- Previous Lecture:
  - Vectors
  - Color computation
  - Linear interpolation
  - Introduction to vectorized computation
- Today's Lecture:
  - "Clean up" details on vectors
  - Vectorized operations
  - 2-d array—matrix
- Announcements:
  - Discussion this week in classrooms as listed in Student Center
  - Prelim I on Oct 13 (Thursday) at 7:30pm

Initialize arrays if dimensions are known

… instead of "building" the array one component at a time

```
% Initialize y
x=linspace(a,b,n);
y=zeros(1,n);
for k=1:n
   y(k)=myF(x(k));
end
```

```
% Build y on the fly
x=linspace(a,b,n);

for k=1:n
   y(k)=myF(x(k));
end
```

Much faster for large n!

Drawing a polygon (multiple line segments)

```
% Draw a rectangle with the lower-left
% corner at (a,b), width w, height h.
x= [                    ];  % x data
y= [                    ];  % y data
plot(x, y)
```

Fill in the missing vector values!

Coloring a polygon (fill)

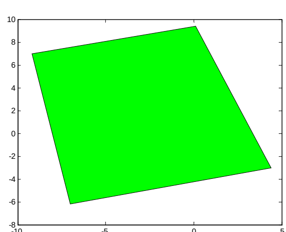
```
% Draw a rectangle with the lower-left
% corner at (a,b), width w, height h,
% and fill it with a color named by c.
x= [a  a+w  a+w  a   a];  % x data
y= [b  b    b+h  b+h b];  % y data
fill(x, y, c)
```

Built-in function **fill** does the "wrap-around" automatically.

Coloring a polygon (fill)

```
x= [0.1 -9.2  -7  4.4];
y= [9.4  7   -6.2  -3];
fill(x,y,'g')
```

Can be a vector (RGB values)



Vectorized code
—a Matlab-specific feature

See Sec 4.1 for list of vectorized arithmetic operations

- 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. If x and y are both vectors, they must be of the same shape and length

Vectorized addition

	x	2	1	.5	8
+	y	1	2	0	1
=	z	3	3	.5	9

Matlab code: **z= x + y**

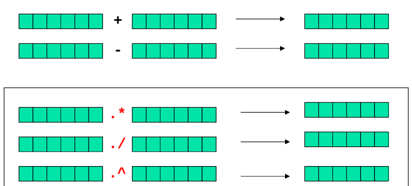
Vectorized multiplication

	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

See full list of ops in §4.1



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**

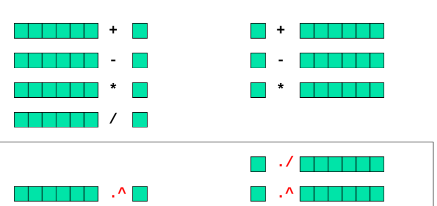
Reciprocate

	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

See full list of ops in §4.1



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

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

Can we plot this?

See `plotComparison.m`

$$f(x)=\frac{\sin(5x)\exp(-x/2)}{1+x^2}$$ for -2 <= x <= 3

Yes!

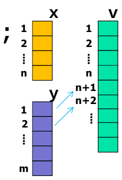
```
x = linspace(-2,3,200);
y = sin(5*x).*exp(-x/2)./(1 + x.^2);
plot(x,y)
```

Element-by-element arithmetic operations on arrays

Lecture 12 23

Concatenating 2 vectors—copy 2 vectors into a new one

```
% given column vectors x and y
v= zeros(length(x)+length(y),1);
for k=1:length(x)
   v(k)= x(k);
end
for k=1:length(y)
   v(length(x)+k)= y(k);
end
```



This is **non-vectorized code**—operations are performed on one component (scalar) at a time

Lecture 13 28

Concatenating 2 vectors—copy 2 vectors into a new one

```
% given column vectors x and y
v= zeros(length(x)+length(y),1);
for k=1:length(x)
   v(k)= x(k);
end
for k=1:length(y)
   v(length(x)+k)= y(k);
end
```

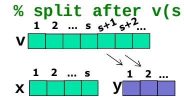
Below is **vectorized code**—ops are performed on multiple components (a vector) at the same time:

```
v = [x; y];
```

Lecture 13 29

Split a vector in 2—copy values into 2 vectors

```
% given row vector v
s= ceil(rand*length(v)); % split after v(s)
x= zeros(1,s);
y= zeros(1,length(v)-s);
for k=1:s
   x(k)= v(k);
end
for k=1:length(y)
   y(k)= v(s+k);
end
```



This is **non-vectorized code**—operations are performed on one component (scalar) at a time

Lecture 13 33

Split a vector in 2—copy values into 2 vectors

```
% given row vector v
s= ceil(rand*length(v)); % split after v(s)
x= zeros(1,s);
y= zeros(1,length(v)-s);
for k=1:s
   x(k)= v(k);
end
for k=1:length(y)
   y(k)= v(s+k);
end
```

Below is **vectorized code**: multiple components (subvectors) are affected/accessed at the same time:

```
x= v(1:s);
y= v(s+1:length(v));
```

Lecture 13 34

End of
Prelim 1 material

Lecture 13 35

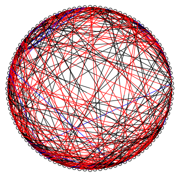
Storing and using data in *tables*

A company has 3 factories that make 5 products with these costs:

C

10	36	22	15	62
12	35	20	12	66
13	37	21	16	59

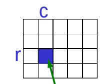
What is the best way to fill a given purchase order?



Connections between webpages

```
0 0 1 0 1 0 0
1 0 0 1 1 1 0
0 1 0 1 1 1 1
1 0 1 1 0 1 0
0 0 1 1 0 1 1
0 0 1 0 1 0 1
0 1 1 0 1 1 0
```


2-d array: **matrix**



- An array is a named collection of like data organized into rows and columns
- A 2-d array is a table, called a *matrix*
- Two *indices* identify the position of a value in a matrix, e.g.,

  `mat(r,c)`

  refers to component in row r, column c of matrix mat
- Array index starts at 1
- Rectangular: all rows have the same #of columns

Creating a matrix

- Built-in functions: ones, zeros, rand
  - E.g., zeros(2,3) gives a 2-by-3 matrix of 0s
  - E.g., zeros(2) gives a 2-by-2 matrix of 0s
- "Build" a matrix using square brackets, [ ], but the dimension must match up:
  - [x y] puts y to the right of x
  - [x; y] puts y below x
  - [4 0 3; 5 1 9] creates the matrix
  - [4 0 3; ones(1,3)] gives
  - [4 0 3; ones(3,1)] doesn't work

4	0	3
5	1	9

4	0	3
1	1	1

Working with a matrix: `size` and individual components

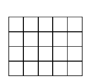
Given a matrix M

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

```
[nr, nc]= size(M)  % nr is #of rows,
                   % nc is #of columns
nr= size(M, 1)  % # of rows
nc= size(M, 2)  % # of columns

M(2,4)= 1;
disp(M(3,1))
M(1,nc)= 4;
```


Example: minimum value in a matrix

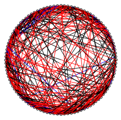


```
function val = minInMatrix(M)
% val is the smallest value in matrix M
```


Pattern for traversing a matrix M

```
[nr, nc] = size(M)
for r= 1:nr
    % At row r
    for c= 1:nc
        % At column c (in row r)
        %
        % Do something with M(r,c) …
    end
end
```

Matrix example: Random Web



- N web pages can be represented by an N-by-N Link Array A.
- A(i,j) is 1 if there is a link on webpage j to webpage i
- Generate a random link array and display the connectivity:
  - There is no link from a page to itself
  - If i≠j then A(i,j) = 1 with probability $\frac{1}{1+|i-j|}$

  ⇨ There is more likely to be a link if i is close to j

Lecture 13 48

```
function A = RandomLinks(n)
% A is n-by-n matrix of 1s and 0s
% representing n webpages

A = zeros(n,n);
for i=1:n
  for j=1:n
    r = rand(1);
    if i~=j && r<= 1/(1 + abs(i-j));
        A(i,j) = 1;
    end
  end
end
```

Lecture 13 49