What’s MATLAB?

- MAT(rix) LAB(oratory)
- It’s an analysis and computation tool...
- ...not a platform for embedded code.
- BUT, good MATLAB models can be converted to C
- MATLAB is a big programmable calculator designed for engineering and analysis use
- Students can buy a largely functional version for $99
- ...which is a steal, considering that my corporate version with fewer features was $3500!
- MATLAB is so uniformly useful that it’s installed on most (if not all) UMBC computers
- The MATLAB environment is the same, regardless of what platform you’re using!
Let’s get started

- On Windows or Mac, find the MATLAB icon, which looks something like this

- On Linux start at shell or kommand window and type
  - > matlab

Which takes you to the environment

![MATLAB environment screenshot](image)
It works like a calculator

- Command window
- Type commands in this window
- MATLAB is interpreted,
- ...which means the commands execute immediately

- It knows about $\pi$ and $e$ and $\sqrt{-1} = i$ (or $\sqrt{-1} = j$)

- Variable names must start with a letter and can be 63 characters long.
- Let’s do some simple things

Matrices

- Matrices are rectangular arrays of numbers.
- We talk about the dimensions of a matrix by
  - Number of rows x number of columns
- The simplest matrix is an array, which has dimensions $1 \times n$ or $n \times 1$
- Think of it as being a group of bins, arranged either in a row or a column, each with it’s own address or index

\[
\begin{pmatrix}
1 \\
2 \\
3 \\
4 \\
\end{pmatrix} \quad \begin{pmatrix}
1 & 2 & 3 & 4 \\
\end{pmatrix} \quad 1 \times 4 \\
4 \times 1
\]
Filling up our arrays

• Several methods to fill the arrays
  • 1 element at a time
  • The function transpose
  • Turns a row into a column
  • And vice versa
  • Commas delimit row elements
  • Semicolons delimit column elements

> a(1)=1
a =
    1
> a(2)=2;
> a(3)=3;
> a(4)=4;
> a =
    1     2     3     4
  ans =
    1
    2
    3
    4

1×4

Or, MATLAB lets us fill the arrays by enumeration

> a=[1,2,3,4]
a =
    1     2     3     4
> a=[1;2;3;4]
a =
    1
    2
    3
    4
  1×4

Or, we might fill the arrays by means of a loop

> for k=1:4
a(k)=k;
end; % need an end statement for every loop
> a
a =
    1     2     3     4
Why make such a big deal?

- Because the matrix (or array) is the basic building block in MATLAB...
- ... and there are thousands of built in functions that operate on matrices directly.
- For example

```matlab
>> % Square each of the values in a
>> asq=a.^2; % note the .^ notation!
>> asq
asq =
    1  4  9  16
>> % Take the square root of the values in a
>> roota=sqrt(a);
>> roota
roota =
   1.0000    1.4142    1.7321    2.0000
```

How about summing and whatnot?

- You can always sum with a loop

```matlab
% test iterations with vectors
N=100000;  %100,000 random numbers
Ntrials=10000;  %10,000 trials
x=rand(1,N);  %100,000 random numbers
%
% Looped ops
%
disp('Looped ops');
xsum=0;
tic
for k=1:Ntrials
    for m=1:N
        xsum=xsum+x(m);
    end;
end;
toc
```

- But this is horrendously inefficient!
Or we can use MATLAB’s vector operators to do the same thing

```matlab
% Vector ops
% disp('Vector sum');
tic
for k=1:Ntrials
    xsumv=sum(x);
end;
toc
Vector sum
Elapsed time is 0.351695 seconds.
Looped ops
Elapsed time is 3.797221 seconds.

A factor of 11 more efficient
```

So matrix (array) operations are much faster

- Includes all matrix/array operations

```matlab
>> x=rand(1,4)
x =
    0.6648    0.5894    0.3663    0.0773
>> y=sin(x) % MATLAB assumes radians
y =
    0.6169    0.5559    0.3581    0.0772
>> y=exp(x) % Using "e"
y =
    1.9440    1.8030    1.4423    1.0804
>> y=10 .^(x) % using .^y =
    4.6213    3.8855    2.3241    1.1948
>> y=cos(x).^2
y =
    0.6195    0.6910    0.8717    0.9940
>> z=cos(x).^2 + sin(x).^2
z =
    1.0000    1.0000    1.0000    1.0000
```
Plotting!!

- MATLAB makes plotting and data analysis EASY
- Simple plotting steps
  - Create an x array (can be called anything)
  - Create a y array (can be called anything)
  - Plot
  - Label
  - Customize

>> x=[0:1:1000]/250; % an array of 1000 points
>> y1=sin(2*pi*x);
>> y2=cos(2*pi*x);
>> plot(x,y1);
>> xlabel('Angle/(2\pi)')
>> ylabel('Amplitude');
>> figure(1)
>> title('My First Plot');
>>
• **Make it easier to see**

```matlab
>> plot(x,y1,'LineWidth',2);
>> xlabel('Angle/(2\pi)','FontSize',12);
>> ylabel('Amplitude','FontSize',12);
>> title('My Second Plot','FontSize',12);
>> grid on
>> figure(1)
```

![My Second Plot](image)

• **Plot more things**

```matlab
>> plot(x,y1,'r',x,y2,'b:','LineWidth',2)
>> xlabel('Angle/(2\pi)','FontSize',12);
>> title('My Third Plot','FontSize',12);
>> legend('\sin(x)', '\cos(x)')
>> grid on
>> figure(1)
```

![My Third Plot](image)
Last time

- We learned how to get on MATLAB...
- ...and to do some simple things.
- MATLAB loves to work in vectors, that is $1 \times n$ (row) or $n \times 1$ column matrices
- We talked about how MATLAB uses vectors as inputs for most function
- We talked about some simple plotting things.

- And I forgot to assign homework!

- So fire up your MATLAB and away we go.

Today

- We’re going to talk about “multiplying” vectors,
- ...and, from that, multiply matrices...
- And what that is good for
- Matrix equations are VERY common in engineering and computer science applications...
- Such as
  - Statics, dynamics, vibrations (ENME)
  - Heat flow, diffusion, decay (CBEE)
  - Circuits, linear systems, and signal processing (CMPE)
  - Graphics, rotations, and scaling (CMSC)
Adding vectors

- MATLAB is really straightforward about adding and subtracting vectors.

```matlab
>> x=[0 3 5 7]; y=[2 -1 6 3];
>> z=x+y
z =
    2   2  11  10
>> w=x-y
w =
   -2    4   -1    4
>> zt=x'+y'
zt =
    2
     2
    11
    10
>> wt=x'-y'
wt =
     -2
      4
     -1
      4
```

- To add or subtract, vectors must be the same size.

What about multiplication?

- We need to define exactly what we mean.
- Element by element multiplication.
  - Use the `.*` operator.

```matlab
>> x.*y
ans =
     0   -3   30   21
>> [x;y;x.*y]
ans =
     0    3    5    7    x
     2   -1    6    3    y
     0   -3   30   21    x.*y
```
Matrix multiplication

- To multiply matrices or vectors, we need the "inner dimensions" to be the same
  - So
    \[ x_{(4 \times 1)} y_{(3 \times 1)} \]
    cannot be performed, because \( 4 \neq 3 \)
    \[ y_{(3 \times 1)} x_{(4 \times 1)} \]
    can be performed, because \( 1=1 \), the result is \( 3 \times 4 \)!
- Let's restrict ourselves to vectors with the same dimensions
  - \( x_{(4 \times 1)} y_{(4 \times 1)} \)
    cannot be performed, because \( 4 \neq 1 \)
  - \( x_{(4 \times 1)} y_{(1 \times 4)}^T \)
    can be performed, because \( 4=4 \), the answer is \( 1 \times 1 \)
- So the answer is a scalar (1 x 1)! How does this work

Inner product or dot product

- The inner product of two vectors \( x_{n \times 1} y_{n \times 1} \)
  or \( x \cdot y \) is defined to be

\[
x = [x_1, x_2, ..., x_n], y = [y_1, y_2, ..., y_n]^T
\]

\[
xy = x \cdot y = x_1 y_1 + x_2 y_2 + ... + x_n y_n = \sum_{k=1}^{n} x_k y_k
\]

- MATLAB knows all about inner and dot products

\[
\text{>> } x*y'
\]
\[
\text{ans = 48}
\]
\[
\text{>> } \text{dot}(x,y)
\]
\[
\text{ans = 48}
\]
\[
\text{>> } \text{sum}(x.*y)
\]
\[
\text{ans = 48}
\]
Who cares?

- Well, consider the Pythagorean theorem (!?)
  \[ a^2 + b^2 = c^2 = axa + bxb = [a,b] \times [a,b] \]

  \[
  \begin{align*}
  & \text{>> a}^2+b^2 \\
  & \text{ans } = \\
  & 25 \\
  & \text{>> dot(} [a,b],[a,b] \text{)} \\
  & \text{ans } = \\
  & 25 \\
  & \text{>> } [a,b]*[a,b]'
  \\
  & \text{ans } = \\
  & 25 \\
  & \text{>> sum(} [a,b].*[a,b] \text{)}
  \\
  & \text{ans } = \\
  & 25
  \end{align*}
  \]

- ...and this generalizes to longer vectors
  \[
  \| x \|^2 = x_1^2 + x_2^2 + \ldots + x_n^2 = \mathbf{x} \cdot \mathbf{x}
  \]

What about larger matrices?

- Addition is still term-by-term, and requires that the matrices be the same size

  \[
  \begin{align*}
  & \text{>> x} = [1, 2, 3; 4, 6, 10] \\
  & \text{>> y} = [3, -5; 2, 0] \\
  & \text{>> z} = [1, 0, 0; 0, 1, 1]
  \\
  & \text{>> x}+\text{z} \\
  & \text{ans } = \\
  & 2 \ 2 \ 3 \\
  & 4 \ 7 \ 11
  \end{align*}
  \]

  \[
  \begin{align*}
  & \text{>> x}+\text{y} \\
  & \text{Error using +} \\
  & \text{Matrix dimensions must agree.}
  \\
  & \text{>> x}+\text{z}
  \\
  & \text{ans } = \\
  & 2 \ 2 \ 3 \\
  & 4 \ 7 \ 11
  \end{align*}
  \]
What about larger matrices

- Multiplication is more complicated
- We basically view an \( n \times m \) matrix as either
  - An array of \( n \times 1 \) row vectors, or
  - An array of \( m \times 1 \) column vectors
- The inner dimension rule still holds
- We do \( n \times m \) inner products, one for each row and column...
- ...resulting in \( n \times m \) numbers in a new \( n \times m \) matrix
- This is all very complicated, so we need an example

```
>> x=[1, 2, 3;4 6 10]
x =
   1     2     3
   4     6    10
>> y=[3 -5;2 0]
y =
   3    -5
   2     0
>> y*x
ans =
   -17   -24   -41
   2     4     6
>> x'*y
ans =
  11    -5
  18   -10
  29   -15
```

The inner dimensions must agree or we can’t do the multiplication!
So why do we care about matrix multiplication?

- Two reasons!
- 1) We can use it to solve systems of equations
  - ...you remember, 7th or 8th grade algebra? Maybe even MATH106?
- 2) We can use it to rotate position vectors
  - ...which is absolutely essential in visualization and gaming and whatnot
- We need 2 more concepts! A matrix identity and matrix inversion

The identity matrix

- Can I find a matrix, \( B \) such that \( AB = A \)
- Yes, provided that
  - \( A \) is \( n \times n \), in otherwords, square
  - \( A \) is invertible
- These are rich subjects, and I can’t do them justice today
- MATLAB, of course, knows all of this

\[
A = \begin{pmatrix}
1 & 2 \\
-4 & -8
\end{pmatrix}
\]

\[
>> \text{inv}(A)
\]

Warning: Matrix is singular to working precision.

\[
\text{ans} =
\begin{pmatrix}
\text{Inf} & \text{Inf} \\
\text{Inf} & \text{Inf}
\end{pmatrix}
\]
If the matrix is square and invertible, then we have the identity
\[ AA^{-1} = A^{-1} A = I \]

Where \( I \) is called the identity matrix, and
\[ AI = IA = A \]

```matlab
>> A=[1 2;-2 1]
A =
    1     2
    -2     1
>> Ainv=inv(A)
Ainv =
   0.2000  -0.4000
   0.4000   0.2000
>> A*Ainv
ans =
   1     0
   0     1
>> Ainv*A
ans =
   1     0
   0     1
>> eye(2)
ans =
   1     0
   0     1
```

Programming

- The MATLAB environment allows you to create programs and functions...
- ...that then become part of the environment.
- Open the editor by clicking on the editor icon
- In the editor, enter your MATLAB commands
- Save the file with a meaningful name
- Then enter file name at the command prompt.
Let’s do something fun!

- Write a script to rotate a rectangle
- We draw the rectangle with x and y vectors
- We do the rotation by means of a matrix multiplication
- We do plots!