Recap

- Basic Computations
- Scalars, Vectors, and Matrices
  - Creation
  - Individual element reference/assignment
  - Vectorized element reference/assignment
  - Vectorized manipulations
- Scripts & Functions
- Simple I/O
- Basic Plotting
Control Flow

Introduction

- Allows more complex procedures
  - Loops/iterations of both known (for) and indeterminate (while) length
  - Conditional branchings with if–elseif–else and switch
Control Flow
An Example: Craps

Craps is a procedural dice game perfect for learning control flow

- There’s a bet amount $b$
Control Flow
An Example: Craps

Craps is a procedural dice game perfect for learning control flow

- There’s a bet amount $b$
- Opening roll: win $b$ with 7 or 11, lose $b$ with 2, 3, or 12
Craps is a procedural dice game perfect for learning control flow

- There’s a bet amount \( b \)
- Opening roll: win \( b \) with 7 or 11, lose \( b \) with 2, 3, or 12
- 4, 5, 6, 8, 9, or 10: the Point is established as that number
Control Flow
An Example: Craps

Craps is a procedural dice game perfect for learning control flow

- There’s a bet amount $b$
- Opening roll: win $b$ with 7 or 11, lose $b$ with 2, 3, or 12
- 4, 5, 6, 8, 9, or 10: the Point is established as that number
- The dice are rolled until the Point is rolled again (win $b$)
  or a 7 is rolled (lose $b$)
Craps is a procedural dice game perfect for learning control flow

- There’s a bet amount $b$
- Opening roll: win $b$ with 7 or 11, lose $b$ with 2, 3, or 12
- 4, 5, 6, 8, 9, or 10: the Point is established as that number
- The dice are rolled until the Point is rolled again (win $b$) or a 7 is rolled (lose $b$)
- Goal: Simulate the game and compute expected return on a $1$ bet
Control Flow
An Example: Craps

- First: we need to create a function `craps.m` to simulate a single game
Control Flow

An Example: Craps

- First: we need to create a function `craps.m` to simulate a single game
- Second: we need to create a script `craps_sim.m` to do multiple simulations and analyze
Control Flow

if–elseif–else

- if–elseif–else allow different code blocks to be executed according to conditional statements
Control Flow

if–elseif–else

- *elseif* is considered if the condition for *if* is false
- *else* is only considered when *if* and all previous *elseif*’s are false
- *elseif* and *else* are optional but the whole construct must conclude with *end*
Control Flow

if–elseif–else Examples

Two examples:

From `craps.m`:

```matlab
1 if roll2 == 7
2    payout = -bet;
3 else
4    payout = bet;
5 end
```

Determine whether \( x < a \), \( a \leq x < b \), or \( x \geq b \) and report the result.

```matlab
1 if x >= a && x < b
2    disp('a <= x ... < b')
3 elseif x >= b
4    disp('x >= b')
5 else
6    disp('x < a')
7 end
```
Control Flow

if–elseif–else Example

- The relationals <, <=, >, >=, ==, and ~= are used to form *Logical Expressions* \((x \geq a \text{ and } x \geq b)\) which *if* and *elseif* process
- Expressions are joined with && (AND) and || (OR)
Control Flow

for Loops

- for loops perform a fixed length loop on a block of code
Control Flow

for Syntax

Code for Flow Diagram

```
1           for i = rowvector
2           % Block
3           end
```

- Initialization, updating, and termination falls under the statement `i = rowvector`
- Within each iteration of the loop, `i` will assume, in order, a value of `rowvector`
Control Flow

for Syntax

Code for Flow Diagram

```matlab
for i = rowvector
    \% Block
end
```

- The number of iterations is the number of columns in `rowvector`.
- The loop finishes when the block is executed with the last element of `rowvector`. After, `i` retains the last value unless changed.
Control Flow

Example: Compute $z = x^y$ for $x = .2, 1.1, 4.2, 5$ and $y = 2, 3, 4$. 

```matlab
x = [.2; 1.1; 4.2; 5];
y = 2:4;
z = zeros(4,3);
for i = 1:3
    z(:,i) = x.ˆy(i);
end
```

The classic usage is:

```
for i = 1:n
    d iren t u s a g e c o u l d b e
    for y = y,
    s o t h a t
    y cycles through the values of y
```
**Example:** Compute \( z = x^y \) for \( x = .2, 1.1, 4.2, 5 \) and \( y = 2, 3, 4 \).

```matlab
1 x = [.2; 1.1; 4.2; 5];
2 y = 2:4;
3 z = zeros(4,3);
4 for i = 1:3
5    z(:,i) = x.^y(i);
6 end
```
Example: Compute $z = x^y$ for $x = .2, 1.1, 4.2, 5$ and $y = 2, 3, 4$.

```matlab
1 x = [.2; 1.1; 4.2; 5];
2 y = 2:4;
3 z = zeros(4,3);
4 for i = 1:3
5 z(:,i) = x.^y(i);
6 end
```

- The classic usage is `for i = 1:n`
- A different usage could be `for yi = y`, so that $yi$ cycles through the values of $y$
Control Flow

while Loops

- while loops perform a loop on a block of code until the condition is false
Control Flow

while Syntax

Code for Flow Diagram

```
1 while Condition
2   % Block
3   end
```

- The condition is a *Logical Expression* just like we used for the *if–elseif–else* constructs.
- Technically, the loop can run forever. Be sure to have a condition that is guaranteed to be false at some point. A cap on the number of iterations is useful.
Control Flow

Example: Compute $\sqrt{6}$ to within $10^{-12}$ relative change between iterates, $\frac{|x_{n+1} - x_n|}{|x_n|} < 10^{-12}$, using the algorithm,

$$x_0 = 2.5, \quad x_{n+1} = \frac{1}{2} \left( x_n + \frac{6}{x_n} \right)$$
Control Flow

while Example

Example: Compute $\sqrt{6}$ to within $10^{-12}$ relative change between iterates, $\frac{|x_{n+1} - x_n|}{|x_n|} < 10^{-12}$, using the algorithm,

$$x_0 = 2.5, \quad x_{n+1} = \frac{1}{2} \left( x_n + \frac{6}{x_n} \right)$$

```
Y = 6; x0 = 2.5;
err = 1; tol = 1e-12;
k = 1;
while k < 1e4 && ...
    err > tol
    x1 = .5*(x0 + ... Y/x0);
    err = ... abs(x1-x0)/... abs(x0);
7 x0 = x1; k = ...
9 end
```

Note how `err` needs to be initialized and how we use a counter but it’s not really necessary.
Control Flow

switch Statements

- switch’s are similar to if–elseif–else but focus more on categorical/discrete cases
Control Flow

switch Syntax

```
switch variable
  case value
    % Block
  case value
    % Block
  otherwise
    % Block
end
```

- When `variable` is a scalar, a case is executed if `variable == value` returns 1. When `variable` is a string, if `strcmp(variable,value)` returns 1
- `value` can actually hold multiple possibilities, like `case {1,2}`
A stream of grammar labels (e.g. noun, verb, preposition,...) need to be converted to a numeric sequence for processing and assignment.

Example: Convert a single grammar label that could be a noun (1), verb (2), preposition (3), or one of many other possibilities (4) into its corresponding integer so that it may be processed.
Control Flow

Example: Convert a single grammar label that could be a noun (1), verb (2), preposition (3), or one of many other possibilities (4) into its corresponding integer so that it may be processed.

```plaintext
switch label
    case 'noun'
        labelnum = 1;
    case 'verb'
        labelnum = 2;
    case 'prep'
        labelnum = 3;
    otherwise
        labelnum = 4;
end
```
Vectorization

- We’ve seen vectorized code already:

```
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:

```plaintext
1  Fn = 0;
2  for i = 1:n
3      Fn = Fn + ...  
4          f(x(i)) \times \text{Delx};
5  end
```

Vectorized:

```plaintext
1  Fn = \text{sum}(f(x)) \times \text{Delx};
```
More Vectorization
for vs. Alternatives

**Example:** Normalize to 1 the row sums of \( A \):

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

1. `for i = 1:size(A,1)`
2. `A(i,:) = A(i,:)/sum(A(i,:));`
3. `end`

- Even with a `for` loop, we are still “vectorizing” the summation and division
Example: Normalize to 1 the row sums of $A$:

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

```matlab
% A = A ./ repmat(sum(A,2),1,5);
```

- `repmat` tiles the row sums so we can divide with `./`
More Vectorization  
for vs. Alternatives

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

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

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

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

repmat and bsxfun

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 \(^.\)

Even More Vectorization

repmat and bsxfun

**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 \(^{\text{.}^\text{.}}\)

```matlab
1 x = [.2; 1.1; 4.2; 5]; y = 2:4;
2 z = bsxfun(@power,x,y);
```

- Check out the function `bsxfun`, it does what we want!
Caution: Inefficient Vectorization
Excess Computation or Memory

Recall two example lines to normalize the rows of a matrix \( A \)

\[
\begin{align*}
1 & \quad A = A ./ \text{repmat} \left( \text{sum}(A,2), 1, 5 \right); \\
2 & \quad \ldots \\
3 & \quad A = \text{diag} \left( 1./\text{sum}(A,2) \right) \times A;
\end{align*}
\]

- These methods could become suboptimal if \( A \) is very large
- \text{repmat} actually creates a very large matrix to do the division
- \text{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
Caution: Inefficient Vectorization

Excess Computation or Memory

What if you wanted to compute $y_i = x_i^T A x_i = (X^T A X)_{ii}$ for $m \times 1$ vectors $x_1, \ldots, x_n$ and an $m \times m$ matrix $A$?

Here are two methods:

```matlab
1 y = diag(X'*A*X);
2 ...
3 for i = 1:n
4 y(i) = X(:,i)'*A*X(:,i);
5 end
```

- $X'*A*X$ creates an $n \times n$ matrix from which we are only interested in the diagonal entries, hence the use of `diag`
- The first method computes $n^2 - n$ extra entries! Use a loop to cut down on excess computation
Vectorization

- MATLAB contains the standard toolbox of Control Flow but also contains extensive built-in vectorized syntax and methods
- Effective combination yields powerful results