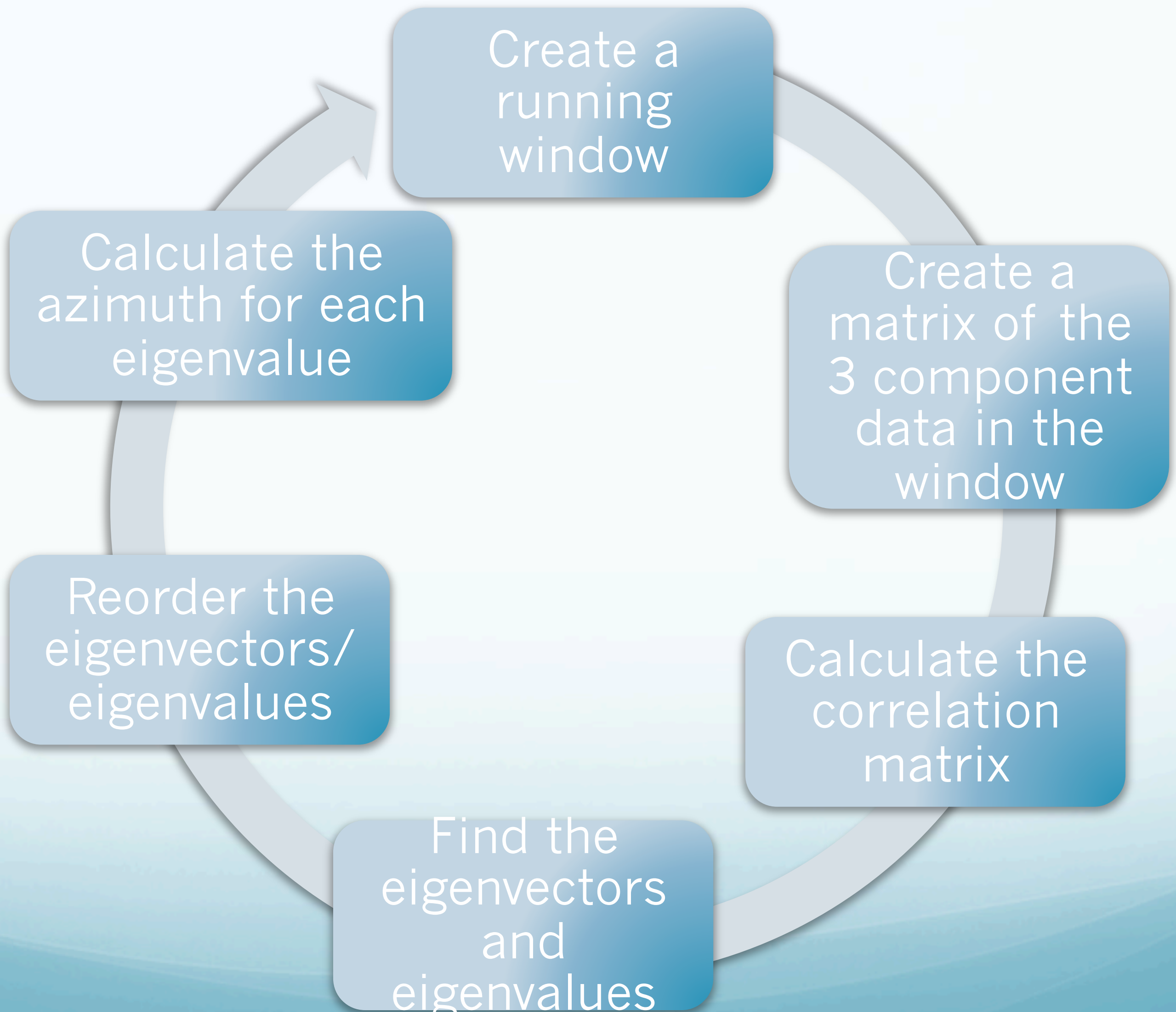
- Recognize that incoming seismic phases should represent the principal components, or the strongest signal, on the 3 component data
- The principal components, in turn, are equal to the eigenvectors of the covariance matrix of the 3 component matrix. This can be derived using PCA techniques
- Eigenvectors/values represent a spatial transformation which maximizes covariance between the 3 components, and they contain information on the azimuth from which the primary signal is derived
- **Since multiple phases may be present, we would prefer to look at short time windows of the 3 component data, or in other words, perform PCA on a running window through the continuous waveforms**

Main Code

Results

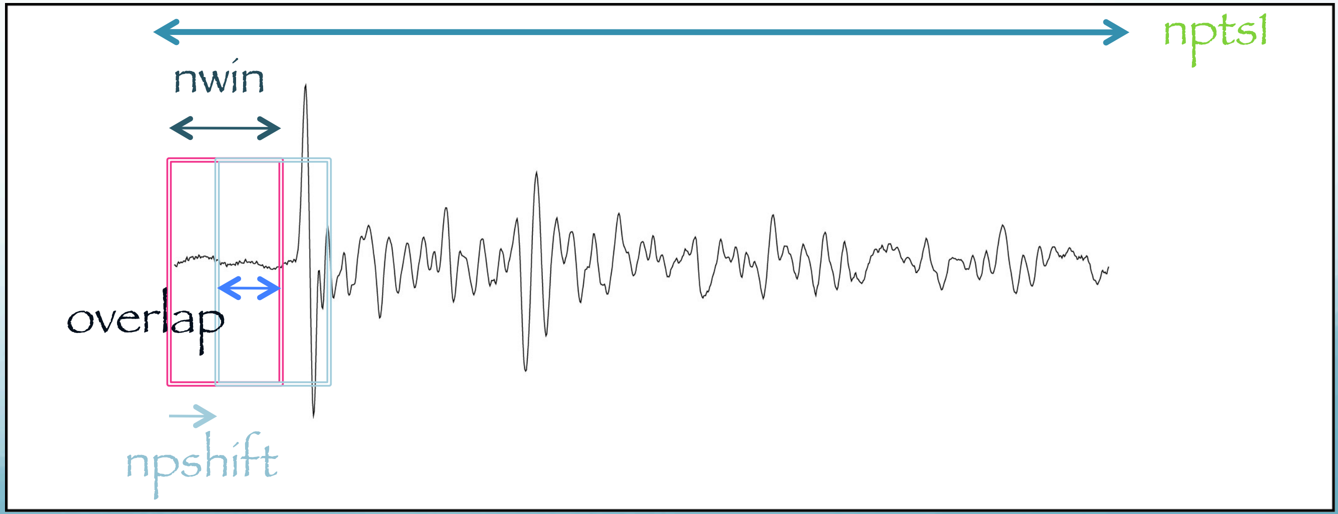
- plot azimuths of eigenvectors
- plot azimuths exceeding 50% of maximum value

Step 2: Main Code

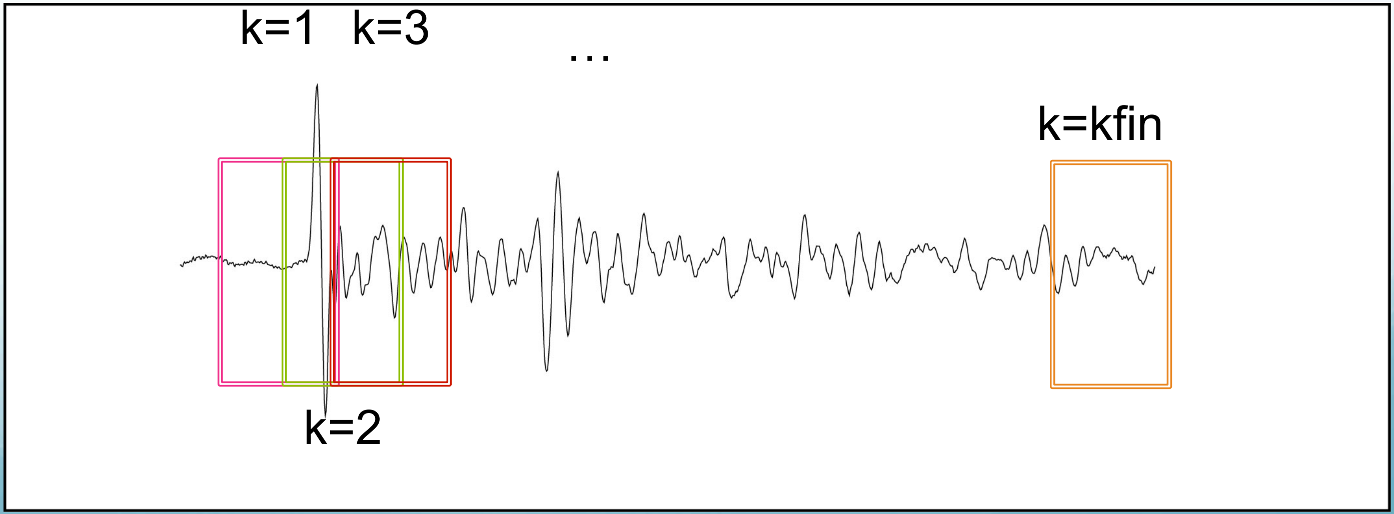


Moving window using loops

```
%   Moving window loop
%
npts1=fix(ttot/delt) + 1;    %  total number of samples to analyze
nwin=fix(twin/delt) + 1;     %  number of samples in a time window
npshift=fix(twin/(2*delt))+1;  % number of samples to shift over
kfin=fix((npts1-nwin)/(npshift+1))+1; % number of time windows
considered

mxde1=0.;
mxde2=0.;
mxde3=0.;
```



```
for k=1:kfin;
   nwinst=(k-1)*(npshift-1)+1;  % start of time window
   nwinfn=nwinst+nwin-1;    % end of time window
   ……..  missing code to be supplied later
   t2(k)=delt*(nwinst-1);    % assign time for this window to the
window start
end;
```



Eigenvalues/Eigenvectors

When a Matrix multiplies a vector in general the direction and magnitude of the vector will change.

BUT there are special vectors where only the magnitude changes (on multiplication by the Matrix). These are called **eigenvectors** The value by which the length changes is the associated **eigenvalue**

We say that x is an eigenvector of A iff

$$Ax = \lambda x$$

In other words, x is an eigenvector if when you multiply it by A it returns a multiple of itself. λ is called the associated eigenvalue.

In Matlab use [V,D] = eig(A) to get a matrix V whose columns are the eigenvectors of A and a **diagonal** matrix D whose entries on the diagonal are the corresponding eigenvalues.

```
>> A

A =

     1     2
     3     4

>> [V,D] = eig(A)

V =

   -0.8246   -0.4160
    0.5658   -0.9094
```

```
D =

   -0.3723         0
         0    5.3723
```

Missing code from inside our loop

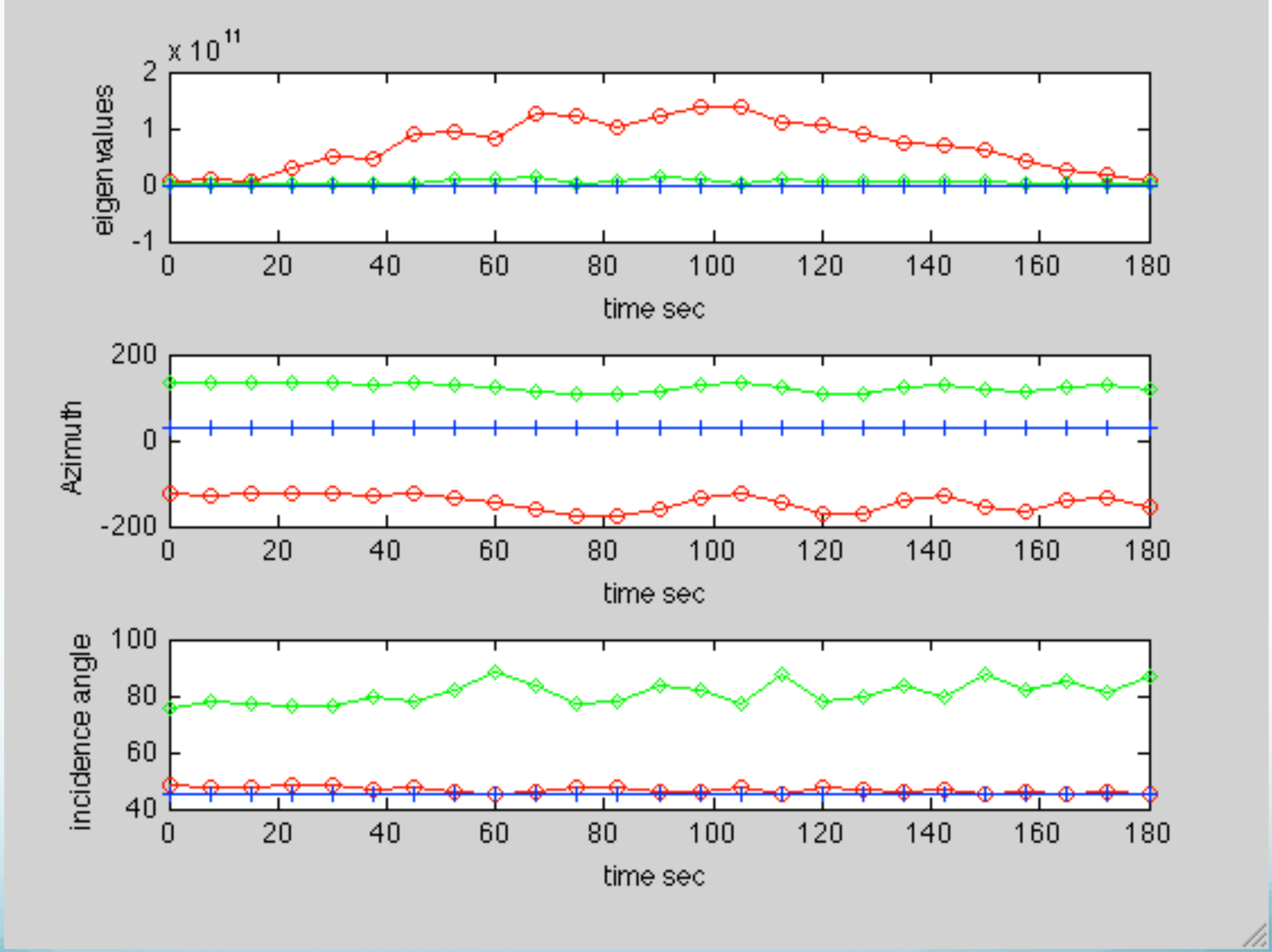
```
a=csigm(e,n,z,nwinst,nwinfn);     % signal matrix
c=a'*a;                           % covariance matrix
[v1,d1]=eig(c);                   % eigenvalue/eigenvectors
[v,d]=order(v1,d1);               % put eigenvalues & eigenvectors
                    in ascending order

% azimuth for each of the 3 eigenvalues
ang1(k)=atan2(v(1,1),v(2,1)) * 180/pi;
ang2(k)=atan2(v(1,2),v(2,2)) * 180/pi;
ang3(k)=atan2(v(1,3),v(2,3)) * 180/pi;

% incidence angle of the 3 eigenvalues
vang1(k)=acos(abs(v(3,1)))* 180/pi; %angle from the vertical
vang2(k)=acos(abs(v(3,2)))* 180/pi;
vang3(k)=acos(abs(v(3,3)))* 180/pi;
```

Still in loop

```
de1(k)=d(1);
de2(k)=d(2);
de3(k)=d(3);

mxde1=max(mxde1,de1(k));  % find the maximum values
mxde2=max(mxde2,de2(k));
mxde3=max(mxde3,de3(k));
```

Outside of Loop again

```
f3=figure('name','Eigenvalues and Inferred Azimuth');
subplot(3,1,1);
plot(t2,de1,'-or',t2,de2,'-dg',t2,de3,'-+b');
xlabel('time sec');
ylabel('eigenvalues');

subplot(3,1,2);
plot(t2,ang1,'-or',t2,ang2,'-dg',t2,ang3,'-+b');
xlabel('time sec');
ylabel('Azimuth ');

subplot(3,1,3);
plot(t2,vang1,'-or',t2,vang2,'-dg',t2,vang3,'-+b');
xlabel('time sec');
ylabel('incidence angle ');
```



x 10^11
eigen values
time sec
Azimuth
time sec
incidence angle
time sec

GOAL: Solve for polarization

Results

- plot azimuths of eigenvectors
- plot azimuths exceeding 50% of maximum value

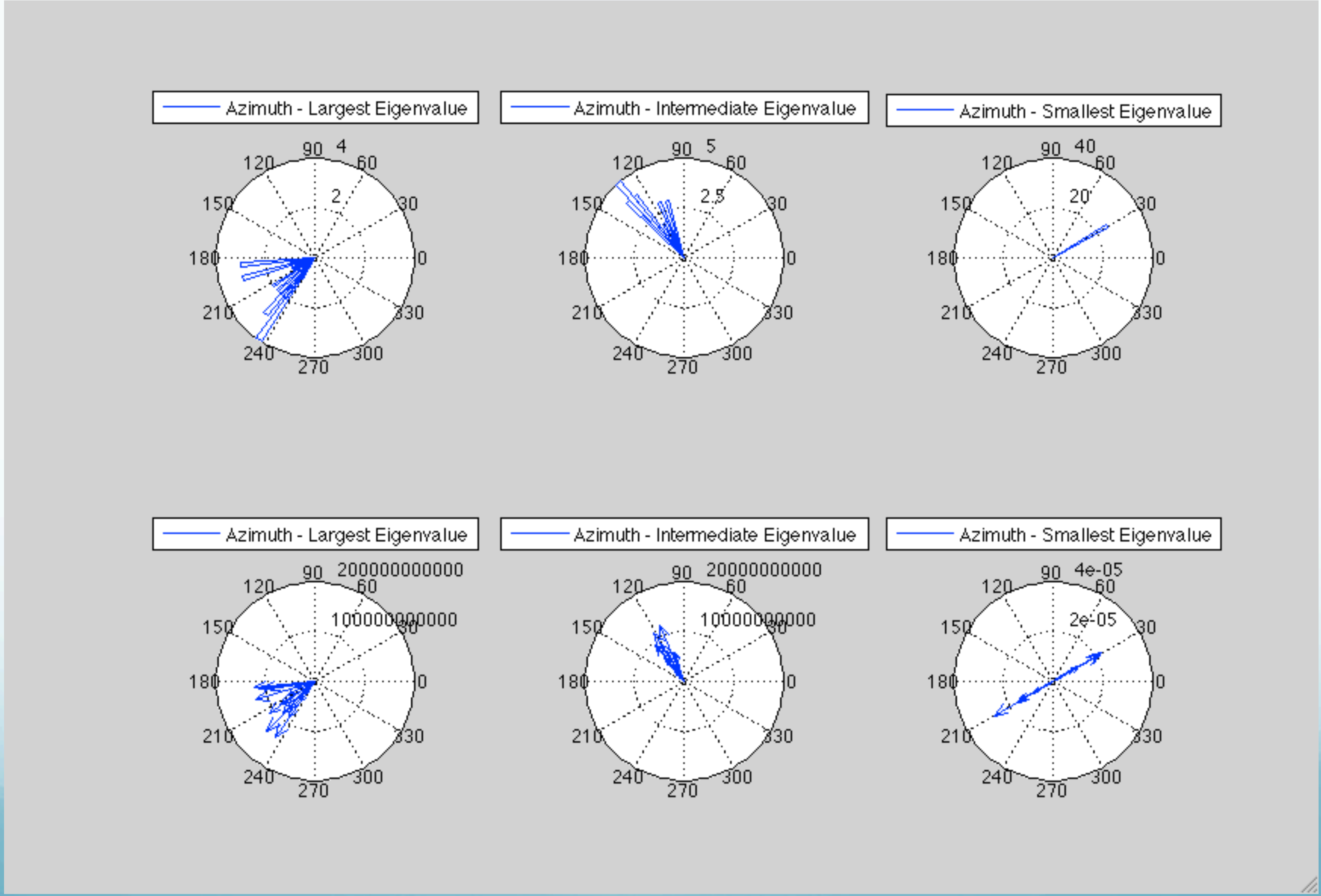
Main Code

-

Rose Diagrams

```
%  Rose plots
f4=figure('name','Azimuth Distribution');
subplot(2,3,1);
title('Azimuth - Largest Eigenvalue');
rose(ang1*pi/180,100);

subplot(2,3,2);
title('Azimuth - Intermediate Eigenvalue');
rose(ang2*pi/180,100);

subplot(2,3,3);
title('Azimuth - Smallest Eigenvalue');
rose(ang3*pi/180,100);
```

```
nskip=1;
if nskip == 1;
   else;
neig1=0;
neig2=0;
neig3=0;
for k=1:kfin;
   if de1(k) >= 0.5*mxde1;
      neig1=neig1+1;
      angm1(neig1)=ang1(k);
   else;
   end;
   if de2(k) >= 0.5*mxde2;
      neig2=neig2+1;
      angm2(neig2)=ang2(k);
   else;
   end;
   if de3(k) >= 0.5*mxde3;
      neig3=neig3+1;
      angm3(neig3)=ang3(k);
   else;
   end;
end;
subplot(2,3,4);
title('Azimuth - Largest
   Eigenvalue,50% Threshold');
rose(angm1*pi/180,100);
subplot(2,3,5);
title('Azimuth - Intermediate
   Eigenvalue,50% Threshold');
rose(angm2*pi/180,100);
subplot(2,3,6);
title('Azimuth - Smallest
   Eigenvalue,50% Threshold');
rose(angm3*pi/180,100);
end;
```


Azimuth - Largest Eigenvalue
Azimuth - Intermediate Eigenvalue
Azimuth - Smallest Eigenvalue
Azimuth - Largest Eigenvalue
Azimuth - Intermediate Eigenvalue
Azimuth - Smallest Eigenvalue

Intro writing GUI's

MATLAB

What is a GUI?

Graphical User Interface

(Aside - what is “wysiwyg”?)

MatLab provides a tool called the

Graphical User Interface Development Environment

(GUIDE)

A GUI used to create GUI's.

You can also be a masochist and write the code from scratch.

A GUI should be consistent and easily understood.

(if you need the manual, there's a bug in the program or a flaw in the gui. Non-UNIX philosophy!)

Provide the user with the ability to use a program without having to worry about commands to run the actual program.

Possible components of a GUI -

Pushbuttons	Menus
Sliders	Interactive Graphics
List boxes	....etc

3 Essential Parts of a GUI –

1

Graphical Components
pushbuttons, edit boxes, sliders, labels, menus, etc…

Static Components
Frames, text strings,…

Both created using the MATLAB function uicontrol.

3 Essential Parts –

2

Figures – components are contained in figures.

3

Callbacks – The functions which perform the required action when a component is “pushed”.

GUIDE Properties

Allows the user to drag and drop components that he/she wants in the "layout" area of the GUI.

All "guide" GUI's start with an opening function.

Callback is performed before user has access to GUI.

GUIDE stores GUIs in two files, which are generated the first time you save or run the GUI:

– .fig file - contains a complete description of the GUI figure layout and the components of the GUI.
Changes to this file are made in the Layout Editor

– .m file - contains the code that controls the GUI.
You program callbacks in this file using the M-file Editor.

Creating a GUI

Typical stages of creating a GUI are:

1. Designing the GUI

2. Laying out the GUI
Using the Layout Editor

3. Programming the GUI
Writing callbacks in the M-file Editor

4. Saving and Running the GUI

Assessing the Value of Your GUI

Ask yourself two basic questions when designing your GUI.

- Do the users always know where they are?
- Do they always know where to go next?

Constantly answering these two questions will help you keep in perspective the goal of your GUI.

Callback function

The “meat” of the GUI process.

Opening function is first callback in every “guide” generated GUI.

Usually used to generate data used in GUI.

Callbacks define what will happen when a figure component is selected.

You must write the callback code!!!!

Summary

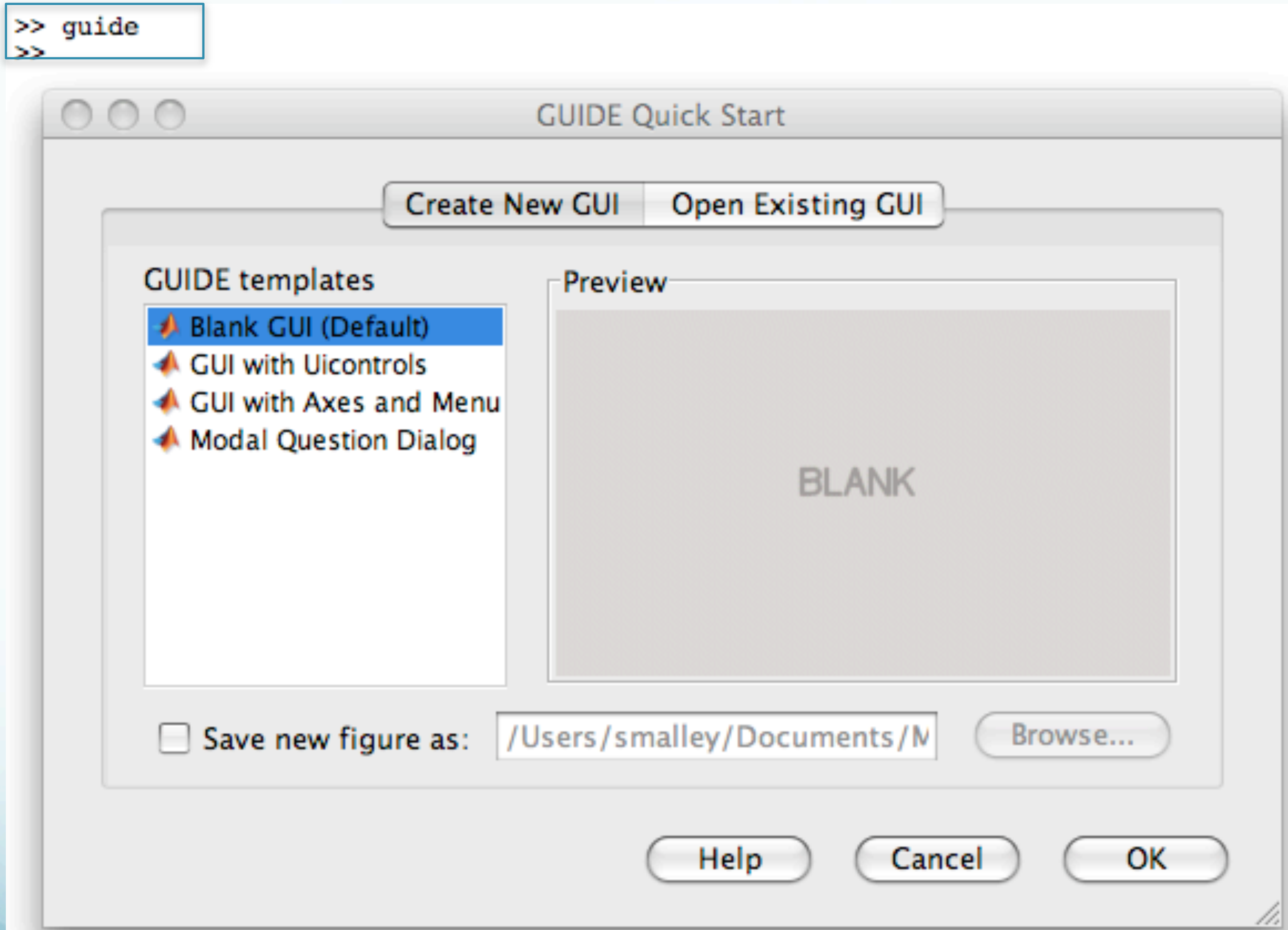
At command prompt type "guide".

Lay out your GUI in the layout editor.

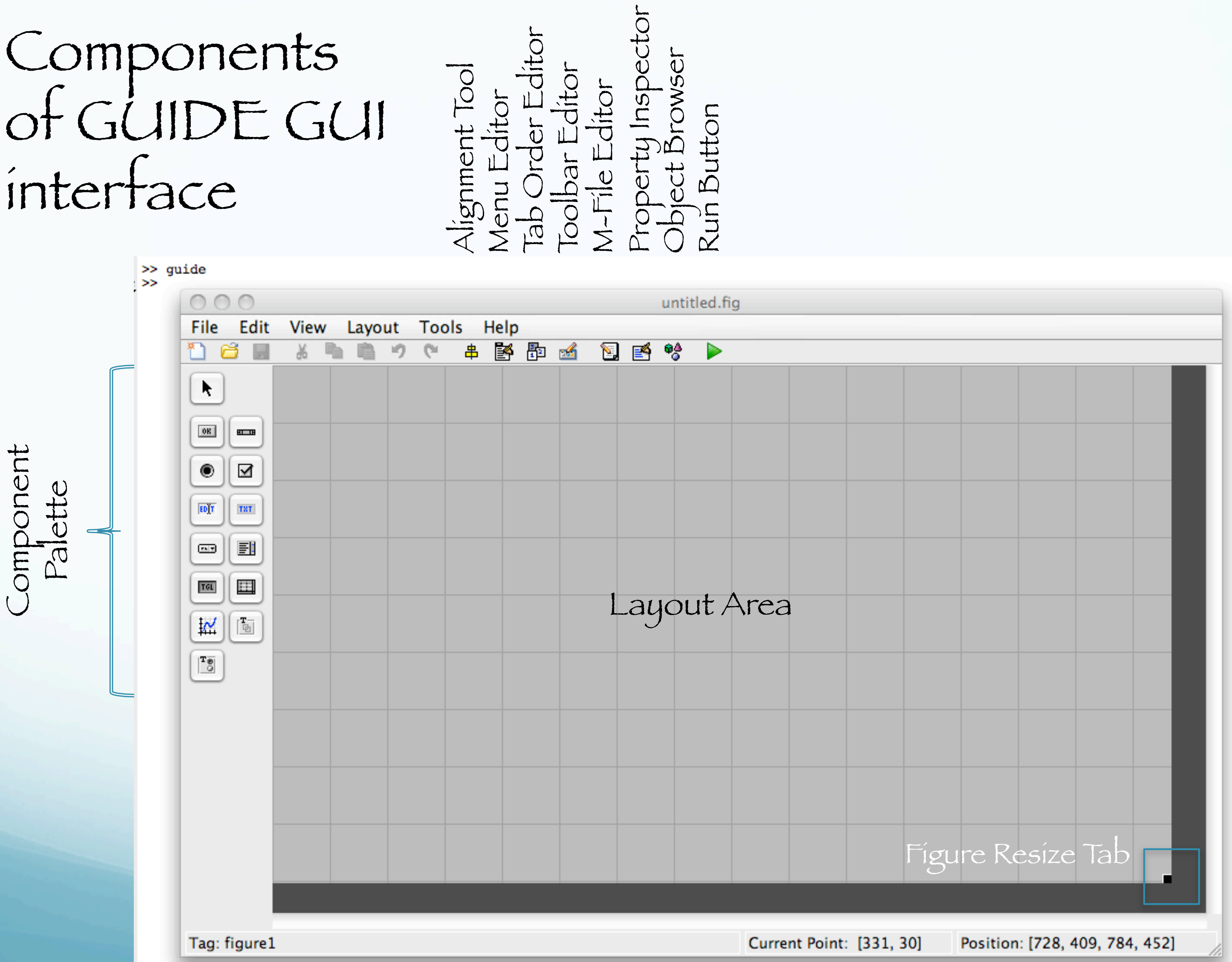
Define data in Opening Function.

Edit/Align your components using
- Tools Menu
- Align
- View menu
- Property Inspector

Write the Callbacks
(This is the most difficult aspect when creating GUI's)


>> guide
>>
GUIDE Quick Start
Create New GUI
Open Existing GUI
GUIDE templates
Blank GUI (Default)
GUI with Uicontrols
GUI with Axes and Menu
Modal Question Dialog
Preview
BLANK
Save new figure as:
/Users/smalley/Documents/M
Browse...
Help
Cancel
OK

Components of GUIDE GUI interface



Writing Callbacks (the hard part).

A callback is a sequence of commands (function) that are execute when a graphics object is activated.

Callbacks are stored in the GUI's **m**-file.

Callbacks are a property of a graphics object (e.g. CreateFnc, ButtonDwnFnc, Callback, DeleteFnc).

(Also called an "event handler" in some programming languages.)

A callback is usually made of the following stages:

1. Get handle of object initiating the action
(the object provides event / information / values).

2. Get handles of objects being affected
(the object thatwhose properties are to be changed).

3. Getting necessary information / values.

4. Doing some calculations and processing.

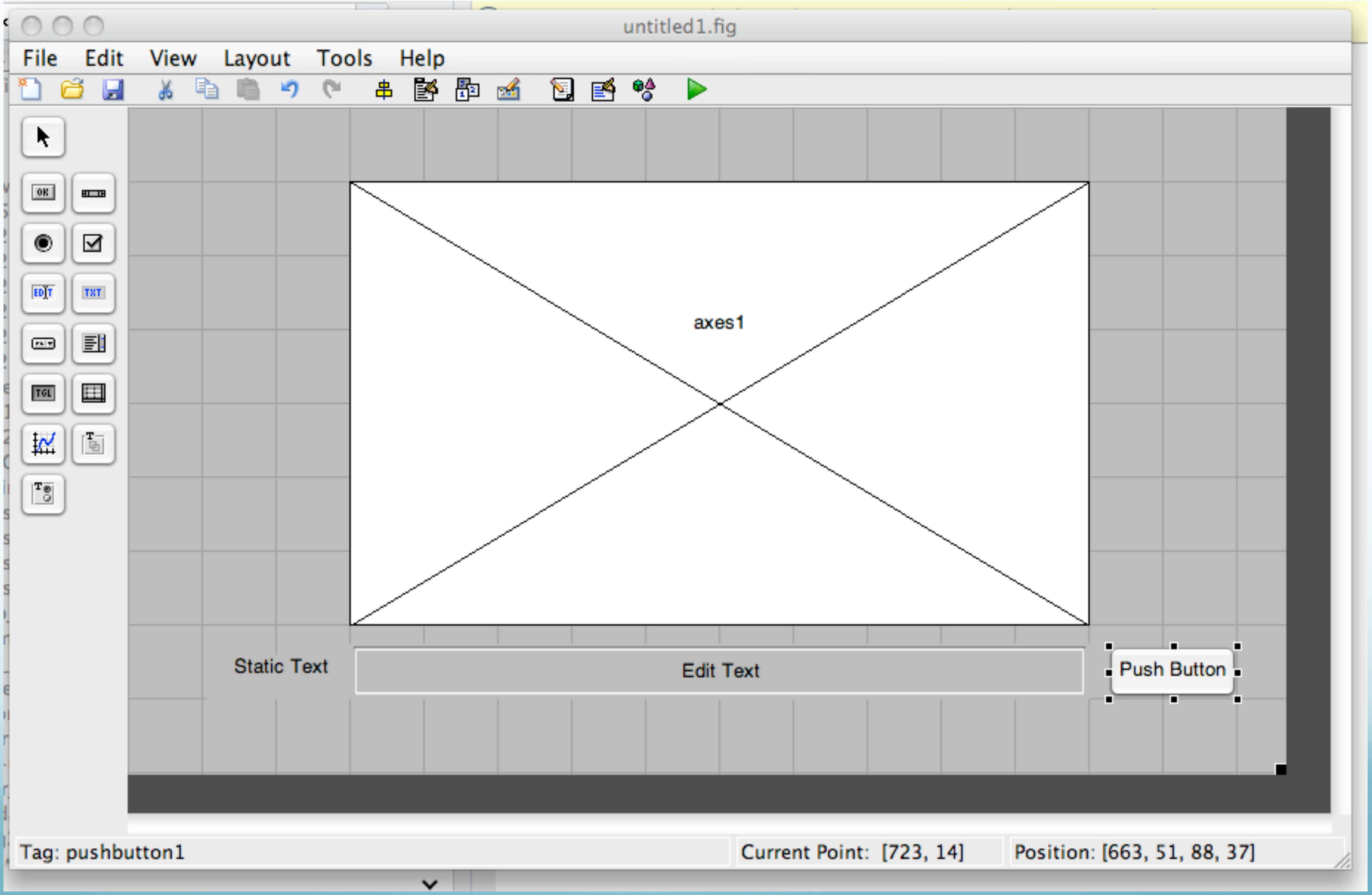
5. Setting relevant object properties to effect action.

Let's create a GUI that plots a function that we can interactively specify.

We first lay out the basic controls for our program, selected from the menu along the left side:

axes,
static text,
edit box,
and a button.

Define and place the axis, static text (will have the prompt for the function), edit text (to interactively enter the function), and a button to do the plot.



Basic Elements of our GUI-

axes: a place to draw.

static text: text that is stuck/fixed/static on the screen, the user can't edit it.

edit box: a white box that the user can type input into.

button: performs an action when user clicks on it.

The Property Inspector

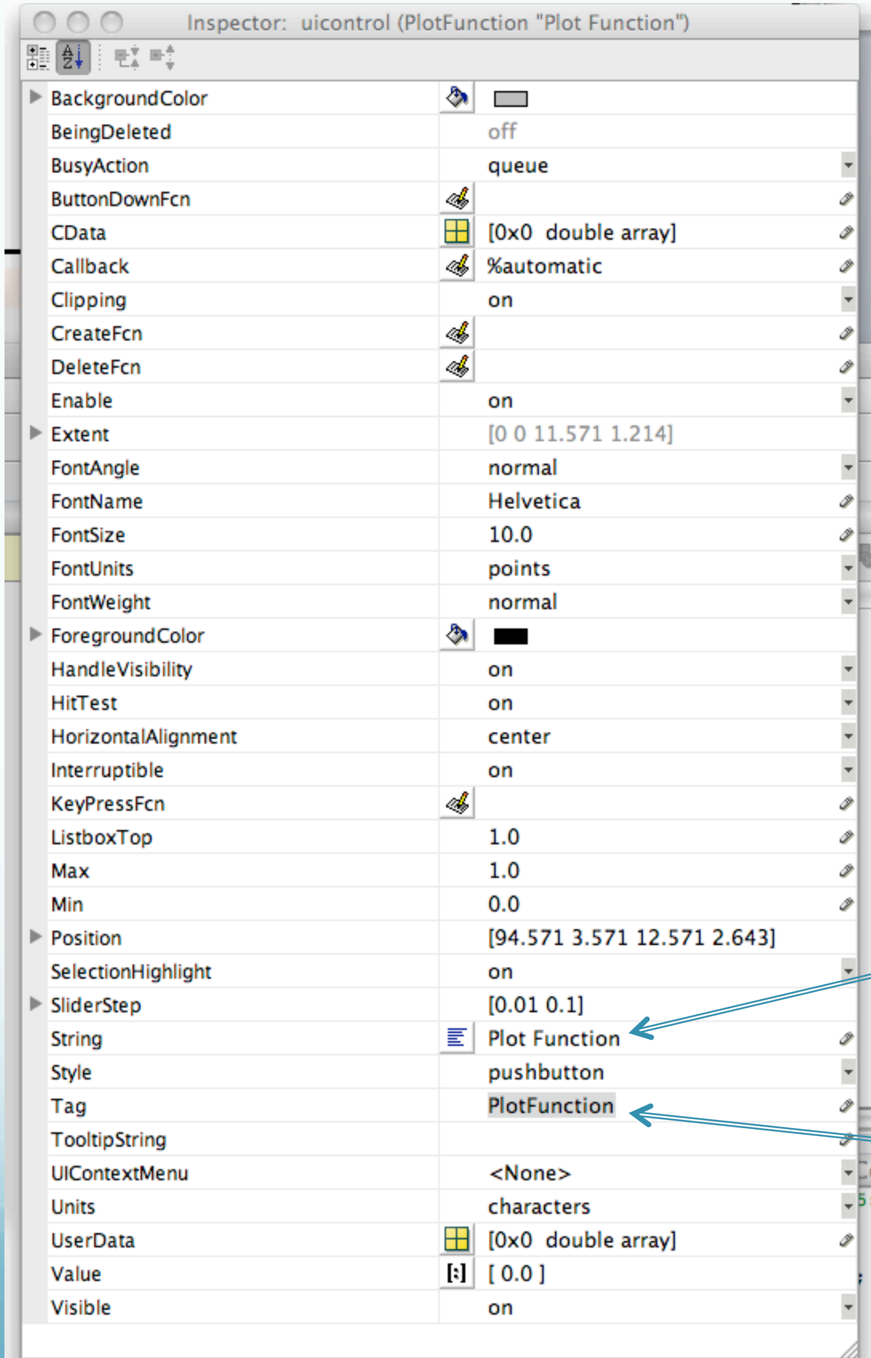
When you double-click on a control, it brings up a window listing all the properties of that control (font, position, size, etc.)

Tag - the name of the control in the code. best to rename it to something identifiable ("PlotButton" vs "button1")

String - the text that appears on the control

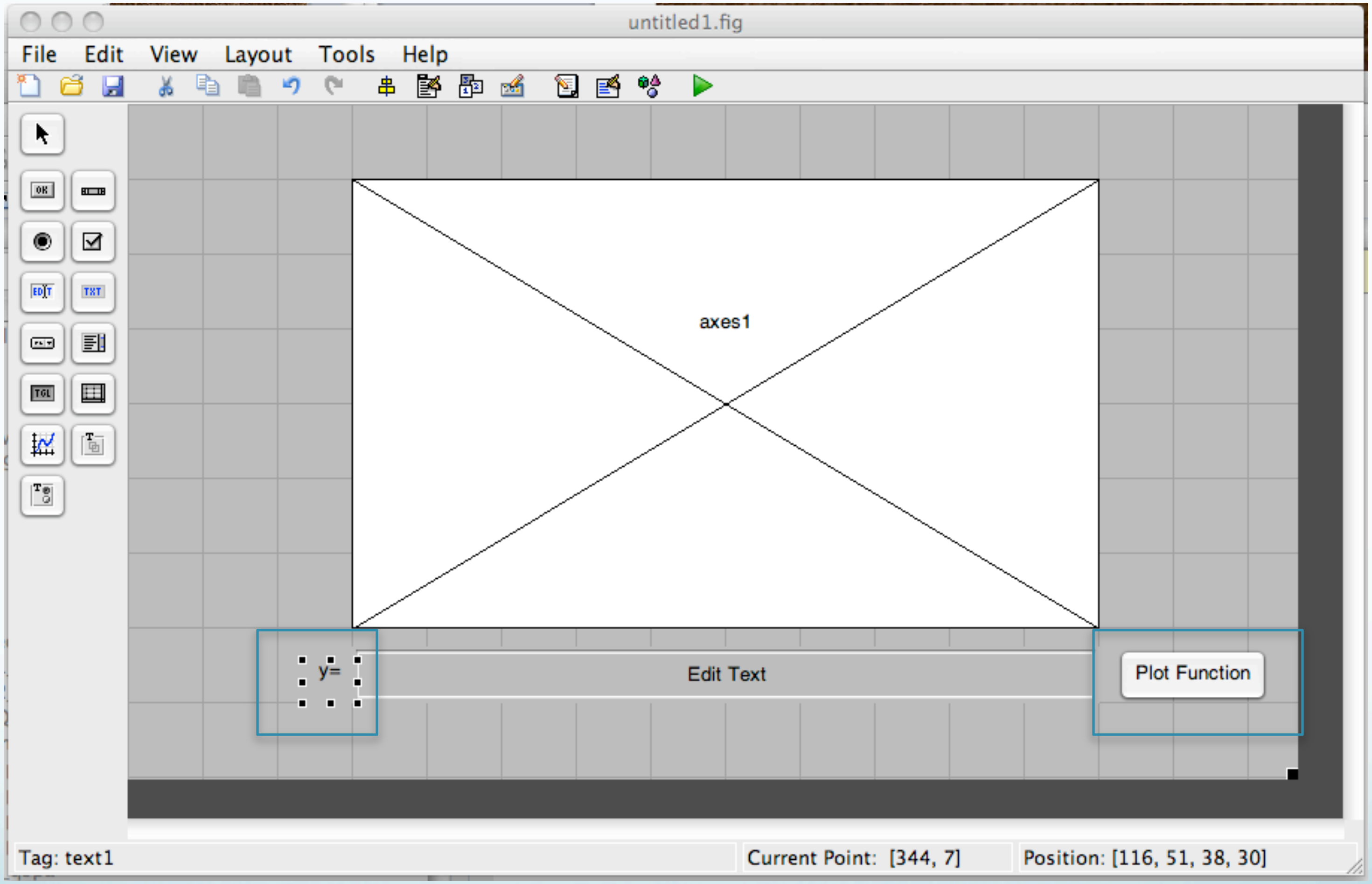
ForegroundColor - color of the text

BackgroundColor - color of the control



Enter text string for pushbutton

Enter tag for pushbutton


untitled1.fig
File
Edit
View
Layout
Tools
Help
axes1
y=
Edit Text
Plot Function
Tag: text1
Current Point: [344, 7]
Position: [116, 51, 38, 30]

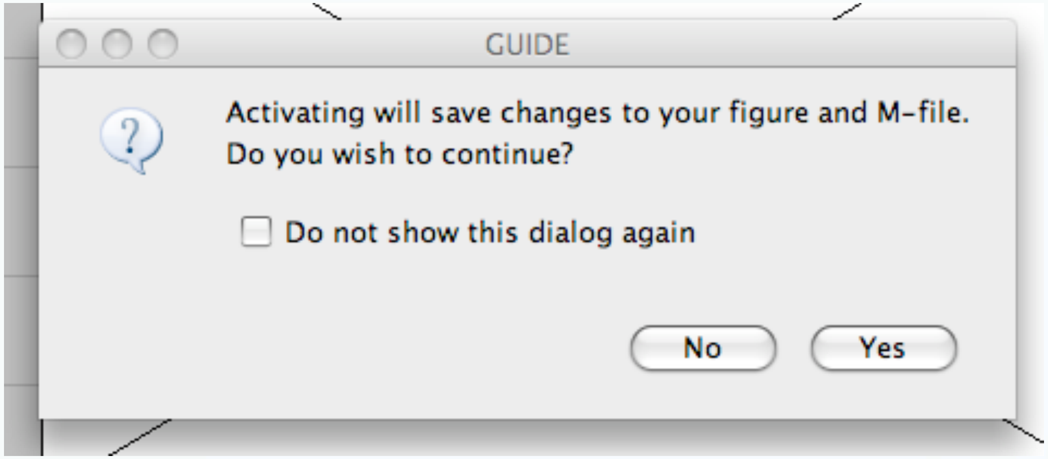
Running

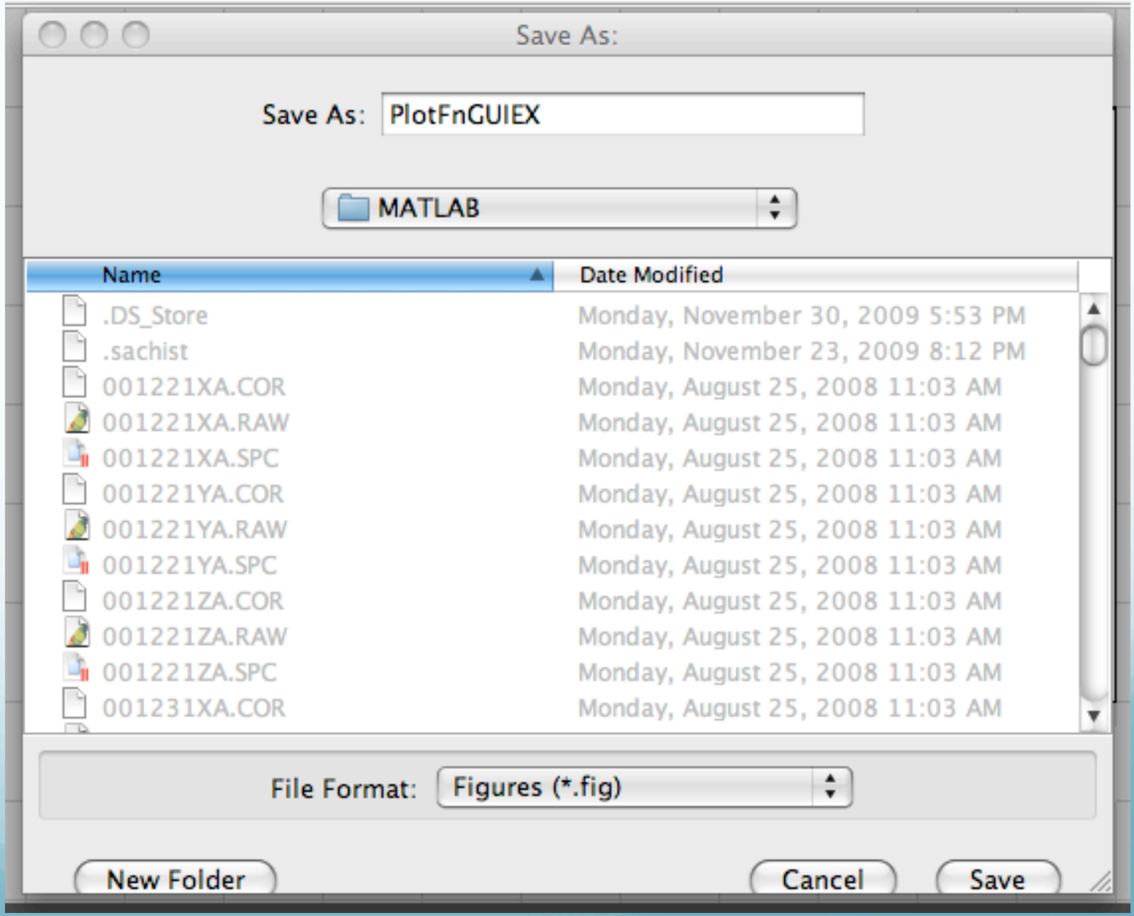
If you press the green arrow at the top of the GUI editor, it will save your current version and run the program.

The first time you run it, it will ask you to name the program.

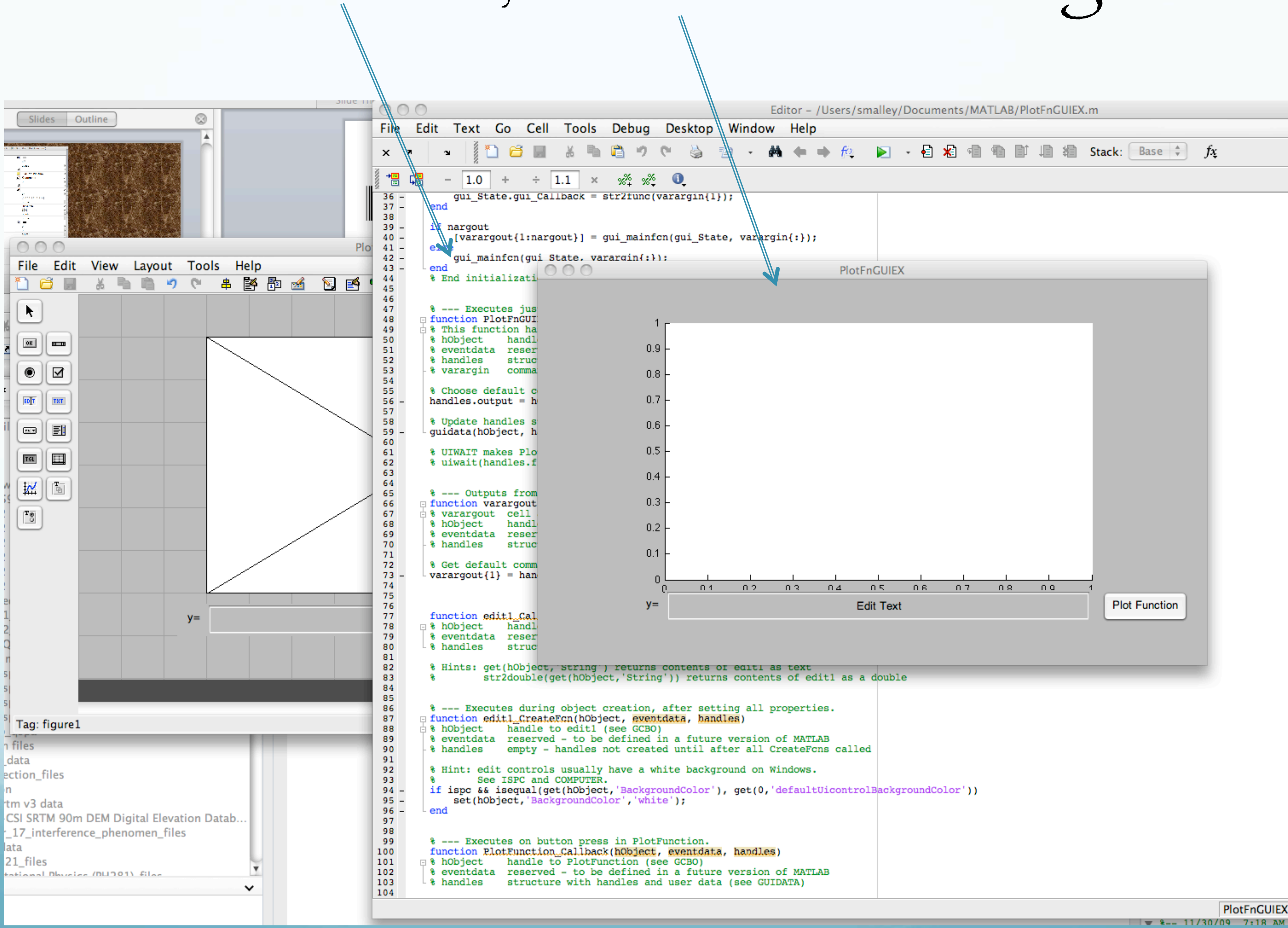
Our figure looks about right, but it doesn't do anything yet.

We have to define a callback for the button so it will plot the function when we press it.


GUIDE
Activating will save changes to your figure and M-file.
Do you wish to continue?
Do not show this dialog again
No
Yes

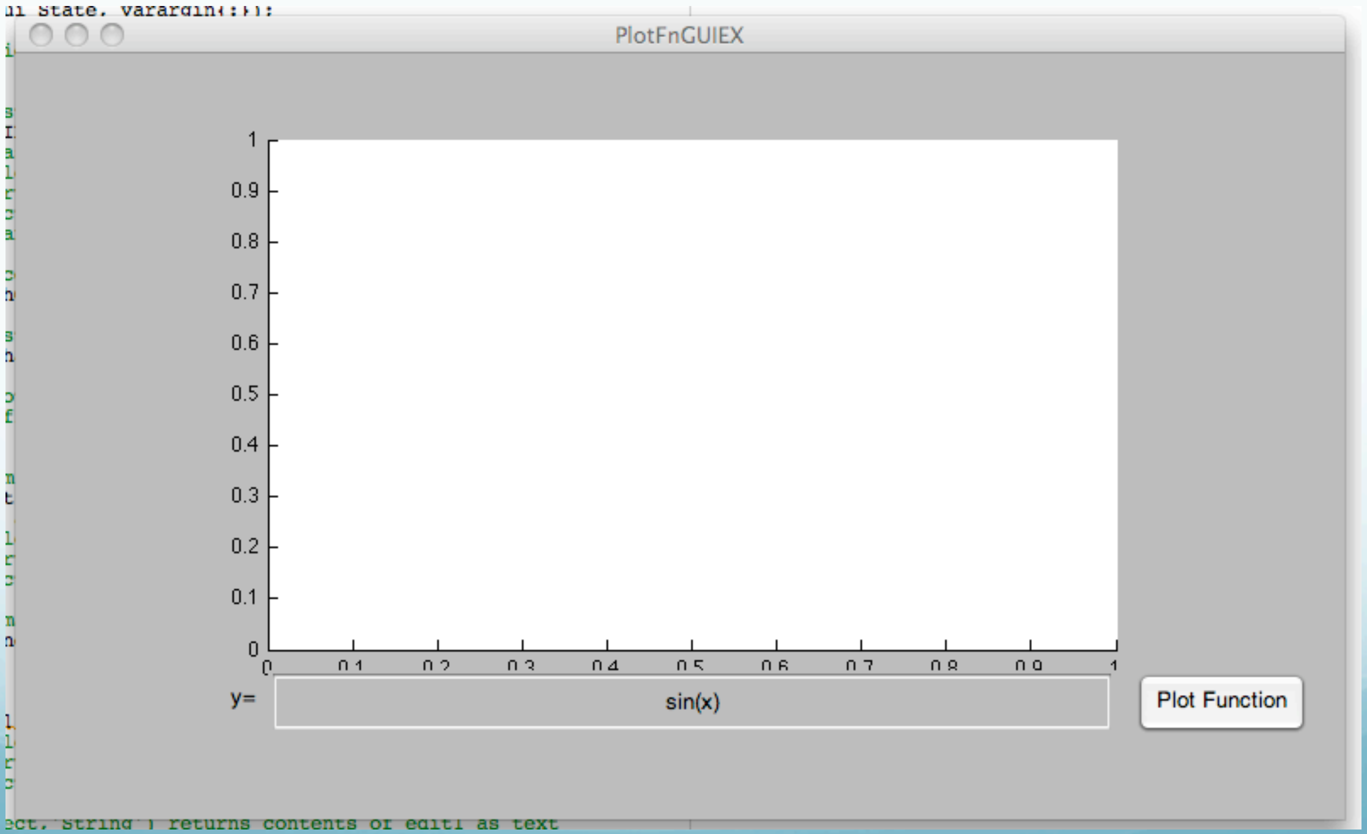

Save As:
Save As: PlotFnGUIEX
MATLAB
Name
Date Modified
.DS_Store Monday, November 30, 2009 5:53 PM
.sachist Monday, November 23, 2009 8:12 PM
001221XA.COR Monday, August 25, 2008 11:03 AM
001221XA.RAW Monday, August 25, 2008 11:03 AM
001221XA.SPC Monday, August 25, 2008 11:03 AM
001221YA.COR Monday, August 25, 2008 11:03 AM
001221YA.RAW Monday, August 25, 2008 11:03 AM
001221YA.SPC Monday, August 25, 2008 11:03 AM
001221ZA.COR Monday, August 25, 2008 11:03 AM
001221ZA.RAW Monday, August 25, 2008 11:03 AM
001221ZA.SPC Monday, August 25, 2008 11:03 AM
001231XA.COR Monday, August 25, 2008 11:03 AM
File Format: Figures (*.fig)
New Folder
Cancel
Save

Pile of windows – GUIDE design window, m file with code for GUI, window with running GUI.



Buttons "work" (respond when click in them), can enter text. But nothing happens.

Have to write callback routine to specify what happens.



Writing Callbacks

As noted, when you run the program, it creates two files.

your_gui.fig -- contains the layout of your controls

your_gui.m -- contains code that defines a callback function for each of your controls

We generally don't mess with the initialization code in the m-file.

We will probably leave many of the control callbacks blank.

Writing Callbacks

In our example, we just need to locate the function for the button.

This is why it is important to have a good Tag so we can keep our controls straight.

You can also right-click on the control and select View Callback.

Writing Callbacks

Initially the button callback looks like this.

```
% --- Executes on button press in PlotFunction.
function PlotFunction_Callback(hObject, eventdata, handles)
% hObject    handle to PlotFunction (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

We can delete the comments and type code.

Note every function has the parameter handles.

This contains all the controls:

`handles.PlotButton, handles.edit1, handles.axes1, …`

We can add variables to handles to make them available to all functions:

```
handles.x = 42;
```

Writing Callbacks

We can look up any property of a control with the get function.

Similarly, we can change any property with the set function.

This is where things get complicated.

Writing Callbacks

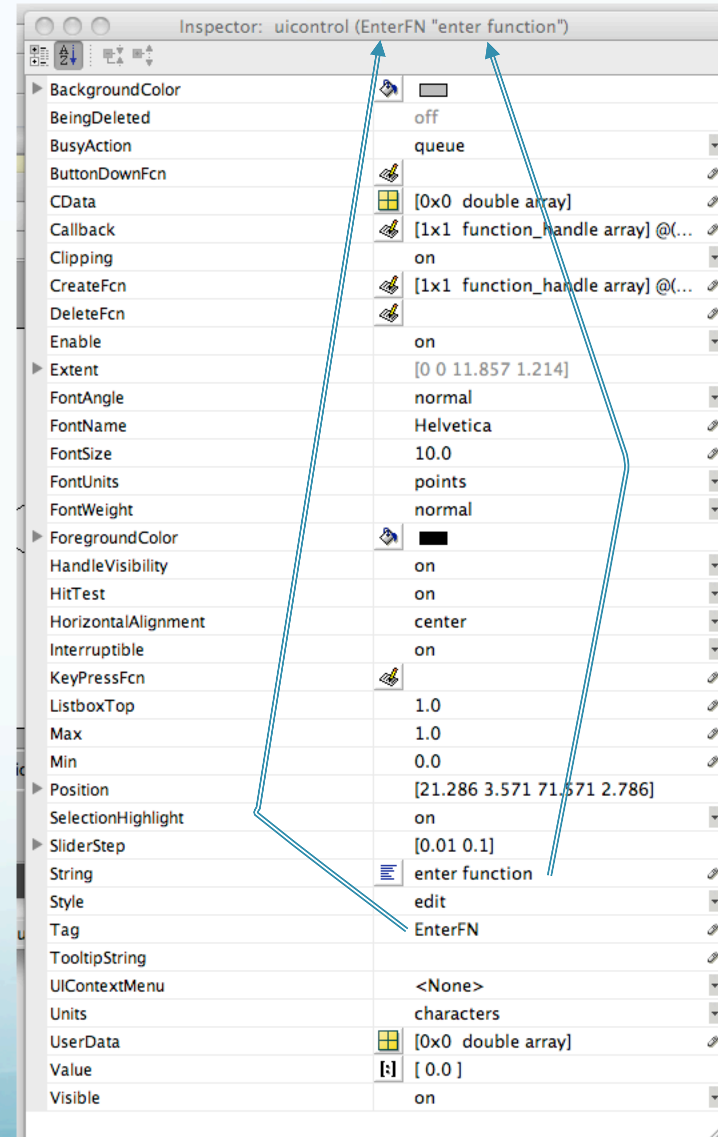
We need two callbacks.

1) We want to get the String typed into the edit box

2) and plot it.

```
function EnterFN_Callback(hObject, eventdata, handles)
. . .
function EnterFN_CreateFcn(hObject, eventdata, handles)
```

Look at properties inspector and **m** file to see how things match up.



1) We want to get the string typed into the edit box

Blue produced by guide, have to add the black (one line). Variable <u>**handles.EnterFn**</u> created here.

```
function EnterFN_Callback(hObject, eventdata, handles)
% hObject    handle to EnterFN (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of EnterFN as
text
%        str2double(get(hObject,'String')) returns contents of
EnterFN as a double
handles.EnterFn=get(hObject,'String');
```

2) and plot it.

Blue produced by guide, have to add the stuff in black (a couple of lines). Variable **`handles.EnterFn`** created by us, while **`handles.axes1`** created by guide.

```
% --- Executes on button press in PlotFunction.
function PlotFunction_Callback(hObject, eventdata, handles)
% hObject    handle to PlotFunction (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)
x=-10:.01:10
s = get(handles.EnterFN, 'String');
y = eval(s); %eval just evaluates the given string
handles.axes1; %Subsequent commands draw on axes1.
plot(x, y);
grid;
```

Final result.