Recap

- Using the command window interactively
- Variables: Assignment, Identifier rules, Workspace, command `who` and `whos`
- Setting the output format using the command `format`
- Basic computations: `help elfun` gives the list of elementary math functions
Matlab is short for “Matrix Laboratory”.

Vectors and Matrices in Matlab store values of the same type only.
Creating Row Vectors

- Comma or space separated values between brackets

```matlab
1 EDU>> v = [ 1 2 3 4]
2 v =
3 1 2 3 4
4 EDU>> v = [1, 2, 3, 4]
5 v =
6 1 2 3 4
```

- The colon operator

```matlab
1 EDU>> v = 1:5
2 v =
3 1 2 3 4 5
4 EDU>> v = 1:2:5 % step size of 2
5 v =
6 1 3 5
```
Creating Row Vectors

▶ `linspace` function

```
EDU>> linspace(1,5,3)
ans =
1     3     5
```

▶ Concatenating row vectors

```
EDU>> v1 = 1:2:5;
EDU>> v2 = linspace(10, 14, 5);
EDU>> v = [v1 v2]
v =
1     3     5    10    11    12 ...
     13    14
```
Creating Column Vectors

- Semicolon-separated values between brackets

```
1 EDU>> v = [ 1; 2; 3; 4]
2 v =
3  1
4  2
5  3
6  4
```

- Colon/linspace create row vectors, need to transpose:

```
1 EDU>> v = 1:2:5;
2 EDU>> v = v' 
3 v =
4  1
5  3
6  5
```
Referring to and Modifying Elements in Vectors

1 2 3 4 5 6 7 8 9 10

1 3 5 7 9 3 6 9 12 15

- Accessing a single element

```matlab
EDU>> v(5)
ans =
    9
```

- Accessing a subset of a vector using the colon operator

```matlab
EDU>> v(4:6)
ans =
    7 9 3
```
Referring to and Modifying Elements in Vectors

1 2 3 4 5 6 7 8 9 10
1 3 5 7 9 3 6 9 12 15

- Accessing a subset of a vector using an index vector

1 EDU>> v([1 5 10])
2 ans =
3    1 9 15
4 % Here [1 5 10] is the index vector

- Logical indexing

1 EDU>> v(v<10)
2 ans =
3    1 3 5 7 9 3 ...
   6 9
Creating Matrices

- Explicitly typing values

```
1  EDU>> mat = [ 4 3 1; 2 5 6]
2  mat =
3      4   3   1
4      2   5   6
5  EDU>> mat2 = [2:4 ; 1:2:5]
6  mat2 =
7      2   3   4
8      1   3   5
```

```
1  EDU>> mat = [ 4 3 1; 2 5 ]
2  ??? Error using ==> vertcat
3  CAT arguments dimensions are not consistent.
```

There must always be the same number of values in each row.
Creating Matrices

- **Special functions**: zeros, ones, eye, rand

```
1 EDU>> zeros(2,3)
2 ans =
3     0 0 0
4     0 0 0
5 EDU>> zeros()
6 ans =
7     0 0
8     0 0
9 EDU>> ones(2,3)
10 ans =
11     1 1 1
12     1 1 1
13 EDU>> eye(2)
14 ans =
15     1 0
16     0 1
```
Creating Matrices

- Constructing matrices in block form

```
1 EDU>> B = [ 1 2 ; 3 4];
2 EDU>> C = [B zeros(2); ones(2) eye(2)]
3 C =
4     1  2  0  0
5     3  4  0  0
6     1  1  1  0
7     1  1  0  1
```

- “tiled” block matrices using `repmat`.

```
1 EDU>> A = repmat(eye(2), 2, 1)
2 A =
3     1  0
4     0  1
5     1  0
6     0  1
```
Creating Matrices

- Creating diagonal matrices: `diag`, `blkdiag`

```matlab
1 EDU>> diag([2,3,4])
2 ans =
3   2   0   0
4   0   3   0
5   0   0   4
6 EDU>> A = blkdiag(2*eye(2), ones(2))
7 A =
8   2   0   0   0
9   0   2   0   0
10  0   0   1   1
11  0   0   1   1
```

- Matlab provides a number of special matrices (`magic`, `hadamard`, ...).
Referring to and Modifying Elements in Matrices

\[
\begin{pmatrix}
1 & 2 & 3 \\
11 & 12 & 13 \\
21 & 22 & 23 \\
31 & 32 & 33 \\
\end{pmatrix}
\]

Accessing an element

1. EDU>> A = [11 12 13; 21 22 23; 31 32 33]
2. EDU>> A(2,3)
3. ans =
4. 23
Referring to and Modifying Elements in Matrices

\[
\begin{bmatrix}
1 & 2 & 3 \\
11 & 12 & 13 \\
21 & 22 & 23 \\
31 & 32 & 33 \\
\end{bmatrix}
\]

- Accessing a row/column

```
1 EDU>> A(2, :)
2 ans =
3  21  22  23
4 EDU>> A(:,2)
5 ans =
6   12
7   22
8   32
```
Referring to and Modifying Elements in Matrices

\[
\begin{pmatrix}
1 & 2 & 3 \\
1 & 11 & 12 & 13 \\
2 & 21 & 22 & 23 \\
3 & 31 & 32 & 33
\end{pmatrix}
\]

- Accessing a block

```
1  EDU>> A(1:2, 2:3)
2  ans =
3     12   13
4     22   23
5  EDU>> A(:, 2:3)
6  ans =
7     12   13
8     22   23
9     32   33
```
Exercises

- Using the colon operator, create the following vectors:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>1.0000</td>
<td>1.5000</td>
<td>2.0000</td>
<td>2.5000</td>
</tr>
<tr>
<td></td>
<td>3.0000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

- Using the `linspace` function, create the following vectors:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-3</td>
<td>-6</td>
<td>-9</td>
<td>-12</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>7</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Exercises

- Create a $4 \times 2$ matrix of all zeros and store it in a variable. Then replace the second row in the matrix with a 3 and a 6.
- Create a $3 \times 5$ matrix of random integers, each in the range from -5 to 5; store it in a variable. Create another matrix that stores for each element the absolute value of the corresponding element in the original matrix. Delete the the third row.
- Create a vector $x = [1 \ 2 \ 3]$. Expand the vector $x$ to have length 6 and assign $x(6) = 4$.
- Create a random $4 \times 1$ vector and get the index of the largest element.
Data Types

Strings and Cell Arrays

- Strings are vectors of characters
  - Denoted by ‘ ’
  - Ex: str = 'hello'

- Cell arrays generalize matrices to allow for arbitrary entries, not just numbers
  - Denoted by {}
  - Ex: cll = {’hello’, 123, 456, ’goodbye’}
# Elementary Matrix and Array Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Matrix sense</th>
<th>Element wise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
<td>.*</td>
</tr>
<tr>
<td>Left Division</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>Right Division</td>
<td>/</td>
<td>./</td>
</tr>
<tr>
<td>Exponentiation</td>
<td>^</td>
<td>.^</td>
</tr>
</tbody>
</table>

Elementary matrix and array operations

<table>
<thead>
<tr>
<th>Matlab notation</th>
<th>Mathematical Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Right division: a/b</td>
<td>a/b</td>
</tr>
<tr>
<td>Left division: a\b</td>
<td>\b/a</td>
</tr>
</tbody>
</table>
Matrix and Array Operations

- dot, cross product of two vectors

```matlab
1 EDU>> i = [ 1 0 0 ]; j = [ 0 1 0 ];
2 EDU>> dot(i,j)
3 ans =
4    0
5 EDU>> cross(i,j)
6 ans =
7    0    0    1
```
Matrix and Array Operations

- Transpose of a matrix/vector

```matlab
1 EDU>> A = [ 1 2 3; 4 5 6]
2 A =
3     1  2  3
4     4  5  6
5 EDU>> A'
6 ans =
7     1  4
8     2  5
9     3  6
10 EDU>> x = [1 2]
11 x =
12     1  2
13 EDU>> x'
14 ans =
15     1
16     2
```
Most functions which act on a scalar can be given a matrix argument, in which case the functions are computed element wise.

```
EDU>> A = [ 2 2 ; 0 2]
A =
 2 2
0 2
EDU>> sqrt(A)
ans =
1.4142 1.4142
0 1.4142
```
Functions of a matrix in linear algebra sense are signified by names ending in m: \texttt{expm, funm, logm, sqrtm}

```
1 EDU>> B = sqrtm(A)
2 B =
3    1.4142   0.7071
4    0   1.4142
5 EDU>> B*B
6 ans =
7    2.0000   2.0000
8    0   2.0000
```
Introduction to Matlab programming

Simple I/O

- **input** function: used as an assignment statement.

```matlab
1 EDU>> length = input('Enter the length: ')    
2 Enter the length: 3    
3 length =    
4 3
```

If a character or string input is desired, 's' must be added after the prompt.

```matlab
1 EDU>> name = input('Enter your name: ')    
2 Enter your name: Brett    
3 ??? Error using ==> input    
4 Undefined function or variable 'Brett'.
```
Introduction to Matlab programming

Simple I/O

EDU>> name = input('Enter your name: ', 's')
Enter your name: Brett
name = Brett

Output Statements: disp and fprintf

- disp: Displays the result of an expression or a string.

EDU>> disp('Hello')
Hello
EDU>> disp(3*7)
21
EDU>> x = 5; y = 2;
EDU>> disp(x*y)
10
**Introduction to Matlab programming**

**Simple I/O**

`fprintf`: Prints formatted output to the screen.

```
1  EDU>> fprintf('The value is %d, for ... sure! \n', 3^2);
2  The value is 9, for sure!
```

The string passed to the `fprintf` function is called the format string.
Introduction to Matlab programming
Simple I/O

1  EDU>> fprintf('The value is %d, for sure! ...
\n', 3^2);
2  The value is 9, for sure!

In the above block, %d is an example of a *place holder*. The character in the placeholder is called the *conversion character*.

%d  decimal integers
%f  floats
%e  exponential notation
%c  single characters
%s  strings

List of simple place holders
Introduction to Matlab programming

Simple I/O

```
1  EDU>> fprintf('%6.3f\n', pi)
2    3.142
3  EDU>> fprintf('%6.3e\n', pi)
4    3.142e+00
```

In the above example, `%` character denotes the start of a format specifier requesting a field of width 6 with 3 digits after the decimal point and `\n` denotes a new line. For a negative number, a minus sign occupies one position of the field width.
Introduction to Matlab programming

Simple I/O

- A minus sign just after the % character causes the field to be left justified.

```plaintext
EDU>> fprintf('%5.0f
%5.0f
', 9, 103)
9
103
EDU>> fprintf('%-5.0f
%-5.0f
', 9, 103)
9
103
```
Introduction to Matlab programming
Simple I/O

What if more numbers are supplied to be printed than there are format specifiers?

```
1   EDU>> fprintf('%-5.0f
%-5.0f
', 9, ...
              103, 11)
2       9
3      103
4      11
```

The format specifiers are simply reused. If a matrix is provided, elements are taken down the first column, then the second column and so on. This feature can be used to avoid loops.
Create the following output using not more than a single `fprintf` statement:

1. 30 miles/hour = 48 kilometers/hour
2. 40 miles/hour = 64 kilometers/hour
3. 60 miles/hour = 96 kilometers/hour
4. 70 miles/hour = 112 kilometers/hour