Week 10
Matlab: Vectorization. (No loops, please!)
Vectorisation

- Matlab is designed to work with vectors and matrices efficiently.
- Many calculations that required loops in many programming languages (e.g. OpenOffice Basic) can be done without any loops in Matlab.
  - This is called vectorisation.
Comparing three methods

• We want to compute the cube roots of all the elements in the vector \( v \) with 1,000,001 elements

\[
v = 0 : 0.1 : 100000
\]

• We will compare 3 methods to do this
  - All three methods are correct but they take different amount of time to complete the work
Method 2: for-loop, allocate storage

- The code is:

```matlab
cubeRoot2 = zeros(size(v)); % allocate storage for the answer
for idx = 1:length(v)
    cubeRoot2(idx) = v(idx) ^ (1/3);
end
```

- The first statement tells Matlab to reserve memory space to store 1,000,001 numbers.
- Key feature of this method: Allocate memory space to store the answer.
Method 1: for-loop; does not allocate storage

- The method is the same as method 2 except it does not allocate storage space in the beginning. The code is:

```plaintext
for idx = idx = 1:length(v)
    cubeRoot1(idx) = v(idx) ^ (1/3);
End
```

- The length of the array `cubeRoot1` is increased by one per iteration.
  - 1\textsuperscript{st} iteration: `cubeRoot1` has a length of 1
  - 2\textsuperscript{nd} iteration: `cubeRoot1` has a length of 2
  - 3\textsuperscript{rd} iteration: `cubeRoot1` has a length of 3
  - etc.
Method 3: using array operation .^ 

- The code is:

```matlab
cubeRoot3 = v.^ (1/3);
```

- Recall there are two aspects to algorithms
  - Correctness and efficiency
- All 3 methods are correct
- But they have different efficiency
Compare the three methods

- The program is in the file `compareMethods.m`
- It uses Matlab built-in functions `clock` and `etime` to determine how long each method takes

```matlab
timeStart = clock;  % get the time before the computation starts

% code block that you want to measure the computation time

etimeNeeded = etime(clock, timeStart);
% time elasped between current time (given by clock) and the
% start time (stored in timeStart)
% etime is short for elapsed time
```
Lesson 1: Allocate storage

- For faster computation, initialise the storage space for arrays
  - This is especially important for big matrices
  - You can do that by initializing a zero matrix at first and then fill in the elements later on, e.g.

```matlab
cubeRoot2 = zeros(size(v)); % allocate storage for the answer
```

- If you don’t know what the size of the matrix should be, do an over-estimation and reduce the size of it later
  - This is what you do in Lab10
Why is it slow if you don’t allocate storage?

- The following gives you 1:100 but is slow

```matlab
v = [];  
for k = 1:100  
    v = [v k]; % Length of vector v increased by 1 at each iteration  
end
```

- For explanation, consider \( v = [1 \ 2 \ 3] \)
- To add a 4 to \( v \), Matlab needs to

  1) Find a memory space to store a vector with 4 elements

  2) Copy \([1 \ 2 \ 3]\) to new memory location

  3) Add 4

Time is wasted on these processes!
Lesson 2 + one other tip

- Array operation is faster than for-loops. Use array operation whenever possible.

- If there is a built-in function that does what you want, use it because it is normally faster
  - Many useful functions use vectorisation
  - You can find a list of functions that commonly used in vectorisation here:
Some useful functions

- `sum`
- `cumsum`: cumulative sum
- `std`: standard deviation
- `var`: variance
- `prod`: product

- `mean`
- `median`
- `max, min`
- `mode`: most frequently occurred value
- `sort`

Note:
- These functions work differently for vectors and matrices.
  - The idea is similar, so we use `sum` as an illustration
  - Some functions can take additional inputs or return more than one output. Check the documentation for details.
The sum function (1)

- \( \text{sum(a one dimensional array)} \) is a scalar

```matlab
v1 = [1 2 3]; % row vector
v2 = [-1 ; -2 ; 4]; % column vector

>> sum(v1)
    6

>> sum(v2)
    1
```
The sum function (2)

\[ m = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}; \quad \% \text{ a 2-by-3 array} \]

\[
\begin{align*}
>> \text{sum}(m,1) & \% \text{ sum in 1st dimension (sum the rows)} \\
5 & 7 & 9 & \% \text{ Note sum}(m) \text{ is the same as sum}(m,1) \\
\end{align*}
\]

\[
\begin{align*}
>> \text{sum}(m,2) & \% \text{ sum in 2nd dimension (sum the columns)} \\
6 & 15 & \% = \text{sum}(m.') \\
\end{align*}
\]

(row index, column index)

1\textsuperscript{st} dim 2\textsuperscript{nd} dim
Quiz: The sum function

Question: How do you sum all the elements of a matrix?

```matlab
m = [1 2 3 ; 4 5 6];
>> sum(sum(m))
```

```matlab
total = 0;
for ii = 1:size(m,1)
    for jj = 1:size(m,2)
        total = total + m(ii,jj);
    end
end
```

In other programming languages, a nested for-loop is required.
Logical arrays

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

\[ z = [1 2 3; 4 5 6; 7 8 9]; \]
\[ b = z > 5; \]  
% \( b \) is a logical array, containing 1 and 0  
% 1 represents TRUE, 0 represents FALSE

\[ b = \]
\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 0 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

- Operations on logical arrays
  - \& is the element-wise \textit{and} operator
  - | is the element-wise \textit{or} operator
  - ~ is the \textit{negation} operator

Note:

\&\& is the scalar \textit{and}

| | is the scalar \textit{or}

They are in fact short-circuit operators – next week
Example: Logical arrays (Slide 1)

- Sometimes what you want to do is not the same for every element, but depends on its value
- Example: replace all negative values in an array by zero, and square the positive ones
- If you use loops, this is what you would write.

```matlab
for idx = 1:length(v)
    if v(idx) < 0
        v(idx) = 0
    else
        v(idx) = v(idx)^2
    end
end
```
Example: Logical arrays (Slide 2)

• You can use logical arrays to select elements in an array
  - The two arrays must have the same size

\[
isNeg = v < 0; \quad \% \text{ isNeg is a logical array the same size as } v
\]

\[
v(isNeg) = 0; \quad \% \text{ true values select which subscripts}
\]

\[
v(~isNeg) = v(~isNeg) .^2; \quad \% ~ \text{ is the not operator}
\]

• Demo
Amongst the many useful Matlab functions is **find**, which can be used to locate elements of an array that meet a set of criteria, returning their indexes:

```matlab
>> x = [3, -4, 0, 1.23, 17, 0, -2, 0];
% 1 2 3 4 5 6 7 8 (indexes)
>> find(x) % default: find non-zero values
ans =
   1 2 4 5 7
>> find(x<0) % negative values
ans =
   2 7
```
Matlab function: find (More examples) (1)

```
>> x = [3, -4, 0, 1.23, 17, 0, -2, 0];
% 1 2 3 4 5 6 7 8 (indexes)

>> find(x<0, 1, 'last') % the last n matches (1)
ans =
 7

>> find(x==0, 2, 'first') % the first n matches (2)
ans =
 3 6
```
Matlab function: find (Quiz)

```matlab
>> x = [3, -4, 0, 1.23, 17, 0, -2, 0];
%   1   2   3   4   5   6   7   8 (indexes)

>> find( (x > 1) & (x < 10) )
ans =
    1    4

>> find( (x < 1) | (x > 10) )
ans =
    2    3    5    6    7    8
```
Example: find

- Example: replace all negative values in an array by zero, and square the positive ones
- We can also solve this problem with \texttt{find}

\begin{verbatim}
indicesNeg = find(v < 0);
v(indicesNeg) = 0;

indicesNonNeg = find(v >= 0);
v(indicesNonNeg) = v(indicesNonNeg).^2
\end{verbatim}
Quiz

• How can you find the number of elements in a vector that is > 5?

Method 1: Using logical array

\[ \text{sum}(w > 5) \]

Method 2: Using find

\[ \text{length}(	ext{find}(w > 5)) \]
Example: diff

- Example: diff computes the difference between consecutive elements

\[
y = [4 \ -1 \ 5 \ 6 \ 2];
\]

\[
diff(y)
\]

\[
ans = \\
-5 \ 6 \ 1 \ -4
\]

- For a vector \( v \) with length \( n \), \( \text{diff}(v) \) has a length of \( n-1 \).

- More options, see documentation.
Matlab functions: all, any

• For a vector \( v \),
  - \( \text{any}(v) \) returns a 1 if there is at least one non-zero element in \( v \)
  - \( \text{all}(v) \) returns a 1 if all the elements are non-zero

• For matrices \( m \),
  - \( \text{any}(m,1), \text{any}(m,2), \text{all}(m,1), \text{all}(m,2) \)
  - Default: \( \text{any}(m,1) \) is \( \text{any}(m) \) etc.
    • Same principle as sum
Exercise

- A pedometer is used to count the number of steps that a person has walked. A method to count the number of steps is to count the number of cycles.

- One method to count the number of cycles is to count the number of times the acceleration crosses the zero level and is increasing.
Illustration of rising zero-crossings
Data

• Load the file walkacc.txt into Matlab
  – The variable name will be walkacc

• The file contains acceleration data
  – 1492 rows and 3 columns

• Each row is a sampling instance

• The 3 columns, corresponding to the acceleration in the x-, y- and z-directions
Questions

1. For each sampling instance, compute the net acceleration:

\[ \sqrt{a_x^2 + a_y^2 + a_z^2} - 1 \]

where \( a_x \) is the acceleration in \( x \)-direction etc. The results should be stored in a vector.

2. Compute the number of rising zero-crossings

3. Compute the number of zero-crossings

You are asked to complete each of the above tasks with only one line of Matlab code.

You can complete your work in pedometer_prelim.m
Question 1: illustration

\[
\begin{bmatrix}
  a(1, 1) & a(1, 2) & a(1, 3) \\
  a(2, 1) & a(2, 2) & a(2, 3) \\
  a(3, 1) & a(3, 2) & a(3, 3) \\
  \vdots & \vdots & \vdots \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  \sqrt{a(1, 1)^2 + a(1, 2)^2 + a(1, 3)^2} - 1 \\
  \sqrt{a(2, 1)^2 + a(2, 2)^2 + a(2, 3)^2} - 1 \\
  \sqrt{a(3, 1)^2 + a(3, 2)^2 + a(3, 3)^2} - 1 \\
  \vdots \\
\end{bmatrix}
\]
Questions 2 and 3: illustration

- Assume that the net acceleration is stored in a vector $a$.

![Diagram showing net acceleration over time with points a(1) to a(8)]
Question 1 (Breaking down to small steps) (1)

Final outcome:

\[
\begin{bmatrix}
    a(1, 1)^2 & a(1, 2)^2 & a(1, 3)^2 \\
    a(2, 1)^2 & a(2, 2)^2 & a(2, 3)^2 \\
    a(3, 1)^2 & a(3, 2)^2 & a(3, 3)^2 \\
    \vdots & \vdots & \vdots
\end{bmatrix}
\]

\[
\begin{bmatrix}
    \sqrt{a(1, 1)^2 + a(1, 2)^2 + a(1, 3)^2} - 1 \\
    \sqrt{a(2, 1)^2 + a(2, 2)^2 + a(2, 3)^2} - 1 \\
    \sqrt{a(3, 1)^2 + a(3, 2)^2 + a(3, 3)^2} - 1 \\
    \vdots
\end{bmatrix}
\]
Question 1 (Breaking down to small steps) (2)

\[
\begin{bmatrix}
    a(1,1)^2 + a(1,2)^2 + a(1,3)^2 \\
    a(2,1)^2 + a(2,2)^2 + a(2,3)^2 \\
    a(3,1)^2 + a(3,2)^2 + a(3,3)^2 \\
\end{bmatrix}

\rightarrow

\begin{bmatrix}
    \sqrt{a(1,1)^2 + a(1,2)^2 + a(1,3)^2} \\
    \sqrt{a(2,1)^2 + a(2,2)^2 + a(2,3)^2} \\
    \sqrt{a(3,1)^2 + a(3,2)^2 + a(3,3)^2} \\
\end{bmatrix}

\text{Final outcome:}

\begin{bmatrix}
    \sqrt{a(1,1)^2 + a(1,2)^2 + a(1,3)^2} - 1 \\
    \sqrt{a(2,1)^2 + a(2,2)^2 + a(2,3)^2} - 1 \\
    \sqrt{a(3,1)^2 + a(3,2)^2 + a(3,3)^2} - 1 \\
\end{bmatrix}