Vectorization

▶ We’ve seen vectorized code already:

```matlab
1 A(:,1) = x;
2 A = B.^C;
```

▶ Vectorized code and methods can be a great tool for simplifying code
Vectorization
for vs sum

**Example:** Compute

$$F_n = \sum_{i=1}^{n} f(x_i) \Delta x$$
**Vectorization**

*for vs sum*

**Example:** Compute

\[ F_n = \sum_{i=1}^{n} f(x_i) \Delta x \]

**for loop:**

```sh
1  Fn = 0;
2  for i = 1:n
3      Fn = Fn + ...
4      f(x(i))*Delx;
5  end
```

**Vectorized:**

```sh
1  Fn = sum(f(x))*Delx;
```
More Vectorization
for vs. Alternatives

Example: Normalize to 1 the row sums of $A$:

$$A = \begin{bmatrix}
0 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0
\end{bmatrix} \rightarrow \begin{bmatrix}
0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} \\
0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\
\frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} \\
0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\
0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0
\end{bmatrix}$$

1. for $i = 1:size(A,1)$
2. $A(i,:) = A(i,:) / \text{sum}(A(i,:));$
3. end

- Even with a for loop, we are still “vectorizing” the summation and division
More Vectorization
for vs. Alternatives

Example: Normalize to 1 the row sums of $A$:

\[
A = \begin{bmatrix}
0 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 \\
\end{bmatrix}
\rightarrow
\begin{bmatrix}
0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} \\
0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\
\frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} \\
0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\
0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \\
\end{bmatrix}
\]

\[A = A ./ \text{repmat}(\text{sum}(A,2),1,5);\]

\[\text{repmat} \text{ tiles the row sums so we can divide with ~/}\]
More Vectorization
for vs. Alternatives

**Example:** Normalize to 1 the row sums of $A$:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \end{bmatrix}$$

```matlab
A = diag(1./sum(A,2))*A;
```

- `diag` creates a diagonal matrix for pre-multiplication to scale the rows of $A$
Even More Vectorization

repmat

Example: Use `repmat` to compute $z = x^y$ for
$x = .2, 1.1, 4.2, 5$ and $y = 2, 3, 4$. 

```
x = repmat([.2; 1.1; 4.2; 5],1,3);  
y = repmat(2:4,4,1);  
z = x.ˆy;  
```
Example: Use `repmat` to compute $z = x^y$ for $x = .2, 1.1, 4.2, 5$ and $y = 2, 3, 4$.

```matlab
1 x = repmat([.2; 1.1; 4.2; 5],1,3);
2 y = repmat(2:4,4,1);
3 z = x.^y;
```

- The looping is handled internally by `.^`
Caution: Inefficient Vectorization

Excess Computation or Memory

Recall two example lines to normalize the rows of a matrix $A$

```
1   A = A ./ repmat(sum(A,2),1,5);
2   ...
3   A = diag(1./sum(A,2))*A;
```

- These methods could become suboptimal if $A$ is very large
- `repmat` actually creates a very large matrix to do the division
- `diag` would create a very large diagonal matrix filled mostly with 0s
- Loops or sparse methods can be used to conserve memory and/or limit excess computation
Logicals

\[
\begin{align*}
  &= &\text{equal,} &\sim &= &\text{not equal} \\
  > &= &\text{grter than} &< &= &\text{less than} \\
  \ge &= &\text{grter or eql} &\le &= &\text{less than or sql} \\
  &&\text{and} &| &= &\text{or}
\end{align*}
\]

Table: Logical Operators

- Logical expressions are True/False. Relational operators above can be used to compare scalar values.
- \&\& (and) and \| (or) can be used to join relations like \( a < b \) and \( c \neq 0 \) to make a more complicated expression.
- “Other” Logical expressions:
  - A variable can be assigned true or false: \( a = \text{true} \)
  - Any non-zero value is considered true
  - Tons of functions exist to return true or false:
    - \text{isinf}, \text{isinteger}, \text{isempty}, \text{isnan},...
Logicals

Exercise:

Let $x = 5; \ y = 3; \ z = []; \ a = [1 \ 3 \ 5];$. Determine the result of the following statements.

1. $x > y$
2. isempty(z)
3. $x > z$
4. $x > a$
5. $(x > y) \| (y^2 \geq x && \text{sum}(a) \leq y^2)$
What happened when we did $x > a$?
Logicals

What happened when we did $x > a$?

- MATLAB compared the value $x$ with every value in $a$ and spit out a vector of 1s and 0s.
Logicals

What happened when we did $x > a$?

- MATLAB compared the value $x$ with every value in $a$ and spit out a vector of 1s and 0s.

These 1s and 0s represent true and false, but can they be used in arithmetic?
Logicals

What happened when we did $x > a$?

- **MATLAB** compared the value $x$ with every value in $a$ and spit out a vector of 1s and 0s.

These 1s and 0s represent **true** and **false**, but can they be used in arithmetic?

- Yes they can and we’ll explore this by revisiting the rounding example.
Logicals

Indexing

**Example:** Pick out all elements in the vector \( v \) such that \( v < 10 \) and set them equal to 0.
Logicals

Indexing

**Example:** Pick out all elements in the vector $v$ such that $v < 10$ and set them equal to 0.

$v(v < 10) = 0;$
Example: Pick out all elements in the vector $v$ such that $v < 10$ and set them equal to 0.

$v (v < 10) = 0$;

- Recall, $v < 10$ is a vector of logical 1s and 0s
- $v < 10$ uses the 1s and 0s as a trigger for which elements to select
- Can be used for reference or assignment

```
1  v(v < 10) = 0;
2  or
3  x = v(v<10);
```
Logicals

Indexing

**Example:** Pick out all elements in the vector \( v \) such that \( v < 10 \) and set them equal to 0.

\[ v(v < 10) = 0; \]

- Note how the logical indexing provides a work around for using loops and conditionals

- Suppose you wanted to zero out the elements with \( v < 10 \) and \( v < 0 \):

\[ v(v < 10 \& v < 0) = 0; \]

- Note how \( \& \) works element-by-element! This can be used in arithmetic too
Recall from the last frame the following statement:

\[ v (v < 10 \ & \ v < 0) = 0; \]
Recall from the last frame the following statement:

\[ v (v < 10 \text{ } \& \text{ } v < 0) = 0; \]

Why is there only one \&?
Recall from the last frame the following statement:

\[ v \left( v < 10 \land v < 0 \right) = 0; \]

Why is there only one \&?

The double ampersand \&\& combines two individual values; the single ampersand \& is needed to combine two *vectors* of logical values.
Example: Consider the following game: roll two six-sided dice, and let $x$ and $y$ represent the numbers rolled. If $x$ and $y \geq 3$, you win $x \times y$. Otherwise, you win nothing. What is the average payout?
Simple File I/O

load and save

1. load <filename> <var1> <var2>
2. save <filename> <var1> <var2>

- load and save handle .mat files, which are used to save variables
- The variable arguments are optional and if left out, MATLAB will save the whole workspace or load the entire .mat file
- Use the function forms load(filename, var) and save(filename, var) when filename is a variable, e.g. you have multiple output for a varying parameter that you want saved to different files with names dependent on the parameter value
The `quake.mat` dataset contains accelerometer data in three directions from the Loma Prieta earthquake.  

**Task:** Load the `quake.mat` dataset. Plot the total ground acceleration, save the total ground acceleration and Peak Ground Acceleration in a `.mat` file, and save the figure.  

**Code:**

```matlab
1 load quake
2 g = 9.80*10^3;
3 e = e/g; n = n/g; v = v/g;
4 delt = 1/200;
5 t = delt*(0:length(e)-1)';
6 a = sqrt(e.^2+v.^2+n.^2);
7 PGA3 = max(a); % PGA2 = max(sqrt(e.^2+n.^2)); PGA2 = max(abs([e;n]));
8
9 figure
10 plot(t,a)
11 str = sprintf(['Total Ground Acceleration during ',...
12 'Loma Prieta Earthquake in G (PGA = %.2fG)'],PGA3);
13 title(str)
14 xlabel('Time (in seconds)')
```
The `quake.mat` dataset contains accelerometer data in three directions from the Loma Prieta earthquake.

**Task:** Load the `quake.mat` dataset. Plot the total ground acceleration, save the total ground acceleration and Peak Ground Acceleration in a `.mat` file, and save the figure.