


# Matlab-II: Computing, Programming and Data Analysis

Division of Statistics and Scientific Computation  
College of Natural Sciences

Instructor: Yla Tausczik  
[tausczik@mail.utexas.edu](mailto:tausczik@mail.utexas.edu)

# Division of Statistics and Scientific Computing (SSC)



WHAT STARTS HERE CHANGES THE WORLD  
THE UNIVERSITY OF TEXAS AT AUSTIN

Search the website...

Division of Statistics + Scientific Computation


COLLEGE OF Natural Sciences

[HOME](#)
[CONSULTING](#)
[COURSES](#)
[FOR UNDERGRADUATE STUDENTS](#)
[FOR GRADUATE STUDENTS](#)
[PEOPLE](#)
[PROGRAMS](#)
[SOFTWARE](#)
[LOGIN](#)

The division opened its doors two years ago and is designed to provide campus-wide services for students, faculty, and staff and to serve as a gathering point for like-minded colleagues. Faculty from colleges and research units across campus -- including Business, Communication, Education, Engineering, Graduate Studies, Liberal Arts, Natural Sciences, Nursing, Pharmacy, Public Affairs, Social Work and the Texas Advanced Computing Center - have joined in this initiative.

Please explore our website further to learn about the many services offered through the division. The division coordinates a suite of **undergraduate and graduate courses** designed to address the needs of students in different disciplines. We also provide **consulting services** for UT students, faculty and staff, bring prominent faculty to campus as part of the **Distinguished Lecture Series**, and organize **Summer Statistics Institute** each year in May

Our hope is that you find the division welcoming and our services useful. We invite you to explore the opportunities available through the division so that you too can take advantage of the strengths in statistics and scientific computation on our campus. It is a worthy enterprise for both this institution and the world in which we live.



## NEWS & EVENTS

[FOLLOW US ON Twitter](#)

**UT Undergraduates Conduct Summer Research in Statistics**

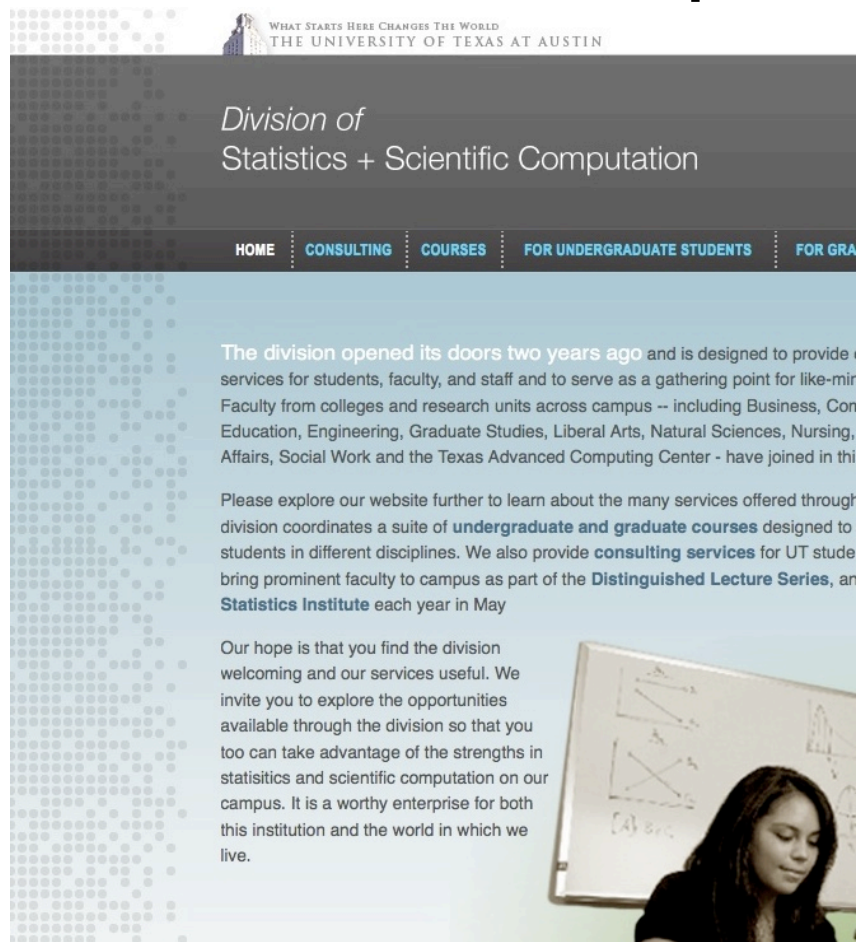
**SSC Co-Sponsoring "Conference on Numerical Linear Algebra: Perturbation, Performance, and Portability," July 19-20**

**Software Short Courses - Summer Schedule Posted**

**New Portfolio Program in Scientific Computation for Graduate Students**

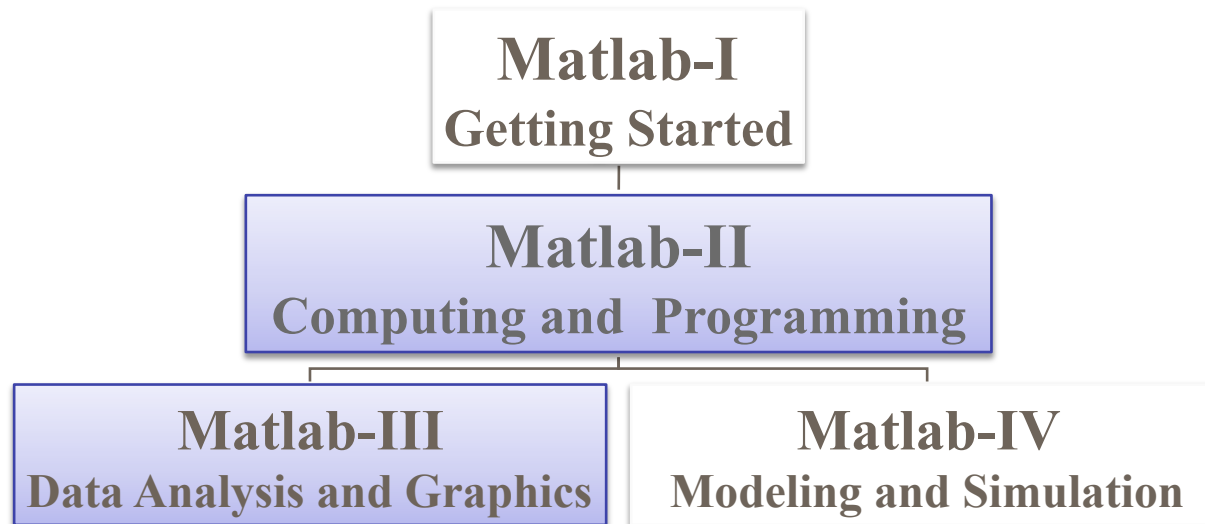
**Summer and Fall Courses Posted**

# Division of Statistics and Scientific Computing (SSC)



- Classes (under & grad)
- Portfolio Programs (under & grad)
- Free Consulting
- Software tutorials

# Matlab Short Course Structure



# Matlab II Outline

## Part I

- Computing
- Data Analysis

10 min break

## Part II

- Graphing
- Programming

Q & A

# Matlab I: Basic Concepts

- Everything is a matrix
- Numerical not symbolic calculations
- Interpreter vs M-files
- Basic Functions

# Tutorial Files Matlab II

<http://ssc.utexas.edu/software/software-tutorials>

- Under Matlab II, download files
- Under Matlab III, download files

# Matlab Computing & Data Analysis

- Constants and functions
- Operational manipulations
- Descriptive Statistics
- Mathematical & Statistical Techniques



# Matlab Constants and Functions

## ■ Built-in Constants

- *Machine independent*
  - *inf*, *NaN*, *i*, *j*, *pi*
  - *ans* (most recent unassigned input)
- *Machine or seed dependent*
  - *eps*, *realmin*, *realmax*
  - *rand*, *randn* (overwritten with each call)

# Matlab Constants and Functions

- Extended **built-in** Mathematical Functions  
Also many **distributed** with Matlab as m-files
  - Trigonometric inverses: **a** prefix → arc, **h** suffix → hyperbolic
    - *asin*, *acos*, *atan*, *acsc*, *asec*, *acot*
    - *asinh*, *acosh*, *atanh*, *acsch*, *asech*, *acoth*
  - Specialized
    - *airy*, *beta*, *legendre*
    - various bessel function types, etc.
  - Miscellaneous
    - *sqrt*, *factorial*, *gamma*, *erf*, *exp*, *log*, *log2*, *log10*
  - **Built-in** : m-file has description but no source code
  - **Distributed**: m-file has annotated source code

# Matlab Constants and Functions

- More specialized math functions

- *Help > Function Browser*
- Function tool **fx** on left bar of Command Window  
**Only available starting with Matlab R2008b**
- *Help > Product Help > Symbolic Math Toolbox  
> Function Reference > Specialized Math*  
**dirac, heaviside, ...**  
**Symbolic Math Toolbox needs to be installed**

# Matlab Computation

- Finding other functions and procedures

```
>> help <name>
```

% information displayed in command window

```
>> doc <name>
```

% information from *Matlab Help* displayed

```
>> lookfor <name>;
```

% finds functions with <name> in description header

```
>> lookfor <name> -all
```

% finds functions with <name> anywhere in description

# Matlab Computation

- Built-in vectors

```
>> x = linspace(a,b,n);  
    % default  $n = 100$   
    % linear spacing  $a \rightarrow b$ , row vector  
  
>> y = logspace(a,b,n);  
    % default  $n = 50$   
    % logarithmic spacing  $10^a \rightarrow 10^b$ , row vector
```

- Built-in matrices

```
>> M = magic(n);    %  $n$  by  $n$  magic square  
>> N = pascal(n);  %  $n$  by  $n$  Pascal triangle
```

# Matlab Constants and Functions

## ■ Characterization Functions

- **Real numbers**

- *factor(x)*   **% prime number component vector**
- *gcd(x,y)*, *lcm(x,y)*, *round(x)*, *ceil(x)*, *floor(x)*

- **Complex numbers** [of form  $x = a+bi$  or  $x = a+bj$ ]

- *abs(x)*, *conj(x)*, *real(x)*, *imag(x)*

- **Logicals** [evaluate to 0 (false) or 1 (true)]

- *isprime(x)*, *isreal(x)*, *isfinite(x)*, *isinf(x)*, ...  
 **$x$  is a logical variable, e.g.,  $x = (5 + 4 \geq 6)$**

# Matlab Constants and Functions

- Characterization Functions
  - **Matrices and vectors**
    - *length, size, det, rank, inv, trace, transpose, ctranspose*
    - *eig, svd, fft, ifft*
  - **Matrix operation functions end with letter 'm'**
    - ***sqrtm***,  $B = \text{sqrtm}(A) \rightarrow B*B = A$  **not  $\text{sqrt}(A(1,1)) = B(1,1)$ , etc.**
    - ***expm***  $B = \text{expm}(A) \rightarrow$   
 $\text{eye}(n) + A + (1/2!)*A*A + (1/3!)*A*A*A + \dots = B$
    - ***logm***  $A = \text{logm}(B) \rightarrow \text{expm}(A) = B$

# Matlab Computation

- Compound Matrix Expressions
  - Created by the user
    - from built-in components
    - from user-created simpler components

## Concatenation of objects:

$$P = M^2 - 2 * (M * N) + N^2$$

## Nesting of groupings:

$$Q = 1 ./ (M ./ (N + 1 ./ M))$$

## Composition of functions:

$$R = \text{inv}(\text{sqrtm}(\cosh(M)))$$



# Matrix Operator Properties

## ■ Associative law

- $(A+B)+C == A+(B+C)$  % true
- $(A.*B).*C == A.*(B.*C)$  % true
- $(A*B)*C == A*(B*C)$  % true

## ■ Distributive law

- $(A + B).*C == A.*C + B.*C$  % true
- $(A + B)*C == A*C + B*C$  % true

## ■ Commutative Law

- $A + B == B + A$  % true
- $A.*B == B.*A$  % true
- $A*B == B*A$  % not true in general

# Matlab Computation

## ■ Operator precedence

- Grouping > Unary > Binary
- Left to right for operations at the same level
- Innermost to outermost for nested groupings
- Computational operations  
 $( ) > \{.^{.}, ^{.}\} > \{\text{unary } +, \text{unary } -\} > \{.*, *, ./, ./.\} > \{+, -\}$
- Logical operations  
 $( ) > \sim > \& > |$
- More details: `>> doc precedence`

# Operator Precedence Example

$$\blacksquare k = 1 - 2^{3/4} + 5*6$$

$$\blacksquare k = (1-2)^{(3/(4+5)*6)}$$

$$k = 1 - 8/4 + 5*6$$

$$k = (-1)^{(3/(4+5)*6)}$$

$$k = 1 - 2 + 5*6$$

$$k = (-1)^{(3/9*6)}$$

$$k = 1 - 2 + 30$$

$$k = (-1)^{(0.333...*6)}$$

$$k = -1 + 30$$

$$k = (-1)^{(2)}$$

$$k = 29$$

$$k = 1$$

# Operator Precedence Example

$$\blacksquare k = 1 - 2^3/4 + 5*6$$

$$k = 1 - 8/4 + 5*6$$

$$k = 1 - 2 + 5*6$$

$$k = 1 - 2 + 30$$

$$k = -1 + 30$$

$$k = 29$$

$\wedge$  is the **first** step

$$\blacksquare k = (1-2)^{(3/(4+5)*6)}$$

$$k = (-1)^{(3/(4+5)*6)}$$

$$k = (-1)^{(3/9*6)}$$

$$k = (-1)^{(0.333...*6)}$$

$$k = (-1)^{(2)}$$

$$k = 1$$

$\wedge$  is the **last** step

# Finite Number Representation

- Decimal expansions rounded off if too long
- Irrational numbers rounded off
- Consequences can include unexpected results
  - Commutivity of addition
$$t = 0.4 + 0.1 - 0.5 \quad \rightarrow \quad 0$$
$$u = 0.4 - 0.5 + 0.1 \quad \rightarrow \quad 2.7756e-017$$
$$t == u \quad \rightarrow \quad \text{logically } \textcolor{red}{False}$$
- Logical relationships may not be as expected
  - $\cos(2*\pi) == \cos(4*\pi) \rightarrow \text{logically } \textcolor{blue}{True}$
  - $\sin(2*\pi) == \sin(4*\pi) \rightarrow \text{logically } \textcolor{red}{False}$   
(but mathematically  $\textcolor{blue}{True}$ )

# Matrix Vectorized Computation

- Example:

Square matrix elements

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

↓

$$B = \begin{pmatrix} 1^2 & 2^2 & 3^2 \\ 4^2 & 5^2 & 6^2 \\ 7^2 & 8^2 & 9^2 \end{pmatrix}$$

- Vectorized

```
>> B = A.^2
```

- Scalar Loop

```
>> for n = 1:3  
    for m = 1:3  
        B(n,m) = (A(n,m)).^2  
    end  
end
```

# Data Analysis

## Uniform

- `uniform_sample = rand(100,1).*(b-a)+a`

## Normal

- `normal_sample = randn(100,1).*σ + μ`

## Poisson

- `poisson_sample = poissrnd(λ,100,1)`

# Descriptive Statistics

- Central tendency variables of vectors
  - mean, median, var, std (of vector elements)
- Operations are on the elements of a vector

If  $a = [1 \ 2 \ 3]$  then  $\text{median}(a) \rightarrow 2$

- Independent operations on matrix columns

If  $A = \begin{matrix} 1 & 2 & 3 \\ 2 & 4 & 9 \\ 6 & 9 & 0 \end{matrix}$  then  $\text{mean}(A) \rightarrow \begin{matrix} 3 & 5 & 4 \end{matrix}$



# Descriptive Statistics

- More collective element properties
  - sum, prod
  - min, max
    - ▶ Position of min or max can be obtained, e.g.

```
>> b = [5 4 3 2 3 4];
```

```
>> [bmin_value bmin_position] = min(b)
```

```
bmin_value = 2
```

```
bmin_position = 4
```

*Note: operations are on absolute values if any elements are complex*

# Descriptive Statistics

- Statistical functions for matrices

Suppose

$$X = \begin{matrix} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 9 \\ 6 & 9 & 0 \end{bmatrix} \end{matrix} \rightarrow Y = \begin{matrix} & \begin{matrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ y_{31} & y_{32} & y_{33} \end{matrix} \end{matrix}$$

**>> Y = cov(X)**

- Covariance matrix for variables in columns of X
- Y has same dimension as X
- $y_{ij}$  = covariance of column  $X(:,i)$  with column  $X(:,j)$
- $y_{ij} = y_{ji}$  (symmetrical)

# Descriptive Statistics

- Statistical functions for matrices

```
>> Z = corrcoef(X)
```

- Correlation coefficient matrix
- Column variables in  $X$ ; Covariant matrix  $Y = \text{cov}(X)$
- $Z$  has same dimension as  $X$  and  $Y$

$$Z_{ij} = \frac{y_{ij}}{\sqrt{y_{ii} * y_{jj}}}$$

# Descriptive Statistics

- Exercises with statistical descriptors -- *data* matrix from *wireless.xls*
  - Find mean quarter hour volume for days of the week  
`>> mean(data)`
  - Find median quarter hour volume on Wednesdays  
`>> median( data(:,4) )`
  - Find sums of volume for quarter hours
    - Transpose the data matrix first – operations by column`>> v = transpose( sum (transpose(data) ) )`
  - Find day of minimum intraday variance  
`>> [minvalue minday] = min( var(data) )`

# Mathematical Functionality

- Matrix eigenvectors and eigenvalues (**QM, factor analysis, images**)
- Roots of polynomials, function zeros (**optimization, cryptography**)
- Convolution & deconvolution of vectors (**polynomial multiplication**)
- Fourier & inverse Fourier transforms (**frequency and power spectra**)
- Numerical differentiation & integration (**observational data analysis**)
- Numerical solution of ODEs, PDEs (**modeling, simulation**)

# Eigenvectors and Eigenvalues

- $A\mathbf{x} = \lambda\mathbf{x}$ 
  - $\mathbf{x}$  eigenvector
  - $\lambda$  eigenvalue
- Defined for a square matrix  $A$ 
  - Number of **eigenvalues** and **eigenvectors** equals the dimension  $n$ 
    - Repeated (degenerate) eigenvalues possible
  - Eigenvalues are roots of a polynomial
    - Defining polynomial  $\det(A - \lambda \cdot \text{eye}(n)) = 0$
    - $n$  degree polynomial in  $\lambda$ ,  $n$  roots:  $\lambda_1 \dots \lambda_n$

# Eigenvectors and Eigenvalues

- Syntax for assigning values:  $[V,D] = \text{eig}(A)$ 
  - $V$  is a matrix with **eigenvectors** in **columns**
  - $D$  has **eigenvalues** on **diagonal**, zeros elsewhere

Note: there can be a single return argument,  $\mathbf{x} = \text{eig}(A)$   
This returns a column vector  $\mathbf{x}$  of eigenvalues

- Relationships (matrix multiplication)
  - $A*V = V*D$       % definition of eigenvector
  - $A = V*D*\text{inv}(V)$       % right multiplication by  $\text{inv}(V)$
  - $D = \text{inv}(V)*A*V$       % left multiplication by  $\text{inv}(V)$

# Eigenvectors and Eigenvalues

- A simple 2-dimensional example

$$A = \begin{pmatrix} 1 & 2 \\ 5 & 4 \end{pmatrix}$$

```
>> A = [1 2; 5 4];
```

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

V =

```
-0.7071 -0.3714  
0.7071 -0.9285
```

D =

```
-1 0  
0 6
```



# Roots and Zeros

- Polynomial notation is a coefficient vector
  - Example: `>> p = [ 1 0 -4 ]`
    - represents  $(1 \cdot x^2 + 0 \cdot x^1 + (-4) \cdot x^0)$  – i.e.,  $x^2 - 4$
  - $(n+1)$  elements for an  $n$  degree polynomial
    - descending ordering by power
  - $n$  elements in a corresponding roots vector
    - Generated as a column vector with **roots** function
      - Example: `>> q = roots(p)`
  - Can be created from a vector of its roots
    - Generated as a row vector with **poly** function
      - Example: `>> r = poly(q)`

# Roots and Zeros

- *roots* and *poly* are inverses of each other

- If  $q = \text{roots}(p)$  then

$$r = \text{poly}(q) = \text{poly}(\text{roots}(p)) = p$$

$$s = \text{roots}(r) = \text{roots}(\text{poly}(q)) = q$$

- roots of a scalar  $z$

```
>> p = [1 0 0 0 0 ... 0 0 -z]; % vector of length  $n + 1$ 
```

**Note: this represents  $x^n = z$**

```
>> q = roots(p) % column vector with all  $n^{\text{th}}$  roots of  $z$ 
```

# Roots and Zeros

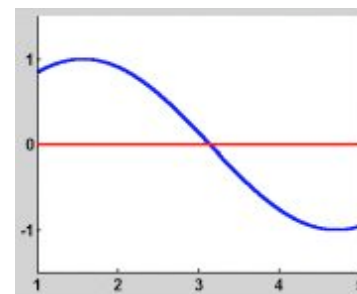
- Function zeros are equivalent to polynomial roots
  - The ***fzero*** function is a function of a function
    - It finds the closest argument with function value 0
  - A starting point or search range needs to be specified
    - Start point must be real and finite
    - Search range limits need function values of opposite sign
    - Syntax:

```
>> fz = fzero(@function, start)
>> fz = fzero('function', [lower upper])
```

# Roots and Zeros

- Function zero example: sine function in [ 1 5 ]

```
>> sinzero = fzero(@sin,3)
sinzero =
    3.14159265358979
```



```
>> sinzero = fzero('sin',[1 5])
sinzero =
    3.14159265358979
```

# Convolution and Deconvolution

- Convolution is like polynomial multiplication
  - Polynomial example of multiplication: **(using algebra)**  
$$c(x) = a(x) \cdot b(x) = (2x^2 + 3x + 4) \cdot (1x^2 - 4x + 7)$$
$$c = 2x^4 - 5x^3 + 6x^2 + 5x + 28$$
  - Equivalent convolution example:  

```
>>c = conv([2 3 4], [1 -4 7]) % i.e., conv(a,b)
      c = [2 -5 6 5 28]
```
- Deconvolution is the inverse of convolution  

```
>> g = conv(f,h)
>> h = deconv(f,g)
>> f = deconv(h,g)
```

# Fast Fourier Transform (FFT)

- The *fft* function transforms vector elements

- Transform:  $[\mathbf{x}_1 \ \mathbf{x}_2 \ \dots \mathbf{x}_N] \rightarrow [\mathbf{X}_1 \ \mathbf{X}_2 \ \dots \mathbf{X}_N]$

$$X_k = \sum_{j=1}^N x_j e^{-2\pi i (j-1)(k-1)/N}$$

- The *ifft* function is an inverse transform

- Inverse:  $[\mathbf{X}_1 \ \mathbf{X}_2 \ \dots \mathbf{X}_N] \rightarrow [\mathbf{x}_1 \ \mathbf{x}_2 \ \dots \mathbf{x}_N]$

$$x(j) = \frac{1}{N} \sum_{k=1}^N X(k) e^{2\pi i (j-1)(k-1)/N}$$

# Fast Fourier Transforms

- 2-D analogs work on matrix columns and rows
  - `fft2` is a two dimensional Fourier transform
  - `ifft2` is a two dimensional inverse transform
  - First by column, then by row

$$\begin{array}{ccccc}
 & & \text{=====}> & & \\
 \parallel & \mathbf{x}_1 & y_1 & z_1 & \xrightarrow{\text{fft}} \underline{\mathbf{x}}_1 \quad \underline{y}_1 \quad \underline{z}_1 \xrightarrow{\text{fft}} \underline{\mathbf{x}}_1 \quad \underline{y}_1 \quad \underline{z}_1 \\
 \parallel & \mathbf{x}_2 & y_2 & z_2 & \xrightarrow{\text{fft}} \mathbf{x}_2 \quad y_2 \quad z_2 \xrightarrow{\text{fft}} \mathbf{x}_2 \quad y_2 \quad z_2 \\
 \parallel & \mathbf{x}_3 & y_3 & z_3 & \xrightarrow{\text{fft}} \mathbf{x}_3 \quad y_3 \quad z_3 \xrightarrow{\text{fft}} \mathbf{x}_3 \quad y_3 \quad z_3 \\
 & \vee & & & 
 \end{array}$$

`fft2(A)` is the same as `(fft (fft (A)).')).'`

# Differentiation and Integration

- Polynomial derivatives
  - Derivatives can be erratic with experimental data
    - Optional initial smoothing with function *polyfit*
    - Syntax: `>> p = polyfit(x,y,n)`
      - Least squares fit to polynomial of degree n
      - Vector elements are descending power coefficients
  - For polynomial coefficient vector, use *polyder*
    - *polyder(x)* → derivative polynomial vector
    - Dimension one less than argument vector (final 0 omitted)
    - Based on  $dx^n/dx = nx^{n-1}$
    - Second derivative: *polyder(polyder(x))*



# Differentiation and Integration

Example:

```
>> x = 0:0.1:2*pi      >> y = sin(x)
>> p = polyfit(x,y,5)  >> y2 = polyval(p,x)
>> plot(x,y)           >> plot(x,y2)
>> w = polyder(p)
>> yderiv = polyval(w,x)
>> plot(x,yderiv)
```

# Differentiation and Integration

- Non-polynomial numerical derivatives

- Approximation using the *diff* function

- The diff function creates a vector of differences

- Differences are of consecutive argument elements

- Example:  $\mathbf{x} = [1 \ 2 \ 4 \ 7 \ 11]$  and  $\mathbf{y} = [1 \ 3 \ 9 \ 21 \ 41]$

- ▶  $\text{diff}(\mathbf{x}) = [(2-1) \ (4-2) \ (7-4) \ (11-7)] = [1 \ 2 \ 3 \ 4]$

- ▶  $\text{diff}(\mathbf{y}) = [(3-1) \ (9-3) \ (21-9) \ (41-21)] = [2 \ 6 \ 12 \ 20]$

- Numerical derivative  $dy/dx$  (very crude):

- ```
>> yderiv = diff(y) ./ diff(x) = [ 2 3 4 5 ]
```

- Second derivative (even more crude):

- ```
>> yderiv2 = diff(diff(y) ./ diff(x)) ./ diff(diff(x))
```

# Differentiation and Integration

Example:

```
>> x = 0:0.1:2*pi    >> y = sin(x)
>> dx = diff(x)      >> dy = diff(y)
>> yderiv2 =dy./dx
>> x2 = x(1:end-1)+dx./2
>> plot(x2,yderiv2)
```

# Differentiation and Integration

- Numerical integration with the *polyint* function
  - Argument is a polynomial coefficient vector
    - Non-polynomials might be approximated with *polyfit*
    - Generated from

$$\int k^n dk = \frac{k^{n+1}}{n+1}$$

- Syntax: `>> y = polyint(x,c)`
- Default constant of integration is  $c = 0$

# Differentiation and Integration

- Numerical integration with the *quad* function

- Syntax: `quad('function_name', lower, upper)`

- Function name can be user defined in an m-file
- Uses an adaptive Simpson's Rule for quadrature
- Example:

```
>>area = quad('x.^2 - 6.*x + 5', 1, 5)
area =
-10.6667
```

- Function does not need to be a polynomial

```
>>area = quad('sin', 0, pi)
area =
1.999999999619084
```

# Solving Differential Equations

- Ordinary Differential Equations (ODE's)

- Solvers are for linear 1<sup>st</sup> order equations
- Higher order recast as system of 1<sup>st</sup> order
  - Example:  $a \cdot x'' + b \cdot x' + c \cdot x + d = 0$

[set  $x_1 = x$  and  $x_2 = x_1'$ ]

$$\begin{aligned} x_1' &= 0 \cdot x_1 + x_2 + 0 \\ x_2' &= (-c/a) \cdot x_1 + (-b/a) \cdot x_2 + (-d/a) \end{aligned}$$

- Solver functions
  - ode23, ode45 (Runge-Kutta methods)
  - ode15s, ode23s (for stiff systems)

# Statistical Functionality

- Regression (**inferential statistics, model fitting**)
- Matlab Statistical Toolbox (**inferential statistics**)

# Regression and Curve Fitting

- Polynomial function fitting with *polyfit*

- Syntax:

- ```
>> polycoeffs = polyfit(x, y, n)
```

- x* is a vector of independent variable values

- y* is a vector of dependent variable values

- n* is the order of the polynomial desired

- ▶ select *n* = 1 for linear regression, *n* = 2 for quadratic, etc.

- Function evaluation with *polyval*

- ```
>> ycalc = polyval(polycoeffs, x)
```

- ```
>> residuals = ycalc - y
```



# Regression and Curve Fitting

- Exercise
  - Fit the curve for population center migration
    - ▶ Examine the m file [\*popcenterfit.m\*](#)
      - \* Plot of *north latitude* as a function of *west longitude*
      - \* Coefficients for linear fit (ordinary regression)
      - \* Coefficients for cubic fit (cubic regression)
      - \* Plot of linear and cubic regression lines
    - ▶ Execute the script
    - ▶ Compare linear and cubic fits on plot
    - ▶ Look at sum of squares of residuals

# Regression and Curve Fitting

Linear

```
>>X = [ones(length(west)),west]
```

```
>>[b,bint,r,rint,stats] = regress(north,X)
```

Cubic

```
>> X3 =
```

```
    [ones(length(west)),west,west.^2,west.^3]
```

```
>>[b,bint,r,rint,stats] = regress(north,X3)
```

# Advanced Statistics

## ■ Matlab Statistics Toolbox

- Help > Product Help > Statistics Toolbox > Function Reference
  - ANOVA
  - Regression
  - Hypothesis testing
  - Probability distributions
  - ... and more
- Example using observational data from wireless data

**% subtract out mean, divide by variance**

```
>> normdata = zscore(data);
```

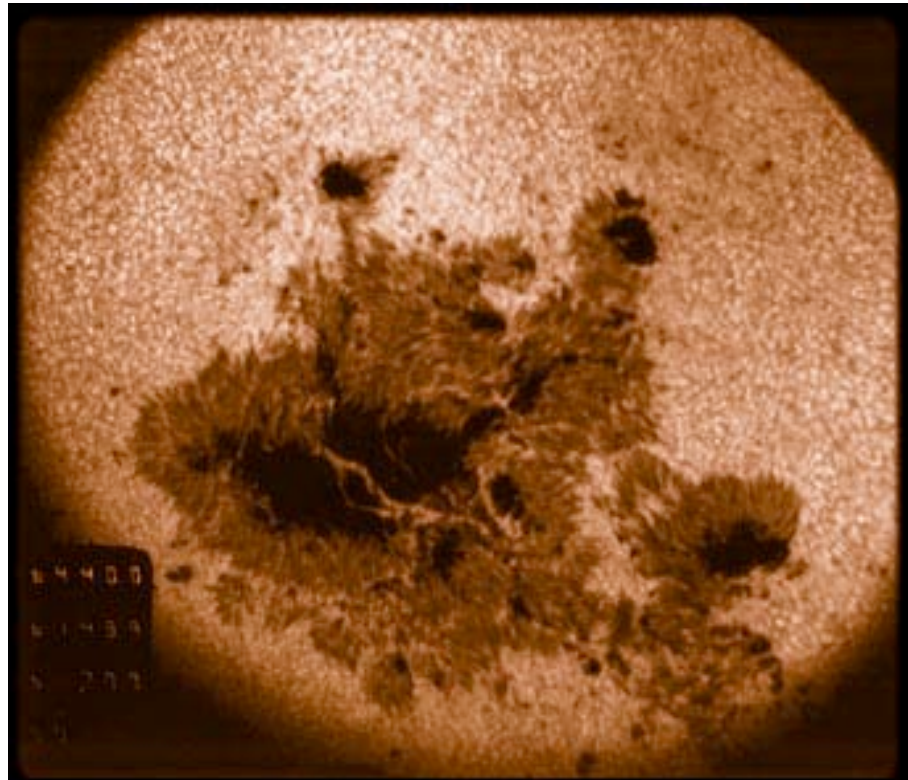
```
>> plot(normdata)
```

# Matlab Visualization & Programming

- Graphing
- Calling M-file scripts and functions
- Flow control and string evaluation
- Debugging

# Graphics and Data Display

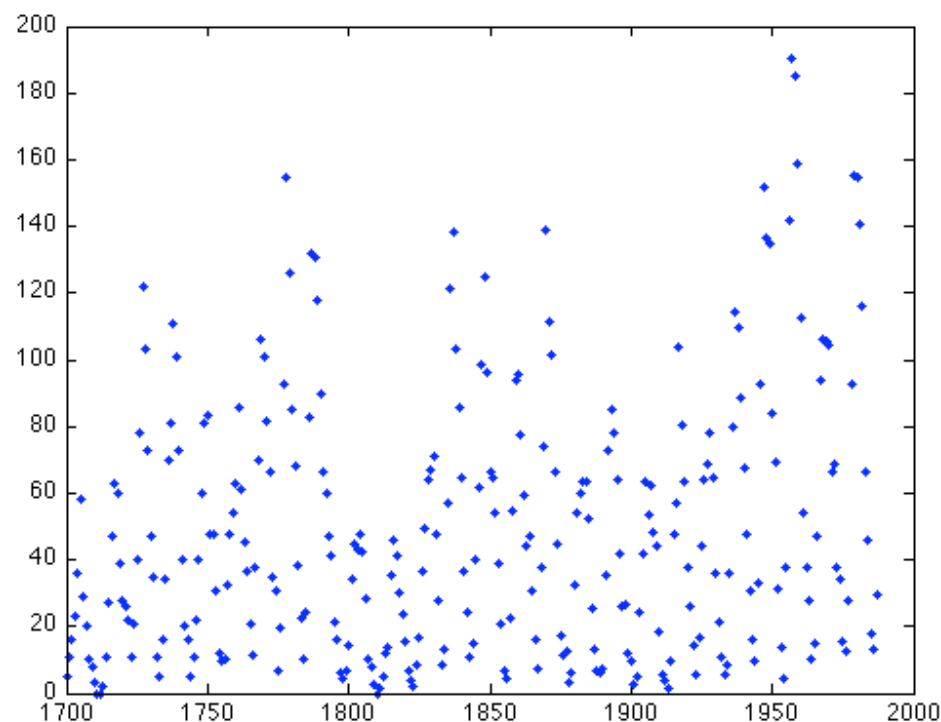
Matlab demo data file: sunspot.dat



# Graphics and Data Display

Syntax: `plotname(x, y, 'style')`

- e.g. `plot(x, y, 'b.')`



# Graphics and Data Display

- Exercise

- Load the file *sunspot.dat*
- Equate variable *year* with the first column  
Use command: `>> year = sunspot(:,1);`
- Equate variable *spots* with second column
- Make a simple line plot of *spots* vs. *year*
- Superimpose data points as red asterisks \*
- Use `'r*'` for 'plotstyle'
- Create a title: `>> title('Sunspots by year')`

# Graphics and Data Display

- 2-D display variants

- Cartesian coordinates

- >> `bar(x,y)`            % vertical bar graph
  - >> `barh(x,y)`        % horizontal bar graph
  - >> `stem(x,y)`        % stem plot
  - >> `area(x,y)`        % color fill from horizontal axis to line
  - >> `hist(y,N)`        % histogram with N bins (default N = 10)

- Polar coordinates

- >> `pie(y)`
  - >> `rose(theta,N)`    % angle histogram, N bins (default 10)



# Graphics and Data Display

- Layout options
  - Single window
    - ▶ Replace data  
`>> hold off;`
    - ▶ Overlay on previous data  
`>> hold on;`
    - ▶ Subplots – rectangular array
  - Multiple windows
    - `>> figure(n)    %  $n^{th}$  plot`
    - `>> figure(n+1) %  $(n+1)^{th}$  plot`
    - `-- etc. --`

# Graphics and Data Display

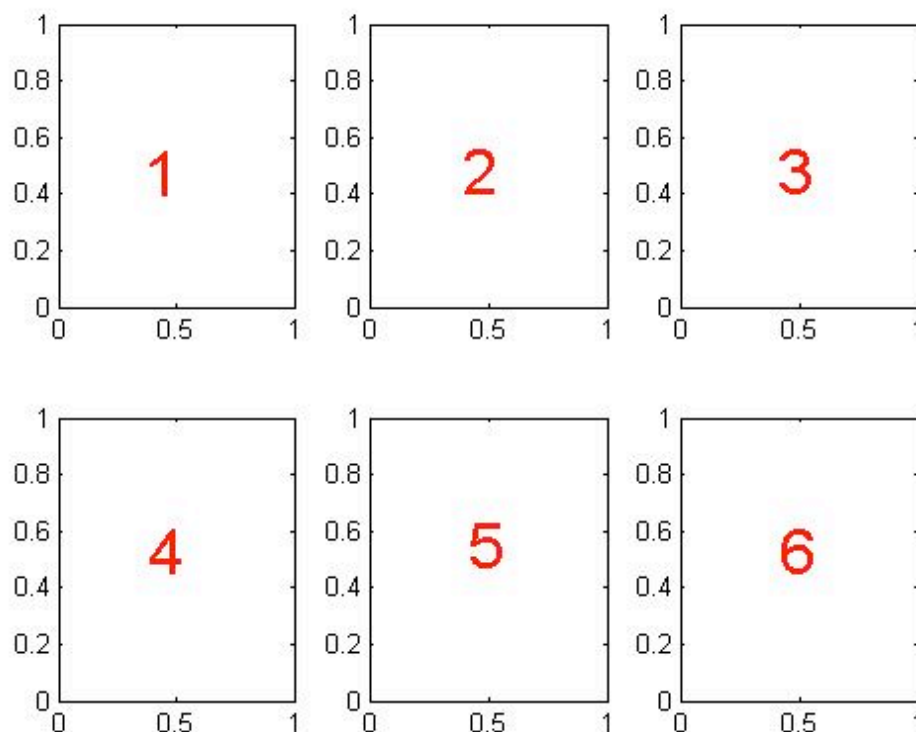
## ■ Subplot syntax

```
>> subplot(i, j, k) , plotname(x, y, 'style')
```

- *i* is the number of rows of subplots in the plot
- *j* is the number of columns of subplots in the plot
- *k* is the position of the plot
  - Sequential by rows, then columns
  - Example: subplot(2, 2, *N*)
    - N* = 1 for upper left, *N* = 2 for upper right
    - N* = 3 for lower left, *N* = 4 for lower right

# Graphics and Data Display

Subplot layout positions (2 x 3 example)



# Graphics and Data Display

- **Exercise:** Make subplot array for *spots* vs. *year*

|                                |                        |
|--------------------------------|------------------------|
| <code>area(year, spots)</code> | % place in upper left  |
| <code>bar(year, spots)</code>  | % place in upper right |
| <code>barh(year, spots)</code> | % place in lower left  |
| <code>hist(spots)</code>       | % place in lower right |

-----

```
>> subplot(2,2,1) , area(year, spots) % etc.
```

# Saving and Exporting Graphics

- **Save with *File > Save* or *File > Save As***
  - Default **location** is the **Current Directory**
  - Default file **type** is Matlab Figure file (\*.fig)
  - Default file **name** is **untitled.fig**
  - "Save As" Window allows changes to defaults
- **Pre-process figure image with *File > Export Setup***
  - Dimensions and settings can be changed
  - Click *Export* button
  - New *Save As* Window appears for editing

# Types and Structures of M-files

## ■ M-file scripts

- Sequence of Matlab command lines
- No required beginning or ending line
- Sequential execution of commands, top to bottom
- Equivalent to macros and subroutines
- Can be called by other M-file scripts
- No passing of variables as arguments  
but when called --
  - ▶ variables go to workspace
  - ▶ Pre-existing variables with same name overwritten

# Types and Structures of M-files

- M-file functions

- First line is **function** declaration
  - Function **elements** to be returned identified
  - Variables can be imported as **arguments**
  - Function name should correspond to m-file name
  - First line format for m-file **function\_name.m** :

```
function [f1 f2 ...] = function_name(a1, a2, ...)
```

- Computed function element values returned
- Can be called by other m-file functions
- Can call themselves (recursion)
- Have their own local workspaces

# Simple M-file Examples

- Example of trivial M-file script: *little.m*

6

(Note -- a single character can comprise a script)

```
>> little
```

```
ans =
```

```
6
```

- Example of trivial M-file function: *doubleval.m*

```
function f = doubleval(x)
```

```
f = 2*x;
```

```
>> doubleval(4)
```

```
ans =
```

```
8
```



# Internal Documentation

- Uncompiled and unexecuted text after %
  - Can be placed anywhere on a line
  - Can be placed on any line in the file
  - Exception: % within single quotes is literal
- Built-in help file can be constructed
  - All consecutive lines starting with %
    - At the top of m-file scripts
    - Immediately after m-file function declaration line
- Example: *tripleval.m*

```
>> tripleval(5)
y =
    15
>> help tripleval
```

# Passing function variables

- Variables within functions are local
  - Can be shared if declared global
    - Syntax:  
`global var1`
    - Global variable needs global declaration
      - ▶ in calling script *before it is defined or used*
      - ▶ In called function code *before it is defined or used*
    - Can share value among subfunctions in a function
      - ▶ Each subfunction must declare it global

# Subfunctions

- Functions can have private subfunctions
  - Embedded within the parent function
  - No separate m-file
  - First line has same format
  - Last line of subfunction must be **end**
    - Subfunctions can have their own subsubfunctions

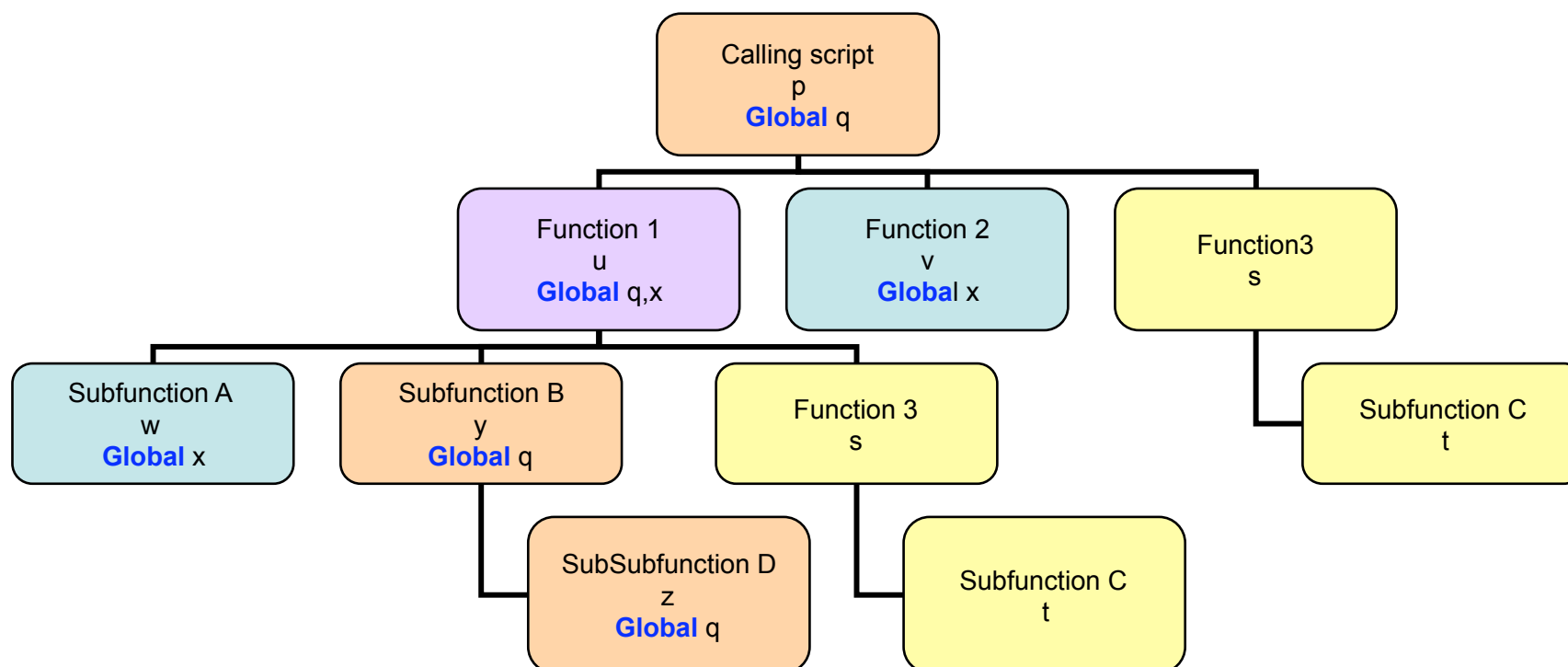
% [ i.e.,  $h = 3 \cdot (2 \cdot 4)$  ]

# Sharing variable values

Dark orange knows about value of q

Blue knows about value of x

Pink knows about value of both q and x



# Passing function variables

- Programming example : *xseven.m*
  - Long method of multiplication by 7

*x\*7*  
function

= *x\*4* + *x\*3*  
subfunction subfunction

= *x\*1* + *x\*3* + *x\*1* + *x\*2*  
global subfunction global subsubfunction

= [ *x\*1* + (*x\*1* + *x\*2*) ] + [ *x\*1* + *x\*2* ]  
subsubfunction subsubfunction

*7* = [ 1 + ( 1 + 2 ) ] + [ 1 + 2 ]

# Function Recursion

- Function scripts can call themselves
  - Example - finding a triangular number: *triangular.m*

| <u>t(1)</u> | <u>t(2)</u> | <u>t(3)</u>  | <u>t(4)</u>     |
|-------------|-------------|--------------|-----------------|
| <u>1</u>    | 1           | 1            | 1               |
|             | 2 <u>3</u>  | 2 3          | 2 3             |
|             |             | 4 5 <u>6</u> | 4 5 6           |
|             |             |              | 7 8 9 <u>10</u> |

$t(1) = 1$ , otherwise

$t(n) = n + t(n-1)$

$= n + ( (n-1) + t(n-2) ) =$

$= n + ( (n-1) + ( (n-2) + t(n-3) ) \dots$

# String Evaluation : *eval*

- Matlab can parse strings as functions
  - Define an expression or function as a string
    - Examples:

```
>> var35 = 'xseven(5)';
```

```
>> x = 'sin(n)';
```

- Evaluate with *eval* function

```
>> thirtyfive = eval(var35)
```

```
    thirtyfive = 35
```

```
>> n = 1;
```

```
>> y = eval(x)
```

```
    y = 0.84147 % i.e., [sin(1)]
```

- *eval* loads the entire command interpreter



# String evaluation : *feval*

- Function strings can be evaluated directly
  - *feval* loads only what is needed – thus faster
    - Function inputs are separate arguments

- Examples

```
>> sample35 = feval('xseven',5)
      sample35 = 35
```

```
>> n = 1;
>> x = 'sin(n) '
>> z = feval('asin', eval(x))
      z = 1           % i.e., [asin(sin(1))]
```

# Storage and transport variables

- Easy to reference contents

- Multi-dimensional arrays

- Elements need to be of same type (number or char)
    - Parentheses as delimiters, e.g. : **A(4,2,5,1) = 16;**

- Awkward to reference contents

- Cells

- Array of arrays – each can be of a different type
    - Curly braces as delimiters, e.g. : **B{2,3} = [1 2; 3 4];**

- Structures

- Root name with extensions and index – types can be mixed
    - Periods as delimiters, e.g. : **C.color(2,1:5) = 'green';**  
**C.colnum(2,1:3) = [0 1 0];**

# Multidimensional Arrays

- **Matlab matrix operations are on 2-D arrays**
  - Storage is by concatenation of columns (1-dimensional)

Suppose

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

- Stored as [ 1 4 7 2 5 8 3 6 9].'
- Thus  $A(2,3) = A(8) = 6$

# Multidimensional Arrays

- Equal rows can be concatenated horizontally
- Equal columns can be concatenated vertically
- Examples: consolidation by concatenation

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 6 & 5 & 4 \end{pmatrix} \quad B = \begin{pmatrix} 4 & 5 & 6 \\ 3 & 2 & 1 \end{pmatrix} \quad C = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \quad D = \begin{pmatrix} 6 & 5 \\ 4 & 3 \\ 2 & 1 \end{pmatrix}$$

```
>> E = [ A B ];    % (2 rows, 6 columns)
>> F = [ C; D ];    % (6 rows, 2 columns)
```

# Multidimensional Arrays

- Arrays can be expanded by assignment
  - Thus size adjustments can be made for concatenation

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 6 & 5 & 4 \end{pmatrix}$$

- **Reference** outside boundary gives error
  - `>> x = A(3,2) ;` ??? Index exceeds matrix dimensions

- **Assignment** outside boundary gives expansion

`>> A(3,2) = 1;`

$$A = \begin{pmatrix} 1 & 2 & 2 \\ 6 & 5 & 4 \\ 0 & 1 & 0 \end{pmatrix}$$

# Debugging

- Matlab has a built-in debug help
  - **Error messages** in Command Window (**red font**)
  - Editor has debug item on navigation bar
    - Choose **Run** from menu
      - Diagnostic messages in Command Window
- Programmer can modify code
  - Remove output suppression from semi-colons
  - Insert markers for screen display: `disp('marker_text')`
- The **db\*** functions can be used
  - Allows pauses to examine status
    - `>> dbstop` at 40 in badfile.m (stops execution at line 40)
    - `>> dbcont` (continues execution until next stop point)
    - `>> dbquit` (exits debugging mode)

# Debugging

- Types of errors
  - **Compile time errors**
    - Errors in syntax
    - Dimension incompatibilities
    - Referenced function not in Path
  - **Run time errors**
    - Divide by zero; vector, matrix index not a positive integer
    - Argument of wrong data type
  - **Unrecognized errors**
    - Errors in logic
    - Errors from computation (round-off, etc.)
    - Errors from unintended overwriting

# Debugging

- Example of faulty syntax: *ebase.m*
  - Computing the value of the natural log base
$$e^k = \frac{k^0}{0!} + \frac{k^1}{1!} + \frac{k^2}{2!} + \dots = \sum_{n=0}^{\infty} \frac{k^n}{n!}$$
  - For  $k = 1$ , the  $n^{\text{th}}$  term appears to be  $1^n/n!$ 
    - But Matlab does not allow array index to be 0
    - Error can be fixed by starting series at  $n = 1$
    - Then the  $(n+1)^{\text{th}}$  term is  $1^n/n!$ , i.e.,  $1/n!$
- Example of faulty logic
  - $n^{\text{th}}$  term set to  $1/n$  instead of  $1/n!$ 
    - Fix by replacing  $n$  with *factorial*( $n$ )



# Debugging

- **Checking on program stalls and long processes**
  - Program can be stalled by an infinite loop
  - Apparent stalling from slow convergence
  - The **keyboard** command in an m-file can help
    - Gives control back to the Command Window prompt
    - Special `K>>` prompt
      - Status of workspace variables can be examined
      - Values of workspace variables can be changed
    - Control returned to m-file with **eturnr** command
    - End function with **dbquit**

# Debugging

- Example for testing long, slow processes: *zeta.m*
  - Calculation of the Riemann zeta function,  $\zeta(n)$

$$\zeta(n) = \frac{1}{1^n} + \frac{1}{2^n} + \dots = \sum_{k=1}^{\infty} \frac{1}{k^n}$$

- $\zeta(1)$  diverges to  $\infty$  (infinite loop for series term addition)
  - So-called harmonic series will never converge
- $\zeta(2)$  converges to  $\pi^2/6$  very, very slowly
  - Still off in seventh decimal place after two million terms
- $\zeta(4)$  converges quite a bit faster to  $\pi^4/90$ 
  - Ten decimal place accuracy after 9743 terms
- tic/toc for timing

# Interactions with External Code

- Matlab m-files are ascii based
  - External word processors can read m-files
    - Example for MS Word: *File -> Open -> matlab\_file.m*
  - Matlab Editor can read external ascii files
    - Example for Matlab Editor: *File -> Open -> msword\_file.txt*
  - Matlab Editor loads but can't use external binary text
    - Example: *File -> Open -> msword\_file.doc* **[cannot compile]**
  - Matlab command line can read external ascii
    - Example: `>> load msword_file.txt`
    - Example: `>> load msword_file.doc` **[error message]**

# Thanks for Coming!

- Please complete course evaluation:
  - <https://www.surveymonkey.com/s/LRJ6WJK>
- More free individual assistance available (e-mail or consulting appointment)