- Today's Lecture:
  - Subfunctions
  - Vectorized code
  - Matrix slicing

- Announcements:
  - Assignment 1 grading feedback expected this weekend; resubmission deadline announced then
  - Assignment 2 to be posted before next lecture.

# Subfunctions

- There can be more than one function in an M-file
- top function is the main function and has the name of the file
- remaining functions are subfunctions, accessible only by the functions in the same m-file
- Each (sub)function in the file begins with a function header
- Keyword **end** is not necessary at the end of a (sub)function. However, if you use it, you must use it consistently.

# Scalar code

- Scalar operation: x + y

  where x, y are scalar variables

  Single value (not containing multiple elements)

- How to add two vectors (element-wise)?
  - Loop over elements
  - Perform scalar operation on each element
  - Generally, vectors should have the same length or shape

    rows/columns

```
for k = 1:length(x)
    z(k)= x(k) + y(k)
end
```

# Vectorized code

—a Matlab-specific feature

- Code that performs element-by-element arithmetic/relational/logical operations on array operands in one step

- Scalar operation: x + y

  where x, y are scalar variables

- Vectorized code: **x** + **y**

  where **x** and/or **y** are vectors. Generally, vectors **x** and **y** should have the same length and shape

# Vectorized addition

$$
\begin{array}{rr|c|c|c|c|}
 & \mathtt{x} & 2 & 1 & .5 & 8 \\
+ & \mathtt{y} & 1 & 2 & 0 & 1 \\
\hline
= & \mathtt{z} & 3 & 3 & .5 & 9
\end{array}
$$

Matlab code: `z= x + y`

# Vectorized subtraction

| | | | | | |
|---|---|---|---|---|---|
| | **x** | 2 | 1 | .5 | 8 |
| - | **y** | 1 | 2 | 0 | 1 |
| = | **z** | 1 | -1 | .5 | 7 |

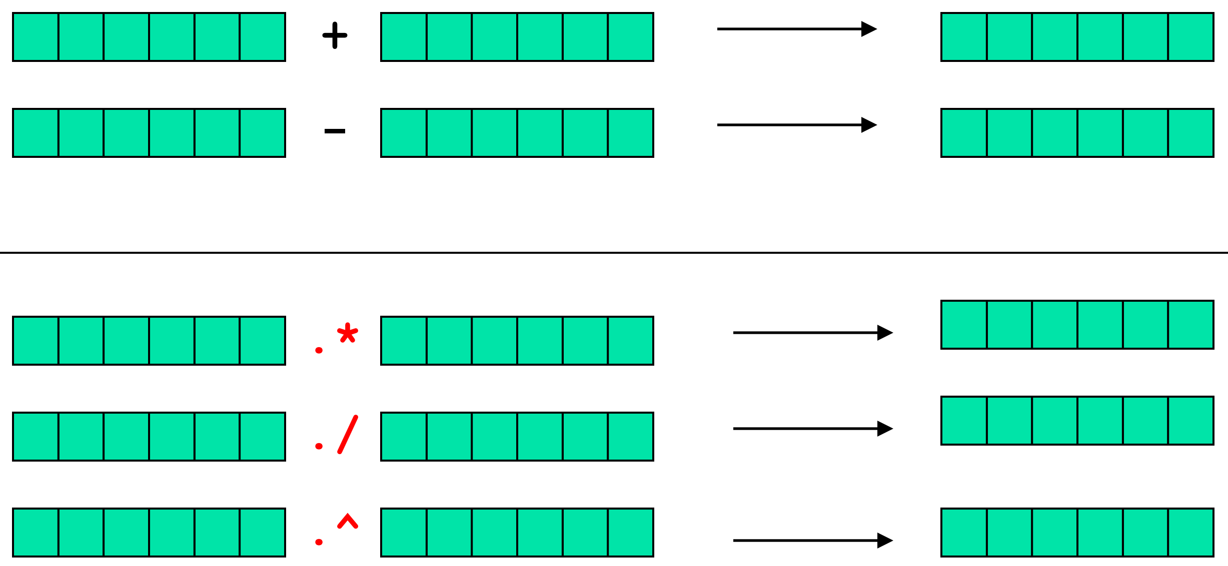
Matlab code: `z= x - y`

# Vectorized multiplication

| | | | | | |
|---|---|---|---|---|---|
| | a | 2 | 1 | .5 | 8 |
| × | b | 1 | 2 | 0 | 1 |
| = | c | 2 | 2 | 0 | 8 |

Matlab code: `c= a .* b`

# Vectorized
## element-by-element arithmetic operations on arrays




A dot (.) is necessary in front of these math operators

# Shift

| | | | | | |
|---|---|---|---|---|---|
| | x | 3 | | | |
| + | y | 2 | 1 | .5 | 8 |
| = | z | 5 | 4 | 3.5 | 11 |

Matlab code: `z= x + y`

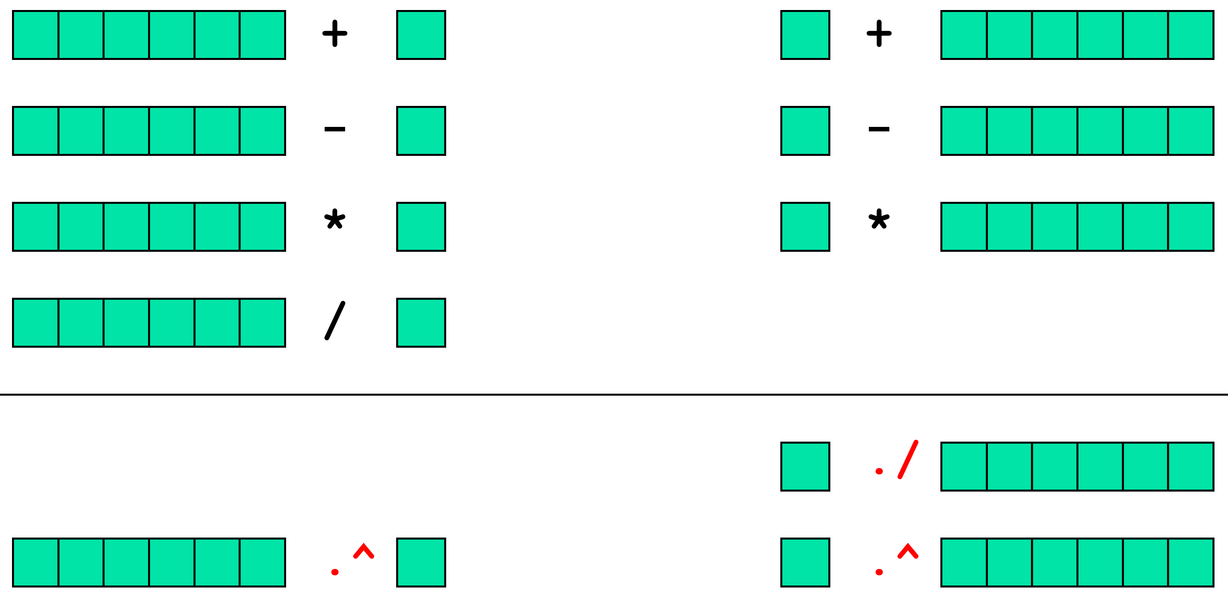
# Reciprocate

| | | | | | | |
|---|---|---|---|---|---|---|
| | `x` | 1 | | | | |
| / | `y` | 2 | 1 | .5 | 8 | |
| = | `z` | .5 | 1 | 2 | .125 | |

Matlab code: `z= x ./ y`

# Vectorized

## element-by-element arithmetic operations between an array and a scalar




A dot (.) is necessary in front of these math operators

Not necessary but OK to use dot for these: .* , .* , ./

# Plot this!

See `plotComparison.m`

$$f(x) = \frac{\sin(5x)\exp(-x/2)}{1+x^2}$$

for

-2 <= x <= 3

```
x = linspace(-2,3,200);
y = sin(5*x).*exp(-x/2)./(1 + x.^2);
plot(x,y)
```

Element-by-element arithmetic operations on arrays

Element-by-element arithmetic operations on arrays…
Also called "vectorized code"

x and y are vectors

```
x = linspace(-2,3,200);
y = sin(5*x).*exp(-x/2)./(1 + x.^2);
```

Contrast with scalar operations that we've used previously…

```
a = 2.1;
b = sin(5*a);
```

a and b are scalars

The operators are (mostly) the same; the operands may be scalars or vectors.

When an operand is a vector, you have "vectorized code."

# Local minimum in a neighborhood

| | | | | |
|---|---|---|---|---|
| 2 | -1 | .5 | 0 | 1 |
| 3 | 8 | 6 | 7 | 7 |
| 5 | -3 | 8.5 | 9 | 10 |
| 52 | 81 | .5 | 7 | 2 |

Component (2,3)

Neighborhood of component (2,3)

# Accessing a submatrix (slicing)

**M**

| | | | | |
|---|---|---|---|---|
| 2 | -1 | .5 | 0 | 1 |
| 3 | 8 | 6 | 7 | 7 |
| 5 | -3 | 8.5 | 9 | 10 |
| 52 | 81 | .5 | 7 | 2 |

Component (2,3)

`M(2,3)`

Neighborhood of component (2,3)

`M(1:3,2:4)`

# Local minimum in a neighborhood

| | | | | |
|---|---|---|---|---|
| 2 | -1 | .5 | 0 | 1 |
| 3 | 8 | 6 | 7 | 7 |
| 5 | -3 | 8.5 | 9 | 10 |
| 52 | 81 | .5 | 7 | 2 |

Component (3,5)

Neighborhood of component (3,5)

# Local minimum in a neighborhood

- Write a function minInNeighborhood
- Input parameters:
  - M: matrix of numeric values
  - loc: location of the middle of the neighborhood

    loc(1), loc(2) are the row, column numbers
- Output parameter: minVal

  The minimum value of the neighborhood

## Lead yourself through problem by asking questions!

- Can you find the min of a (sub)matrix?
  - Yes! Our function minInMatrix(A)
- Given the indices r, c (representing element M(r,c)), is it easy to define the neighborhood?
  - Yes, for the general case the neighborhood is
    M(r-1:r+1, c-1:c+1)
  - But need to deal with the "border cases"

# Local minimum in a neighborhood

M

| | | | | |
|---|---|---|---|---|
| 2 | -1 | .5 | 0 | 1 |
| 3 | 8 | 6 | 7 | 7 |
| 5 | -3 | 8.5 | 9 | 10 |
| 52 | 81 | .5 | 7 | 2 |

Component (3,5)

Want to be able to use the general case,

**`M(r-1:r+1,c-1:c+1)`**

# Local minimum in a neighborhood

M

| | | | | |
|---|---|---|---|---|
| 2 | -1 | .5 | 0 | 1 |
| 3 | 8 | 6 | 7 | 7 |
| 5 | -3 | 8.5 | 9 | 10 |
| 52 | 81 | .5 | 7 | 2 |

Component (3,5)

Want to be able to use the general case,

`M(r-1:r+1,c-1:c+1)`

# Local minimum in a neighborhood

| | | | | | | |
|---|---|---|---|---|---|---|
| B | B | B | B | B | B | B |
| B | 2 | -1 | .5 | 0 | 1 | B |
| B | 3 | 8 | 6 | 7 | 7 | B |
| B | 5 | -3 | 8.5 | 9 | 10 | B |
| B | 52 | 81 | .5 | 7 | 2 | B |
| B | B | B | B | B | B | B |

Create a border of values (B is some big number)

Want to be able to use the general case,

`m(r-1:r+1,c-1:c+1)`

*Note:* This is an exercise on manipulating a matrix. Method not suitable for a large matrix!