# Scientific Computing Lecture Series
# Introduction to MATLAB Programming

Mehmet Alp Üreten*

*Scientific Computing, Institute of Applied Mathematics

Lecture II
Control Loops, Advanced Data Structures, Graphics, and Symbolic Toolbox

# Lecture II–Outline

# Rational and Logical Operators

- Boolean values: zero is false, nonzero is true
- Some of the logical operators:

| Operator | Meaning |
|---|---|
| $<, <=, >, >=$ | less than, less than or equal to, etc. |
| $==$, $\sim=$ | equal to, not equal to |
| & | logical AND |
| $\|$ | logical OR |
| $\sim$ | logical NOT |
| all | all true |
| any | any true |
| xor | Xor |

# Logical Indexing

- Construct a matrix R

```
>> R = rand(5)
R =
0.8147    0.0975    0.1576    0.1419    0.6557
0.9058    0.2785    0.9706    0.4218    0.0357
0.1270    0.5469    0.9572    0.9157    0.8491
0.9134    0.9575    0.4854    0.7922    0.9340
0.6324    0.9649    0.8003    0.9595    0.6787
```

- Test for some logical cases

```
>> R(R<0.15)'
ans =
0.1270    0.0975    0.1419    0.0357
>> isequal(R(R<0.15), R(find(R<0.15)))
ans =
1
```

# If/Else/Elseif

- The general form of the if statement is

```
if    expression1
        statements1
elseif    expression2
        statements2
        ⋮
else
        statements
end
```

- No need for parentheses:command blocks are between reserved words

# Switch

- The general form of the switch statement is

```
switch    variable
   case    variable value1
         statements1
   case    variable value2
         statements2
         ⋮
   otherwise    (for all other variable values)
         statements
end
```

# Try–Catch

- The general form:

```
try
    statements1
catch
    statements2
end
```

- A simple example:

```
a = rand(3,1);
try
    x = a(10);
catch
    disp('error')
end
```

# For

- for loops: use for a known number of iterations
- The basic syntax is

```
for variable = expr
    statements;
end
```

- A simple example:

```
M = rand(4,4); suma = 0;
for i = 1:4
    for j = 1:4
        suma = suma + M(i,j);
    end
end
fprintf('sum = %d\n',suma);
```

# While

- Don't need to know number of iterations
- The basic syntax is

```
while   a logical test
        commands to be executed
        when the condition is true
end
```

- A simple example:

```
S=1; n=1;
while S+(n+1)^2 < 100
    n=n+1; S=S+n^2;
end
>> [n,S]
ans = 6    91
```

- Beware of infinite loops!

# Remarks

- break - immediately jumps execution to the first statement after the loop.
- return - immediately end a functions routine.
- Precaution: Avoid i and j if you are using complex values.
- Loops are very inefficient in MATLAB. Only one thing to do: AVOID THEM !!!
- Try using built–in–functions instead
- Allocating memory before loops greatly speeds up computation times !!!

# Find

- find returns indices of nonzero values. It can simplify code and help avoid loops
- basic syntax: index = find(condition)

```
>> x = rand(1,10)
x =
Columns 1 through 5
0.4505    0.0838    0.2290    0.9133    0.1524
Columns 6 through 10
0.8258    0.5383    0.9961    0.0782    0.4427

>> inds = find(x>0.4 & x<0.7)
inds =
1     7    10
>> x(inds)
ans =
0.4505    0.5383    0.4427
```

# Scoping Exceptions

- A global variable is a factor whose value can be accessed and changed from any other workspaces
- Any variable may be declared global
- The trouble with global variables is that they do not scale well to large or even moderately sized projects

- A persistent variable is a factor whose value is preserved between invocations to that particular function.
- Any variable may be declared global
- It is less general than a global variable and requires a little care to ensure correct use

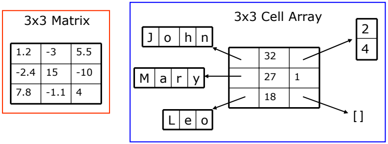
- Persistent variables can be used to record information about a function's internal state, or to preserve costly preliminary results that can be reused later.
- Compute the Fibonacci numbers:

```
function y = fib(n)
persistent f
if length(f) < 2,
    f = [1 1];
end
for k = length(f)+1:n
    f(k) = f(k-2) + f(k-1);
end
y = f(1:n);
```

- In future calls to fib, any previously computed members of the sequence are simply accessed rather than recomputed.

# Cell Arrays

- Cell arrays are a mechanism for gathering dissimilar objects into one variable.
- Indexed like regular numeric arrays, but their elements can be anything, including other cell arrays.
- Cell arrays can have any size and dimension, and their elements do not need to be of the same size or type.
- Because of their generality, cell arrays are mostly just containers
- Created or referenced using curly braces {} rather than parentheses.




- Cell initialization:

```
>> a = cell(3,2);
>> a = {'hello world', [1,5,7], rand(2,4)}
```

- To access a cell element, use curly braces {}

```
>> a = {'hello world', [1,5,7], rand(2,4)}
a = 'hello world'    [1x3 double]    [2x4 double]
>> a{1,1}
ans = hello world
>> a{1,3}
ans =
0.9058    0.9134    0.0975    0.5469
0.1270    0.6324    0.2785    0.9575
```

- ```
  T = cell(1,9);
  T(1:2) = { [1], [1 0] };
  for n=2:8
      T{n+1}=[2*T{n} 0] - [0 0 T{n-1}];
  end
  ```

  ```
  >> T
  T =
  Columns 1 through 5
  [1] [1x2 double] ... [1x5 double]
  Columns 6 through 9
  [1x6 double] [1x7 double] ... [1x9 double]
  ```
  ```

# Structures

- Structures are essentially cell arrays that are indexed by a name rather than by number.
- The field values can be anything.
- Values are accessed using the dot notation.
- ```
  >> student.name = 'Moe';
  >> student.homework = [10 10 7 9 10];
  >> student.exam = [88 94];
  >> student
  student =
  name: 'Moe'
  homework: [10 10 7 9 10]
  exam: [88 94]
  ```
  ```

- Add another student:

```
>> student(2).name = 'Curly';
>> student(2).homework = [4 6 7 3 0];
>> student(2).exam = [53 66];
>> student
student =
1x2 struct array with fields:
homework
exam
```

- Array and field names alone create comma-separated lists of all the entries in the array.

```
>> roster = {student.name}
roster =
'Moe' 'Curly'
```

# cell2mat – cell2struct

- cell2mat    Convert cell array to ordinary array of the underlying data type

```
C = {[1],    [2 3 4];
[5; 9], [6 7 8; 10 11 12]}
C =
{[       1]}    {1x3 double}
{2x1 double}    {2x3 double}
A = cell2mat(C)
A =
1     2     3     4
5     6     7     8
9    10    11    12
```

- cell2struct    Convert cell array to structure array

```
>> fields={'number','name','value'};
>> c={'one','Hamdullah',3;'two','Hamdi',7};
>> cStruct=cell2struct(c,fields,2)
cStruct =  2x1 struct array with fields:
number
name
value
```

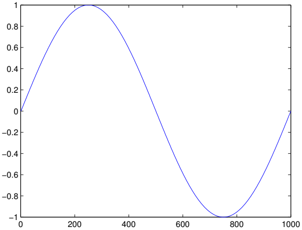
# Basic Plotting

- plot() generates dots at each $(x, y)$ pair and then connects the dots with a line

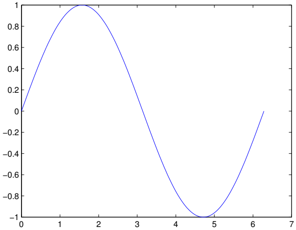
```
>> x = linspace(0,2*pi,1000);
>> y = sin(x);
>> plot(x,sin(x))
```

- Plot values against their index

```
>> plot(y)
```




(a) plot(y)




(b) plot(x,sin(x))

# Basic Plotting

- figure To open a new Figure and avoid overwriting plots

```
>> x = [-pi:0.1:pi];
>> y = sin(x);
>> z = cos(x);
>> plot(x,y);   (automatically creates a new Figure!)
>> figure
>> plot(x,z);
```

- close Close figures

```
>> close 1
>> close all
```

- hold on/off Multiple plots in same graph

```
>> plot(x,y);   hold on
>> plot(x,z,'r');  hold off
```

# Basic Plotting

- To make plot of a function look smoother, evaluate at more points
- *x* and *y* vectors must be same size or else you will get an error

```
>> plot([1,2],[1 2 3])
??? Error using ==> plot
Vectors must be the same lengths.
```

- To add a title

```
>> title('My first title')
```

- To add axis labels

```
>> xlabel('x-label')
>> ylabel('y-label')
```

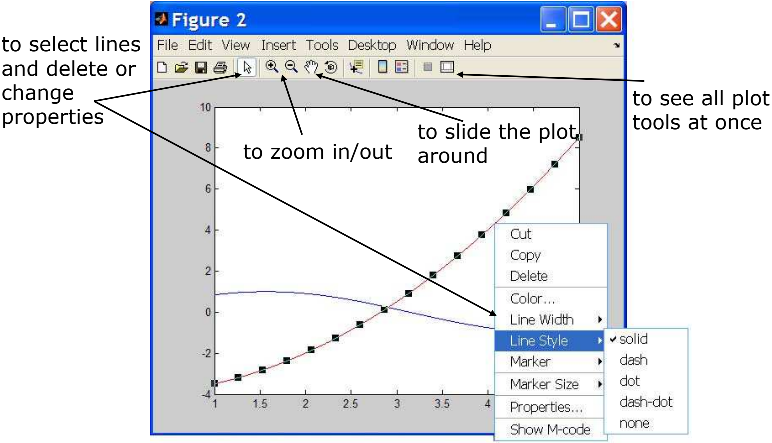
- Can change the line color, marker style, and line style by adding a string argument

```
>> plot(x,y,'k.-');
```

- Basic legend syntax:

legend('First plotted','second plotted', 'Location','Northwest')

# Playing with the Plot

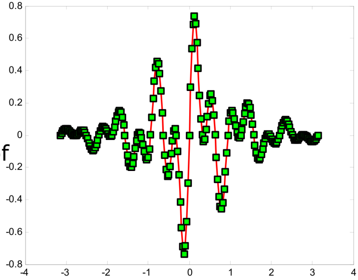


# Line and Marker Options

- Everything on a line can be customized

```
» plot(x,y,'--s','LineWidth',2,...
               'Color', [1 0 0], ...
               'MarkerEdgeColor','k',...
               'MarkerFaceColor','g',...
               'MarkerSize',10)
```

You can set colors by using a vector of [R G B] values or a predefined color character like 'g', 'k', etc.




- See **doc line_props** for a full list of properties that can be specified

# Basic Plotting

- semilogx        logarithmic scales for x-axis
- semilogy        logarithmic scales for y-axis
- loglog        logarithmic scales for the x,y-axes
- plotyy        2D line plot: y–axes both sides

```
>> plotyy(X1,Y1,X2,Y2) (plot X1,Y1 using left axis and X2,Y2 using right axis)
```

- errorbar        errors bar along 2D line plot

```
>> errorbar(X,Y,E)   (create 2D line plot from data X, Y
with symmetric error bars defined E)
>> errorbar(X,Y,L,U) (create 2D line plot from data X, Y  with upper error
bar defined by U and lower error bar defined by L)
```

# Basic Plotting

- bar, barh        Vertical, horizontal bar graph

```
>> x = 100*rand(1,20);
>> bar(x)
>> xlabel('x');
>> ylabel('values');
>> axis([0 21 0 120]);
>> title('First Bar');
```

- pie, pie3        2D, 3D pie chart

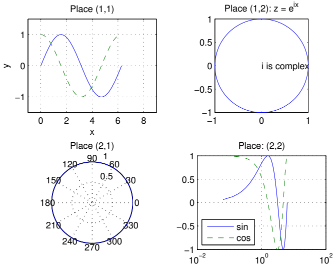
```
>> x = 100*rand(1,5);
>> pie(x)
>> title('my first pie');
>> legend('val1','val2','val3','val4','val5');
```

- area        Filled are 2D plot

```
>> x = [-pi:0.01:pi]; y=sin(x);
>> plot(x,y); hold on;
>> area(x(200:300),y(200:300));
>> area(x(500:600),y(500:600)); hold off
```

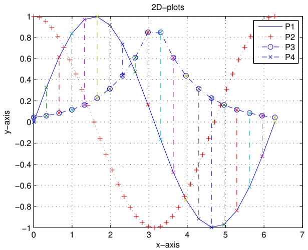
- subplot()    Multiple plots in the same figure

```
>> x = linspace(0,2*pi);
>> subplot(2,2,1); plot(x, sin(x), x, cos(x), '--')
>> axis([-1 9 -1.5 1.5])
>> xlabel('x'), ylabel('y'), title('Place (1,1)'), grid on
>> subplot(2,2,2); plot(exp(i*x)), title('Place (1,2): z = e^{ix}')
>> axis square, text(0,0, 'i is complex')
>> subplot(2,2,3); polar(x, ones(size(x))), title('Place (2,1)')
>> subplot(2,2,4); semilogx(x,sin(x), x,cos(x), '--')
>> title('Place: (2,2)'), grid on
>> legend('sin', 'cos', 'Location', 'SouthWest')
```



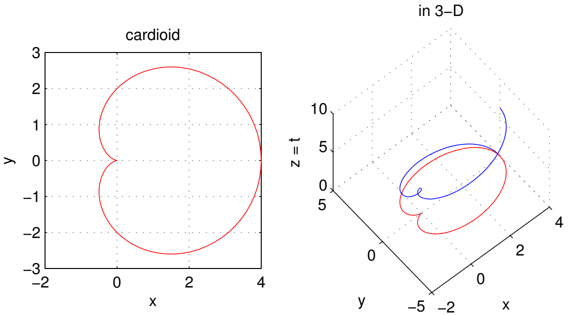
# Continue...

```
>> x1 = linspace(0,2*pi,20); x2 = 0:pi/20:2*pi;
>> y1 = sin(x1); y2 = cos(x2); y3 = exp(-abs(x1-pi));
>> plot(x1, y1), hold on                    % "hold on" holds the current picture
>> plot(x2, y2, 'r+:'), plot(x1, y3, '-.o')
>> plot([x1; x1], [y1; y3], '-.x'), hold off
>> title('2D-plots')                        % title of the plot
>> xlabel('x-axis')                         % label x-axis
>> ylabel('y-axis')                         % label y-axis
>> grid                                     % a dotted grid is added
>> legend('P1', 'P2', 'P3', 'P4')           % description of plots
>> print -deps fig1                         % save a copy of the image in a file
% called fig1.eps
```



# 3-D Plotting: plot3

```
>> t = linspace(0,2*pi); r = 2 * ( 1 + cos(t) );
>> x = r .* cos(t); y = r .* sin(t); z = t;
>> subplot(1,2,1), plot(x, y, 'r'), xlabel('x'), ylabel('y')
>> axis square, grid on, title('cardioid')
>> subplot(1,2,2), plot3(x, y, z), xlabel('x'), ylabel('y'), hold on
>> axis square, grid on, title('in 3-D'), zlabel('z = t')
>> plot3(x, y, zeros(size(x)), 'r'), view(-40, 60)
```



# Surf

- meshgrid(x,y) produces grids containing all combinations of $x$ and $y$ elements, in order to create the domain for a 3D plot of a function $z = f(x, y)$
- surf puts vertices at specified points in space $x, y, z$, and connects all the vertices to make a surface
  - Make the x and y vectors

    ```
    » x=-pi:0.1:pi;
    » y=-pi:0.1:pi;
    ```

  - Use meshgrid to make matrices (this is the same as loop)

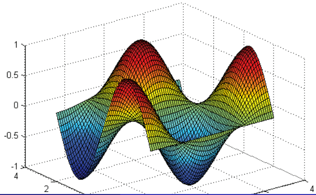
    ```
    » [X,Y]=meshgrid(x,y);
    ```

  - To get function values, evaluate the matrices

    ```
    » Z =sin(X).*cos(Y);
    ```

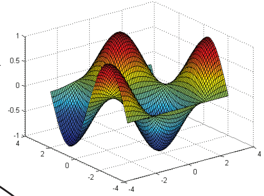
  - Plot the surface

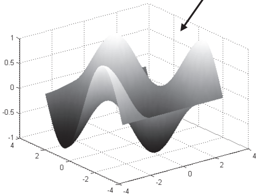
    ```
    » surf(X,Y,Z)
    ```

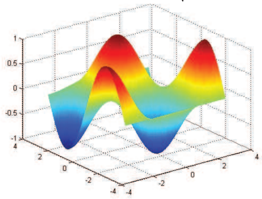


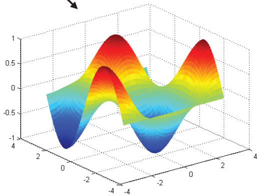
# Surf Options

- See **help surf** for more options
- There are three types of surface shading
  - » `shading faceted`
  - » `shading flat`
  - » `shading interp`
- You can change colormaps
  - » `colormap(gray)`



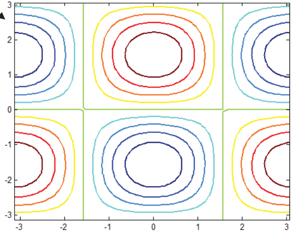



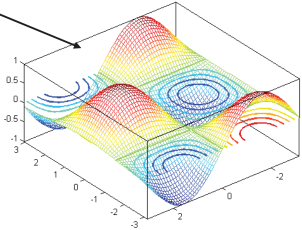



# Contour

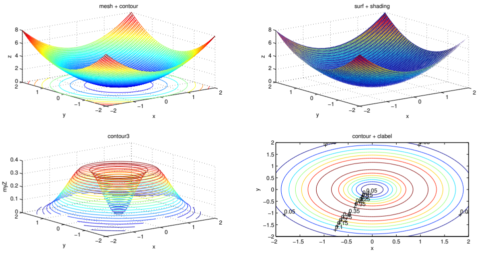
- You can make surfaces two-dimensional by using contour
  - » `contour(X,Y,Z,'LineWidth',2)`
    - ➢ takes same arguments as surf
    - ➢ color indicates height
    - ➢ can modify linestyle properties
    - ➢ can set colormap
  - » `hold on`
  - » `mesh(X,Y,Z)`



# Continue...

```
>> x = linspace(-2,2); y = linspace(-2,2,50);
>> [X, Y] = meshgrid(x,y);
>> z = X.^2 + Y.^2;
>> subplot(2,2,1), mesh(x,y,z), xlabel('x'), ylabel('y')
>> zlabel('z'), hold on, contour(x,y,z), title('mesh + contour')
>> subplot(2,2,2), surf(x,y,z), xlabel('x'), ylabel('y')
>> zlabel('z'), shading interp, title('surf + shading')
>> myZ = z .* exp(-z);
>> subplot(2,2,3), contour3(x,y,myZ,20), xlabel('x'), ylabel('y')
>> zlabel('myZ'), title('contour3')
>> subplot(2,2,4), H = contour(x,y,myZ); xlabel('x'), ylabel('y')
>> title('contour + clabel'), clabel(H)
```



# Symbolic Math Toolbox

- Numeric approach:
  - Always get a solution
  - Can make solutions accurate
  - Easy to code
  - Hard to extract deeper understanding
  - Numerical methods sometimes fail
  - Can take a while to compute
- Symbolic approach:
  - Analytical Solutions
  - Lets you perceive things about solution form
  - Sometimes can not be solved
  - Can be overly complicated

# Symbolic Variables

- Symbolic variables are a type, like double or char
- To make symbolic variables, use sym

```
>> a=sym('1/3');
>> mat=sym([ 1 2;3 4]);
```

- Or use syms

```
>> syms a b c d
>> A = [a^2, b, c ; d*b, c-a, sqrt(b)]
A = [ a^2,     b,       c]
[ b*d, c - a, b^(1/2)]
>> b = [a;b;c];
>> A*b
ans =    a^3 + b^2 + c^2
b^(1/2)*c - b*(a - c) + a*b*d
```

# Arithmetic, Relational, and Logical Operators

- Arithmetic Operations
  - ceil, floor, fix, cumprod, cumsum, real, imag, minus, mod, plus, quorem , round
- Relational Operations
  - eq, ge, gt, le, lt, ne, isequaln
- Logical Operations
  - and, not, or, xor, all, any, isequaln, isfinite, isinf, isnan, logical

See http://www.mathworks.com/help/symbolic/operators.html for more details

# Symbolic Expressions

- expand    multiplies out
- factor    factors the expression
- inv    computes inverse
- det    computes determinant

```
>> syms a b
>> expand((a-b)^2)
ans = a^2 - 2*a*b + b^2
>> factor(ans)
ans = (a - b)^2
>> d=[a, b; 0.5*b a];
>> inv(d)
ans =
[ (2*a)/(2*a^2 - b^2), -(2*b)/(2*a^2 - b^2)]
[    -b/(2*a^2 - b^2),  (2*a)/(2*a^2 - b^2)]
>> det(d)
ans = a^2 - b^2/2
```

- pretty makes it look nicer
- collect collect terms
- simplify simplifies expressions
- subs replaces variables with number or expressions
- solve replaces variables with number or expressions

```
>> g = 3*a +4*b-1/3*a^2-a+3/2*b;
>> collect(g)
ans =
(11*b)/2 + 2*a - a^2/3
>> subs(g,[a,b],[0,1])
ans = 5.5000
```

# Symbolic Integration/Derivation

- Differentiation: diff(function,variable,degree)
- Integration: int(function,variable,degree,option)

```
>> syms x y t
>> f=exp(t)*(x^2-x*y +y^3);
>> fx=diff(f,x)
fx = exp(t)*(2*x - y)
>> fy=diff(f,y,2)
fy = 6*y*exp(t)
>> int(f,y)
ans = (y*exp(t)*(4*x^2 - 2*x*y + y^3))/4
>> int(f,y,0,1)
ans = (exp(t)*(4*x^2 - 2*x + 1))/4
```

# Symbolic Summations/Limits

- Summation: symsum
- Limit: limit
- Taylor seris: taylor

```
>> syms x k
>> s1 = symsum(1/k^2,1,inf)
s1 = pi^2/6
>> s2 = symsum(x^k,k,0,inf)
s2 = piecewise([1 <= x, Inf], [abs(x) < 1, -1/(x - 1)])
>> limit(x / x^2, inf)
ans = 0
>> limit(sin(x) / x)
ans = 1
>> f = taylor(log(1+x))
f = x^5/5 - x^4/4 + x^3/3 - x^2/2 + x
```

# Calculus Commands

| Command | Description |
|---|---|
| diff | Differentiate symbolic |
| int | Definite and indefinite integrals |
| rsums | Riemann sums |
| curl | Curl of vector field |
| divergence | Divergence of vector field |
| gradient | Gradient vector of scalar function |
| hessian | Hessian matrix of scalar function |
| jacobian | Jacobian matrix |
| laplacian | Laplacian of scalar function |
| potential | Potential of vector field |
| taylor | Taylor series expansion |
| limit | Compute limit of symbolic expression |
| fourier | Fourier transform |
| ifourier | Inverse Fourier transform |
| ilaplace | Inverse Laplace transform |

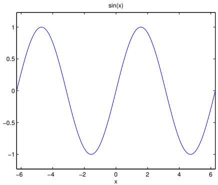
# Linear Algebra Commands

| Command | Description |
|---|---|
| adjoint | Adjoint of symbolic square matrix |
| expm | Matrix exponential |
| sqrtm | Matrix square root |
| cond | Condition number of symbolic matrix |
| det | Compute determinant of symbolic matrix |
| norm | Norm of matrix or vector |
| colspace | Column space of matrix |
| null | Form basis for null space of matrix |
| rank | Compute rank of symbolic matrix |
| rref | Compute reduced row echelon form |
| eig | Symbolic eigenvalue decomposition |
| jordon | Jordan form of symbolic matrix |
| lu | Symbolic LU decomposition |
| qr | Symbolic QR decomposition |
| svd | Symbolic singular value decomposition |

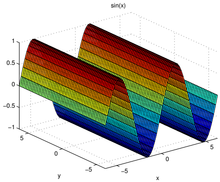
# Symbolic Plotting

- Plot a symbolic function over one variable by using the figures/ezplot function

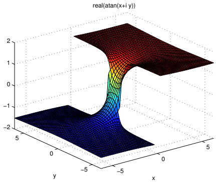
```
>> syms x
>> y = sin(x);
>> ezplot(y);
>> f = sin(x);
>> ezsurf(f);
>> ezsurf( 'real(atan(x+i*y))' );
```




(c) ezplot(y)




(d) ezsurf(f)




(e) ezsurf( 'real(atan(x+i*y))' )

# End of Lecture

1. Control Loops
2. Advanced Data Structures
3. Graphics
4. Symbolic Toolbox

# Exercises I

- In order to get and save current date and time, write a script by following steps:
  - Create a variable **start** using the function **clock**
  - What is the size of **start**?
  - What does **start** contain? See help **clock**
  - Convert the vector **start** to a string. Use the function **datestr** and name the new variable **startString**
  - Save **start** and **startString** into a mat file named **startTime**

# Exercises II

- If $A$ is a square matrix (i.e. of dimension $n \times n$), the matrices $\cos(A)$ and $\sin(A)$ can defined by the formulas

$$\cos(A) = \sum_{k=0}^{\infty} (-1)^k \frac{A^{2k}}{2k!}, \qquad \sin(A) = \sum_{k=0}^{\infty} (-1)^k \frac{A^{2k+1}}{(2k+1)!},$$

respectively. The partial sums

$$C_N(A) = \sum_{k=0}^{N-1} (-1)^k \frac{A^{2k}}{2k!} \qquad S_N(A) = \sum_{k=0}^{N-1} (-1)^k \frac{A^{2k+1}}{(2k+1)!}$$

can thus be used to approximate the matrices $\cos(A)$ and $\sin(A)$.

  - Write a function whose inputs are a square matrix $A$ and a tolerance number (**TOL**), and whose ouputs are the matrices $\cos(A)$ and $\sin(A)$. The outputs should be obtained by using Matlab to compute the sequences $C_N(A)$, and $S_N(A)$, $N = 1, 2, \ldots$ and stopping when the maximum of the absolute values of the entries of the matrix $C_{N+1}(A) - C_N(A)$ and $S_{N+1}(A) - S_N(A)$ is less than **TOL**. (Note that $\cos(A)$ and $\sin(A)$ is NOT the matrix obtained by computing the cosine of the individual entries of the matrix) (Hint: Use the **while** loop as well as the command **max**.)
  - Let

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

where $a_{11}$, $a_{12}$, $a_{21}$ $a_{22}$ are the last 4 digits of your student number. Use the above function to compute $\cos(A)$ and $\sin(A)$. Save your answers in the variables Answer1 and Answer2, respectively. Use Matlab to compute the matrix $(\cos(A))^2 + (\sin(A))^2$. Save your answer in the variables Answer3.

# Exercises III

- Write a function whose input is a positive integer and whose outputs a matrix and a vector such that $A = (a_{ij})$, where $a_{ij} = i/j$ and $x_j = j$, respectively. Display a warning message if $n$ is nonpositive by using **fprintf** command.

# Exercises IV

- Write a function to compute the factorial value of a single scalar argument. This function should have the following components:
  - An **if** statement which returns an error message if the argument is negative by using **disp** command.
  - An **elseif** statement which returns an error message if the argument is not an integer. You should use either the built-in **round**, **floor** or **ceil** functions to test for non-integers.
  - An **else** statement with an embedded **for** loop that does the actual factorial calculation. Make sure that your function is able to handle any non-negative integer, including 0.

# Exercises V

- Write a script to generate a figure with a $1 \times 2$ array of windows. In one window draw a loglog plot of the function $C(\omega) = \frac{1}{\sqrt{1+\omega^2}}$ for $10^{-2} \leq \omega \leq 10^{-3}$, and in the other window draw a plot of $C(\omega)$ with the horizontal axis scaled logarithmically and the vertical axis scaled linearly. Be sure to label the axes and title the plot.
- Write a script to graph the surface given by $z = x^2 - y^2$ for $-3 \leq x \leq 3$, $-3 \leq y \leq 3$ on a $2 \times 2$ array of windows. Please use the following formatting instructions:
    - Draw with **shaded faceted** in the $(1, 1)$ position
    - Draw with **shaded interp** in the $(1, 2)$ position
    - Draw contour of surface in 3D in the $(2, 1)$ position
    - Draw contour of surface in 2D in the $(2, 2)$ position
    - Be sure to label your axes and title the plot.

# Exercise VI

Write a function `newton(f,fprime,x0,tol)` that implements Newton's iteration for rootfinding on a scalar function:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

The first two inputs are handles to functions computing $f$ and $f'$, and the third input is an initial root estimate. Continue the iteration until either $|f(x_{n+1})|$ or $|x_{n+1} - x_n|$ is less than `tol`.

# For More Information

- http://iam.metu.edu.tr/scientific-computing
- https://iam.metu.edu.tr/scientific-computing-lecture-series
- https://www.facebook.com/SCiamMETU/

**...thank you for your attention !**