Vectorization[1]

- MATLAB favors array operations.
- When two arrays have the same dimensions, addition, subtraction, multiplication, and division apply on an element-by-element basis.
- For example,

```
>> x = [1, 2, 3];
>> y = [4, 5, 6];
>> x + y

 ans =

      5     7     9
```

[1] More about vectorization.

Element-By-Element Operations

Symbol	Operation	Form	Example
+	Scalar-array addition	A + b	[6,3]+2=[8,5]
−	Scalar-array subtraction	A − b	[8,3]−5=[3,−2]
+	Array addition	A + B	[6,5]+[4,8]=[10,13]
−	Array subtraction	A − B	[6,5]−[4,8]=[2,−3]
.*	Array multiplication	A.*B	[3,5].*[4,8]=[12,40]
./	Array right division	A./B	[2,5]./[4,8]=[2/4,5/8]
.\	Array left division	A.\B	[2,5].\[4,8]=[2\4,5\8]
.^	Array exponentiation	A.^B	[3,5].^2=[3^2,5^2] 2.^[3,5]=[2^3,2^5] [3,5].^[2,4]=[3^2,5^4]

- The left division is used in the inverse matrix problems.[2]

[2]We will visit this in the chapter of matrix computation.

Relational Operators[3]

Relational Operator	Interpretation
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
~=	not equal to

- Note that relational operators make comparisons between two arrays of equal size.

[3] See Table 8.1 in Moore, p. 274.

Logical Values

- For example,

```
>> x = 1; y = 2;
>> x == y

ans =

     0
```

- In general, the numeric number 0 is regarded as false while 1 (even any nonzero number) is regarded as true.
- The function **true** and **false** represent logical true and false, respectively.[4]

[4] The usage of **true** and **false** is similar to **zeros**.

Filtering

- Logical arrays are often used as masks (or filters) to manipulate arrays.

```
>> scores = { "Arthur", 50;
                   "Bob", 60;
                 "Cynthia", 70};
>> mask = [scores{:, 2}] >= 60

mask =

     0   1   1

>> scores(mask, 1)

ans =

     {"Bob"     }
     {"Cynthia"}
```

Logical Operators

- Assume $x = 0$.
- How about $1 < x < 3$? (Surprising!)

Logical operator	Name	Description
& Example: $A\&B$	AND	Operates on two operands (A and B). If both are true, the result is true (1), otherwise the result is false (0).
| Example: $A\|B$	OR	Operates on two operands (A and B). If either one, or both are true, the result is true (1), otherwise (both are false) the result is false (0).
~ Example: $\sim A$	NOT	Operates on one operand (A). Gives the opposite of the operand. True (1) if the operand is false, and false (0) if the operand is true.

Truth Table[5]

- Let A and B be two logical variables.
- Then you can find the truth table for logical operators as follows:

A	B	∼A	A&B	A | B
T	T	F	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	F

[5]Note that the basic instructions, such as the plus operator, are implemented by logic gates. See any textbook for digital circuit design.

Exercise: & vs. ==[6]

```
>> u = [0, 2, 0, 4];
>> v = [0, 0, 3, 4];
>> u == v

ans =

     1     0     0     1

>> u & v

ans =

     0     0     0     1
```

[6] Thanks to a lively class discussion (MATLAB-237) on April 16, 2014.

Precedence of Operators[7]

Operators	Precedence
parentheses: ( )	Highest
transpose and power: ' , ^ , .^	
unary: negation (-), not (~)	
multiplication, division *, /, \, .*, ./, .\	
addition, subtraction +, -	
relational <, <=, >, >=, ==, ~=	
element-wise and &	
element-wise or |	
and && (scalars)	
or || (scalars)	
assignment =	Lowest

[7] See Table 1.2 Operator Precedence Rules in Attaway, p. 25.

```
>> Lecture 2
>>
>>              -- Programming Basics
>>
```

"If debugging is the process of removing software bugs, then programming must be the process of putting them in."

– Edsger W. Dijkstra (1930–2002)

Flow Controls

- We wish the computers could make decision on their own.
- Also, the computers should repeat actions for a specified number of times or until the stopping condition is satisfied.
  - As known as loops.
- These two features facilitate the usefulness of computers.
  - Think about the max algorithm.

Building Blocks

- Sequential operations: be executed in order.
- Selections: check which condition is satisfied and then execute the actions accordingly.
- Repetitions: repeat some instructions and stop while the termination condition is satisfied.

Selections

- We start with if followed by a logical expression.
- If true, then do the corresponding statements; otherwise, leave the structure.
- You can also use else to specify the actions if the condition is false.
- For both cases, you need the end statement to finish the selection.

Example: Circle Area

- Write a program which takes a number as input.
  - We use the function **input** which takes a number from the keyboard.
- If the input is positive, then output the resulting circle area.

```
1 clear; clc;
2
3 r = input("Enter r? ");
4 if r > 0
5     A = pi * r ^ 2;
6     disp("The circle area is " + A + "."]);
7 else
8     disp(num2str(r) + " is negative.");
9 end
```

Example: Nested Conditional Statements

```
1  clear; clc;
2
3  s = input("Enter r? ", "s");
4  r = str2num(s);
5  if isempty(r)
6      disp(s + " is not a number.");
7  else
8      if r > 0
9          A = pi * r ^ 2;
10         disp("The circle area is " + A + ".");
11     else
12         disp(s + " is negative.");
13     end
14 end
```

- Use **str2num** to convert from a string to a number.
- Use **isempty** to check if the variable is null.

Example: **if-elseif-else**

```
1  clear; clc;
2
3  s = input("Enter r? ", "s");
4  r = str2num(s);
5  if isempty(r)
6      disp(s + " is not a number.");
7  elseif r >= 0
8      A = pi * r ^ 2;
9      disp("The circle area is " + A + ".");
10 else
11     disp(s + " is negative.");
12 end
```

- More clear!

Exercise

- Write a program to convert centesimal points to letter grades.
- Let x be the input score.
- The conversion rule is as follows:
  - if $90 \leq x \leq 100$, then x is converted to 4;
  - if $80 \leq x < 90$, then 3;
  - if $70 \leq x < 80$, then 2;
  - if $60 \leq x < 70$, then 1;
  - otherwise, 0.

```
1  clear; clc;
2  x = input("Enter your score? ");
3  if 90 <= x && x <= 100
4      disp("4");
5  elseif 80 <= x && x < 90
6      disp("3");
7  elseif 70 <= x && x < 80
8      disp("2");
9  elseif 60 <= x && x < 70
10     disp("1");
11 else
12     disp("0");
13 end
```

- Note that we use && to join two criterion in Line 3.

Short-Circuit Evaluation: && and ||

- Let A and B be two logical results.
- Consider A && B.
- If A returns false, then B won't be evaluated.
- This facilitates time-saving.
- The case of A || B is similar.
- We need to guarantee that the condition is a scalar.

Another Selection Structure: **switch-case**

```
clear; clc;

city = input("Enter a city name: ", "s");
switch city
    case {"Taipei", "New Taipei"}
        disp("Price: $100");
    case "Taichung"
        disp("Price: $200");
    case "Tainan"
        disp("Price: $300");
    otherwise
        disp("Not an option.");
end
```

Equivalence between if and switch[8]

```
1  clear; clc;
2
3  city = input("Enter the city name: ", "s");
4  if city == "Taipei" || city == "New Taipei"
5      disp("Price: $100.");
6  elseif city == "Taichung"
7      disp("Price: $200.");
8  elseif city == "Tainan"
9      disp("Price: $300.");
10 else
11     disp("Not an option.");
12 end
```

[8] Thanks to a lively class discussion (MATLAB-244) on August 20, 2014.

Quantifiers[9]

- The function **all** determines if all elements are true.
- The function **any** determines if there is any true element in the array.

```
>> scores = [50, 60, 70];
>> all(scores >= 60)
ans =

     0

>> any(scores >= 60)
ans =

     1
```

[9] See https://en.wikipedia.org/wiki/Quantifier_(logic).

More Logical Functions

Logical function	
`ischar(A)`	Returns a 1 if `A` is a character array and 0 otherwise.
`isempty(A)`	Returns a 1 if `A` is an empty matrix and 0 otherwise.
`isinf(A)`	Returns an array of the same dimension as `A`, with 1s where `A` has 'inf' and 0s elsewhere.
`isnan(A)`	Returns an array of the same dimension as `A` with 1s where `A` has 'NaN' and 0s elsewhere. ('NaN' stands for "not a number," which means an unde ned result.)
`isnumeric(A)`	Returns a 1 if `A` is a numeric array and 0 otherwise.
`isreal(A)`	Returns a 1 if `A` has no elements with imaginary parts and 0 otherwise.

- NaN: *N*ot *A* *N*umber, caused by $\frac{\infty}{\infty}$ and $\infty - \infty$.[10]

[10] See NaN.

"Logic is the anatomy of thought."

– John Locke (1632–1704)

"This sentence is false."

– anonymous

"I know that I know nothing."

– Plato

(In Apology, Plato relates that Socrates accounts for his seeming wiser than any other person because he does not imagine that he knows what he does not know.)

Repetitions

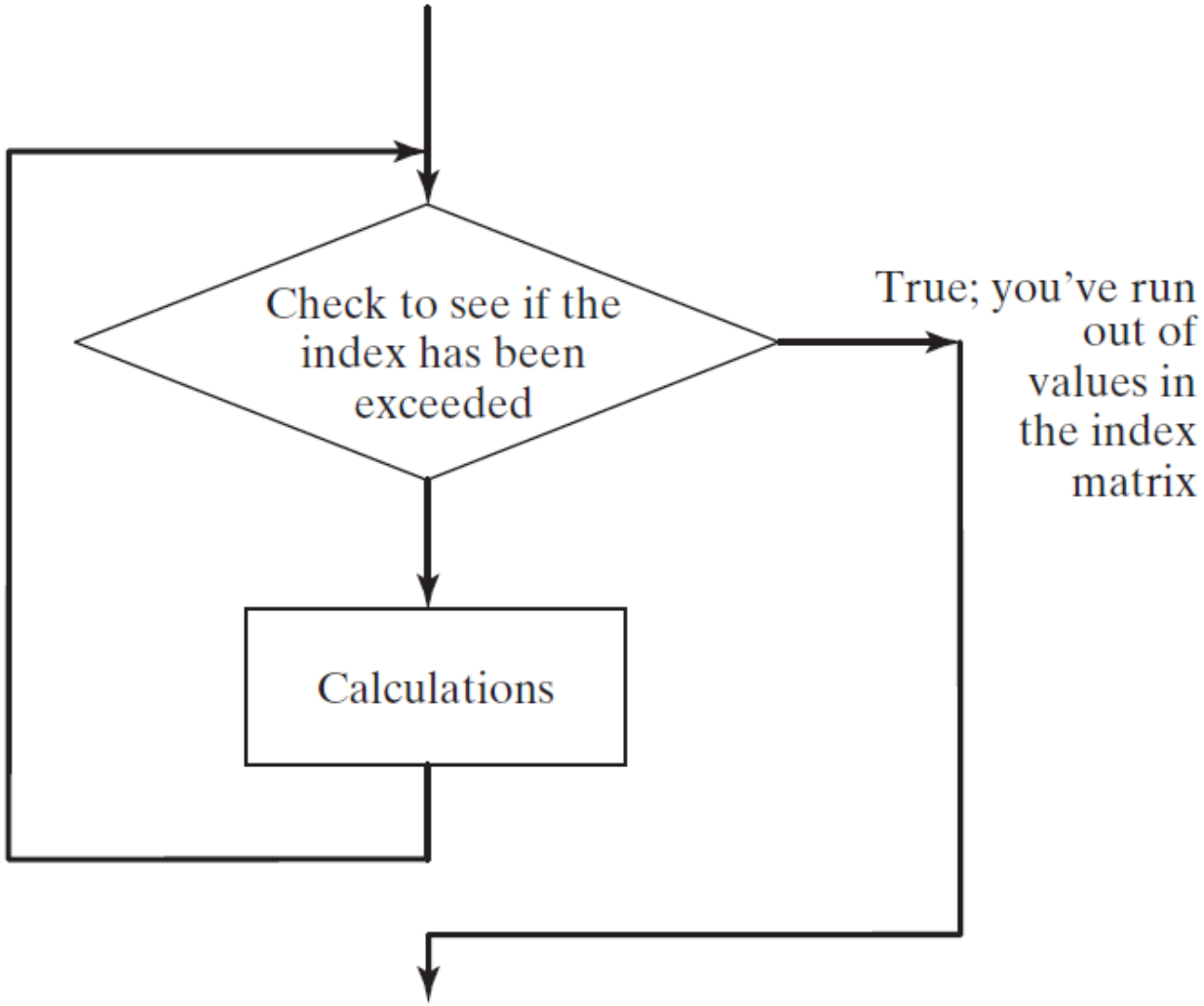
- If some instructions are potentially repeated, you should wrap those in a loop.
- All loops can be done in the following three parts:
  - find the repeated pattern for each iteration;
  - warp them by a proper loop;
  - set the continuation condition by defining a loop variable with some criterion.
- MATLAB has two types of loops: for loops and while loops.
  - Use for loops if you know the number of iterations.
  - Otherwise, use while loops.

for Loops

- A for loop is the easiest choice when you know how many times you need to repeat the loop.

```
1 for loopVar = someArray
2     % body
3 end
```

- Particularly, we often use for loops to manipulate arrays (data)!


Check to see if the index has been exceeded
True; you've run out of values in the index matrix
Calculations

Examples

- Print 1 to 10.

```
1 for i = 1 : 10
2     disp(i);
3 end
```

- How to show the odd integers from 1 to 9?

```
1 stock_list = ["tsmc", "aapl", "goog"];
2 for stock = stock_list
3     disp(stock);
4 end
```

- Clearly, MATLAB has for-each loops, which is an enhanced one compared to the naive one in C.

Example: Find Maximum (Revisited)

```
1  clear; clc;
2
3  data = [4, 9, 7, 2, -1, 6, 3];
4  result = data(1);
5  for item = data(2 :  end)
6      if result < item
7          result = item;
8      end
9  end
10 result
```

- Use **max** in your future work.[11]
- Can you find the location of the maximum element?
- Try to find the minimum element and its location.

[11] Don't repeat yourself.

Exercise: Where is Maximum?

- Write a program which indicates where the maximum is.

```
clear; clc;

data = [4, 9, 7, 2, -1, 6, 3];
loc = 1;
for i = 2 : length(data)
    if data(i) > data(loc)
        loc = i;
    end
end
loc
```

- Note that **max** could return the index of maximum as the second output.

Example: Running Sum

- Write a program which calculates the sum of data.
- Use **randi** to generate a random integer array as testing data.

```
1  clear; clc;
2
3  n = 5;
4  data = randi(100, 1, n)
5
6  sum = 0;
7  for i = 1 : n
8      sum = sum + data(i); % running sum
9  end
10 sum
```

- Of course, you could use **sum** for the same functionality.

Digression: Programming feat. Math

- To sum the sequence $1, 2, \ldots, n$, we could write

$$\text{sum} = 1 + 2 + \cdots + n = \sum_{i=1}^{n} i.$$

- Recall that you write down a loop to add i from 1 to n one by one to an accumulator, say sum.
- See? A summation is realized by a loop!
- From now, you know how to program when you meet a formula like above.

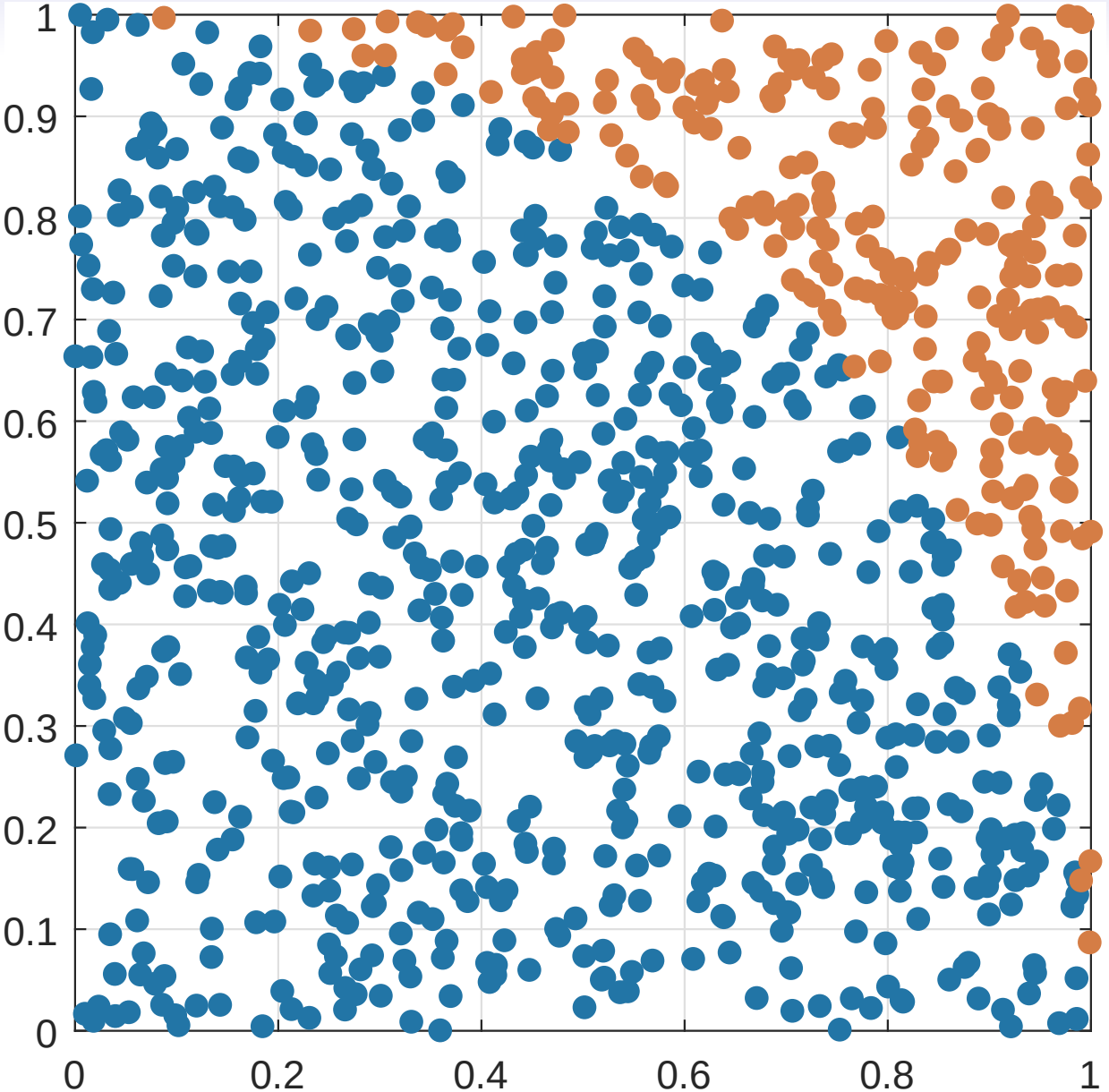
Numerical Example: Monte Carlo Simulation

- Let m be the number of sample points falling in the region of the quarter circle shown in the next page, n be the total number of sample points.
  - Use **rand** to generate a value between 0 and 1 (exclusive).
- Write a program which estimates π by

$$\hat{\pi} = 4 \times \frac{m}{n}.$$

- Note that $\hat{\pi} \to \pi$ as $n \to \infty$ by the law of large numbers (LLN).[12]

[12] See `https://en.wikipedia.org/wiki/Law_of_large_numbers`.


1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1

```
clear; clc;

n = 1e5;
m = 0;

for i = 1 : n

    x = rand(1);
    y = rand(1);

    if x ^ 2 + y ^ 2 < 1
        m = m + 1;
    end

end
result = 4 * m / n
```

- Try to vectorize this program.