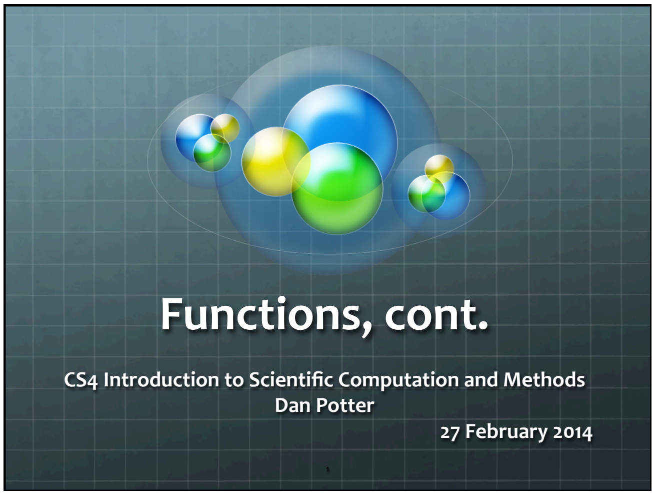
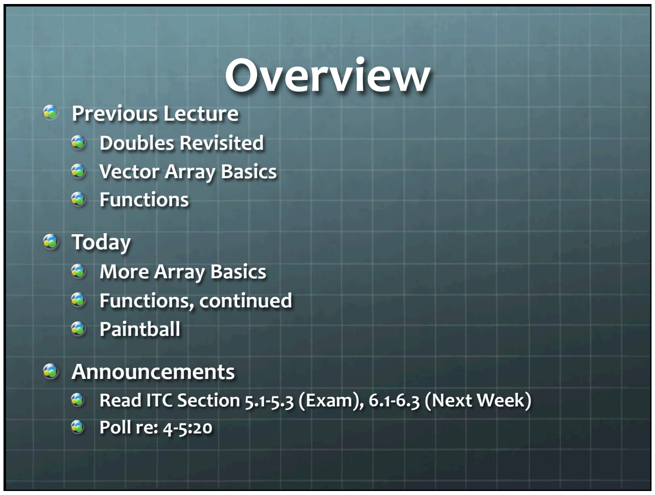


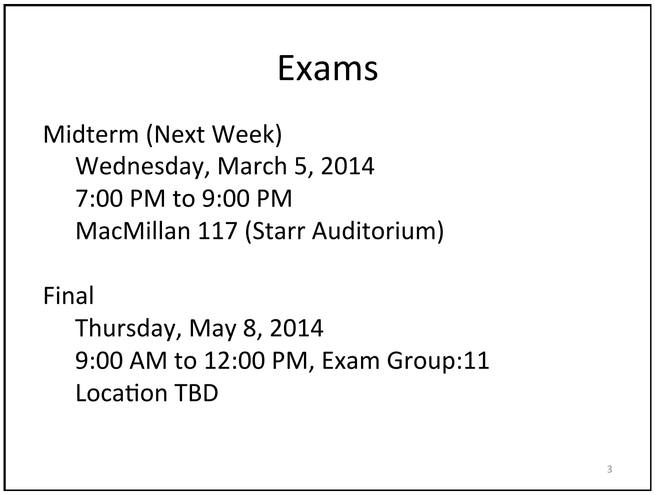
Functions, cont.

CS4 Introduction to Scientific Computation and Methods
Dan Potter
27 February 2014



Overview

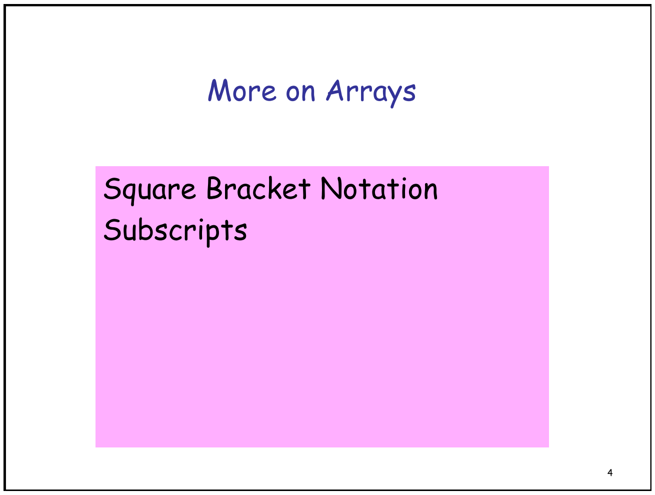
- Previous Lecture
  - Doubles Revisited
  - Vector Array Basics
  - Functions
- Today
  - More Array Basics
  - Functions, continued
  - Paintball
- Announcements
  - Read ITC Section 5.1-5.3 (Exam), 6.1-6.3 (Next Week)
  - Poll re: 4-5:20



Exams

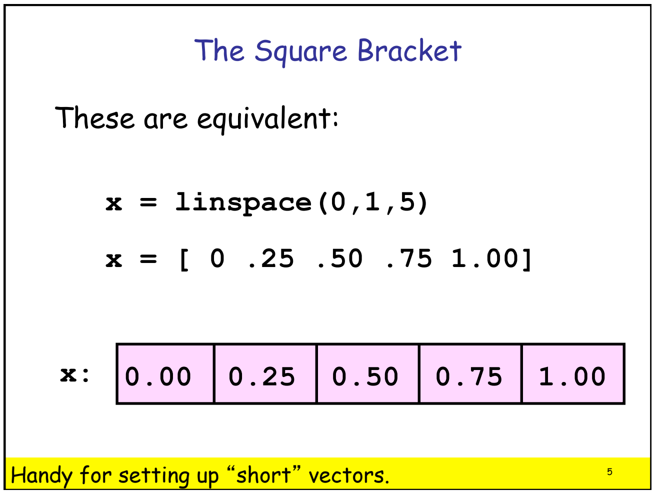
Midterm (Next Week)
Wednesday, March 5, 2014
7:00 PM to 9:00 PM
MacMillan 117 (Starr Auditorium)

Final
Thursday, May 8, 2014
9:00 AM to 12:00 PM, Exam Group:11
Location TBD



More on Arrays

Square Bracket Notation
Subscripts



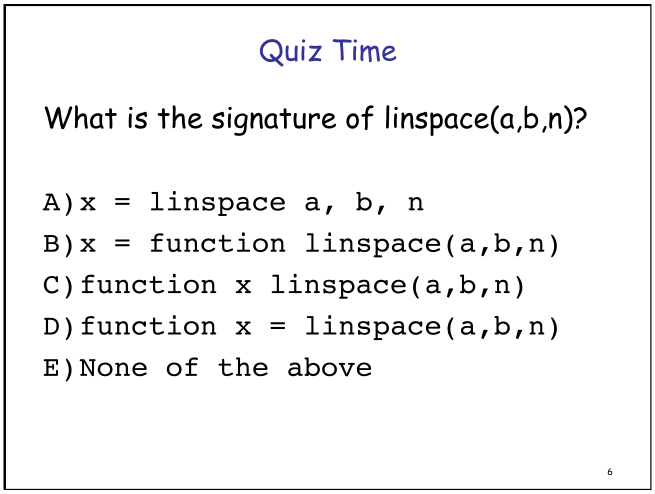
The Square Bracket

These are equivalent:

```
x = linspace(0,1,5)

x = [ 0 .25 .50 .75 1.00]
```

x:				
0.00	0.25	0.50	0.75	1.00

Handy for setting up “short” vectors.



Quiz Time

What is the signature of linspace(a,b,n)?

```
A)x = linspace a, b, n
B)x = function linspace(a,b,n)
C)function x linspace(a,b,n)
D)function x = linspace(a,b,n)
E)None of the above
```

```
function x = linspace(a,b,n)
% linspace(a,b,n) returns n
% equally spaced points
% between a and b, inclusive.
```

Subscripts

It is possible to access and change specific entries in an array.

x:					
	0.00	0.25	0.50	0.75	1.00

The value of **x(1)** is 0.00 .

The value of **x(2)** is 0.25 .

The value of **x(3)** is 0.50 .

The value of **x(4)** is 0.75 .

The value of **x(5)** is 1.00 .

Subscripts

It is possible to access and change specific entries in an array.

x:					
	0.00	0.25	0.50	0.75	1.00

```
a = x(1)
a = x(2)
a = x(3)
a = x(4)
a = x(5)
```

```
0.00
0.25
0.50
0.75
1.00
```

For a = 10, and an array x, what does the following code do?

```
x(1) = a;
x(end) = x(1)
```

Subscripts

It is possible to access and change specific entries in an array.

x:					
	0.00	0.25	0.50	0.75	1.00

```
for k = 1:5
  a = x(k)
  …
end
```

```
0.00
0.25
0.50
0.75
1.00
```

Subscripts

It is possible to access and change specific entries in an array.

x:					
	0.00	0.25	0.50	0.75	1.00

```
a = x(1)+x(2)
a = x(2)+x(3)
a = x(3)+x(4)
a = x(4)+x(5)
```

```
0.25
0.75
1.25
1.75
```

Subscripts

It is possible to access and change specific entries in an array.

x:	0.00	0.25	0.50	0.75	1.00

```
for k=1:4
  a = x(k)+x(k+1)
end
```

```
0.25
0.75
1.25
1.75
```

Subscripts

This

```
x = linspace(a,b,n)
```

is equivalent to this

```
h = (b-a)/(n-1);
for k=1:n
   x(k) = a + (k-1)*h;
end
```

Subscripts

x:	0.00	0.25	0.50	0.75	1.00

```
h = (1-0)/(5-1);
x(1) = 0 + 0*h;
x(2) = 0 + 1*h;
x(3) = 0 + 2*h;
x(4) = 0 + 3*h;
x(5) = 0 + 4*h;
```

Question Time

What is the output?

```
x = [10 20 30];
y = [3 1 2]
k = y(3)-1;
z = x(k+1)
```

A. 11 B. 20 C. 21 D. 30 E. 31

Question Time

What is the output?

```
x = [10 20 30];
y = [3 1 2]
k = y(3)-1;
z = x(k+1)
```

A. 11 [✓ B. 20] C. 21 D. 30 E. 31

Subscripts & Assignment Summary

```
h = (b-a)/(n-1);
for k=1:n
   x(k) = a + (k-1)*h ;
end
```

Where to put it.*

Recipe for a value

* Only now we compute where to put it.

```
function x = linspace(a,b,n)
% linspace(a,b,n) returns n equally spaced
% points between a and b, inclusive.

h = (b-a)/(n-1);
for k=1:n
    x(k) = a + (k-1)*h;
end
```

Question Time

```
function s = prod(x)
% prod(x) returns the product
% of the elements of x, i.e.,
% s = x(1)*x(2)*…*x(end)
```

Code this!

```
function s = prod(x)
% prod(x) returns the product of the
elements of x, i.e.,
% s = x(1)*x(2)*…*x(end)

s=1;
for i = 1:length(x);
    s = s*x(i);
end

% my_prod(0)?, my_prod([])?
% my_prod(1:5)==factorial(5)
% my_prod(0)==factorial(0)?
```

- Properties of log
- Vectorization

If I only have *, ^ and log10, how can I add a and b?

a + b == ???

If I only have *, ^ and log10, how can I add a and b?

a + b== log10(10^a*10^b)
      == log10(10^a)+log10(10^b)
      == a*log10(10)+b*log10(10)
      == a + b

Quiz Time

If I only have prod, .^ and log10, how can I add up the elements of x?

```
sum(x) == ???

A)sum(x) == sum(10^log10(x))
B)sum(x) == log10(prod(x))
C)sum(x) == log10(10.^x)
D)None of the above
```

```
D) None of the above

>> sum(x) == log10(prod(10.^x))
ans =
     1
```

12. More on Functions

Header, Specification, Body
Input Parameter List
Output Parameter List
Built-Ins: **randn, imag, real, max, min, ginput**

Eg. 1: "Gap N"

Keep tossing a fair coin until

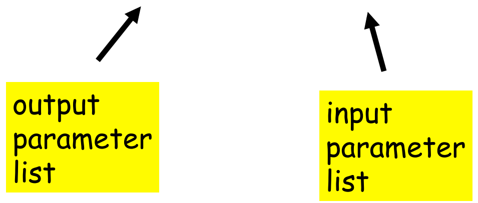
| Heads - Tails | == N

Score = total # tosses

Write a function Gap(N) that returns the score and estimate the average value.

The Header…

```
function nTosses = Gap(N)
```



The Specification

```
function nTosses = Gap(N)
% Simulates a game where you
% keep tossing a fair coin
% until |Heads - Tails| == N.
% N is a positive integer and
% nTosses is the number of
% tosses needed.
```

The Body

```
Heads = 0; Tails = 0; nTosses = 0;
while abs(Heads-Tails) < N
   nTosses = nTosses + 1;
   if rand <.5
       Heads = Heads + 1;
   else
       Tails = Tails + 1;
   end
end
```

The necessary output value is computed.

Local Variables

```
Heads = 0; Tails = 0; nTosses = 0;
while abs(Heads-Tails) < N
   nTosses = nTosses + 1;
   if rand <.5
       Heads = Heads + 1;
   else
       Tails = Tails + 1;
   end
end
```

The Packaging…

```
function nTosses = Gap(N)
```

```
Heads = 0; Tails = 0; nTosses = 0;
while abs(Heads-Tails) < N
   nTosses = nTosses + 1;
   if rand <.5
       Heads = Heads + 1;
   else
       Tails = Tails + 1;
   end
end
```

A Helpful Style

```
Heads = 0; Tails = 0; n = 0;
while abs(Heads-Tails) < N
   n = n + 1;
   if rand <.5
       Heads = Heads + 1;
   else
       Tails = Tails + 1;
   end
end
nTosses = n;
```

Explicitly assign output value at the end.

Estimate Expected Value of Gap(N)

Strategy:

Play “Gap N” a large number of times.

Compute the average “score.”

That estimates the expected value.

Solution…

```
N = input('Enter N:');
nGames = 10000;
s = 0;
for k=1:nGames
   s = s + Gap(N);
end
ave = s/nGames;
```

A very common methodology for the estimation of expected value.

Sample Outputs

```
N = 10  Expected Value =  98.67

N = 20  Expected Value = 395.64

N = 30  Expected Value = 889.11
```

Solution…

```
N = input('Enter N:');
nGames = 10000;
s = 0;
for k=1:nGames
   s = s + Gap(N);
end
ave = s/nGames;
```

Program development is made easier by having a function that handles a single game.

What if the Game Was Not “ Packaged”?

```
s = 0;
for k=1:nGames
   score = Gap(N)
   s = s + score;
end
ave = s/nGames;
```

```
s = 0;
for k=1:nGames
    Heads = 0; Tails = 0; nTosses = 0;
    while abs(Heads-Tails) < N
       nTosses = nTosses + 1;
       if rand <.5
           Heads = Heads + 1;
       else
           Tails = Tails + 1;
       end
    end
    score = nTosses;
   s = s + score;
end
ave = s/nGames;
```

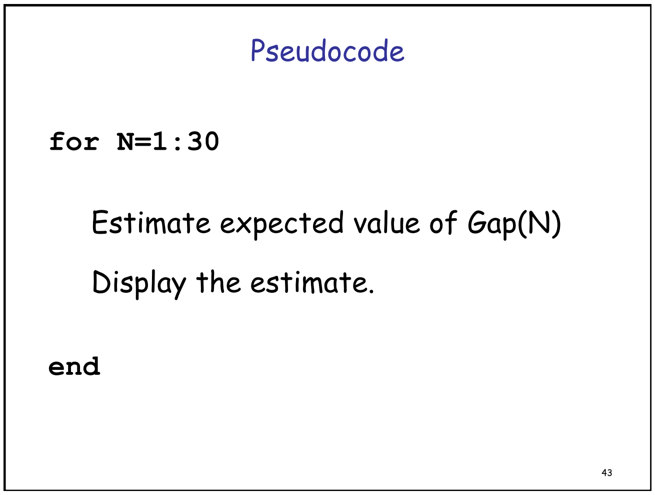
A more cumbersome implementation

Is there a Pattern?

```
N = 10  Expected Value =  98.67

N = 20  Expected Value = 395.64

N = 30  Expected Value = 889.11
```

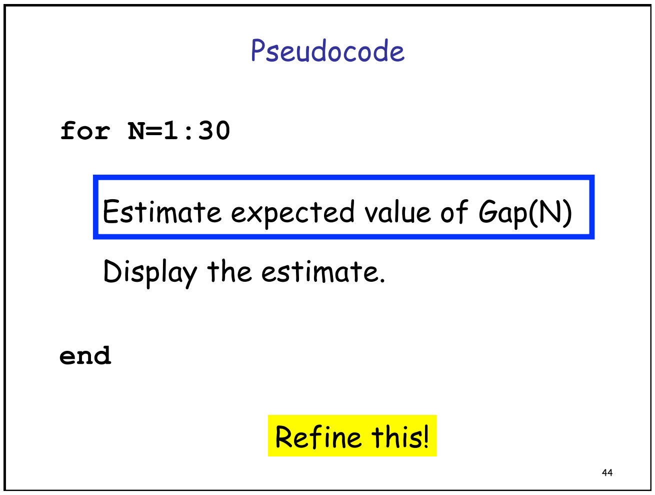
New Problem

Estimate expected value of Gap(N) for a range of N-values, say, N = 1:30



Pseudocode

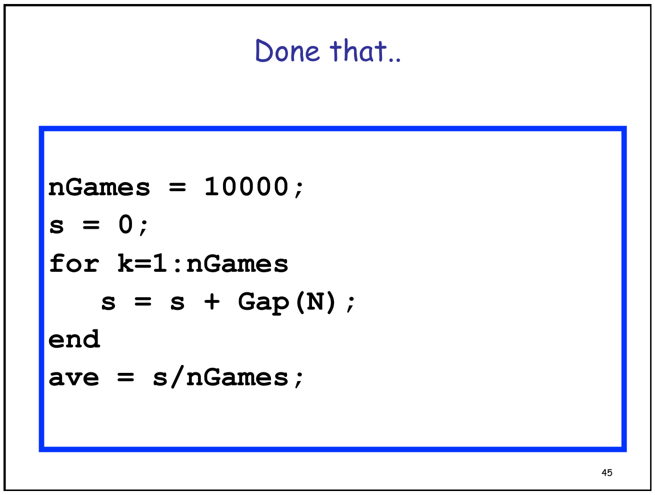
```
for N=1:30

    Estimate expected value of Gap(N)

    Display the estimate.

end
```



Pseudocode

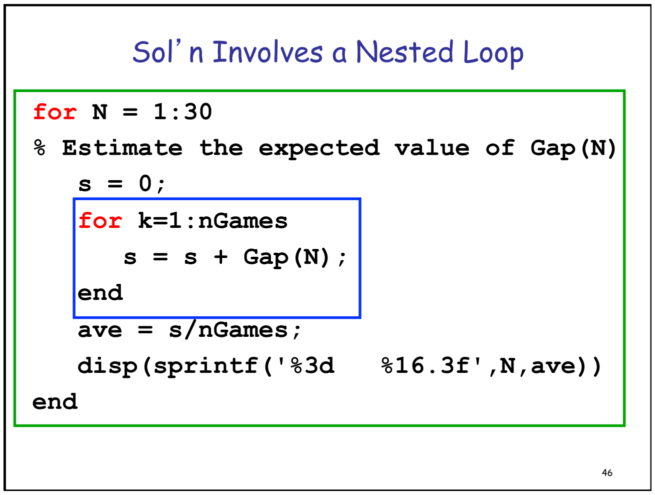
```
for N=1:30

    Estimate expected value of Gap(N)

    Display the estimate.

end
```

Refine this!



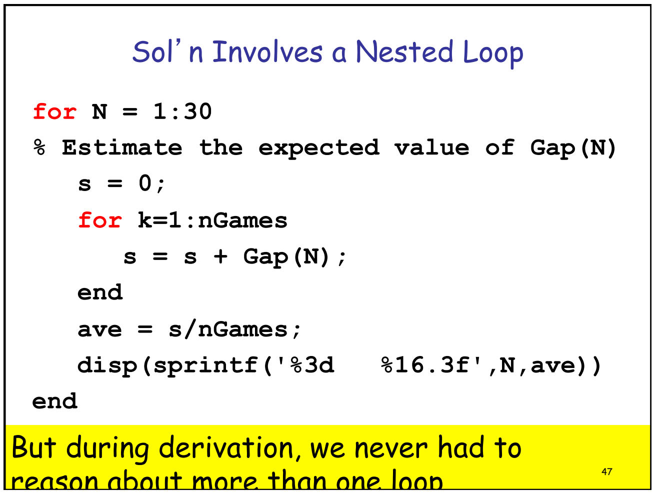
Done that..

```
nGames = 10000;
s = 0;
for k=1:nGames
   s = s + Gap(N);
end
ave = s/nGames;
```



Sol' n Involves a Nested Loop

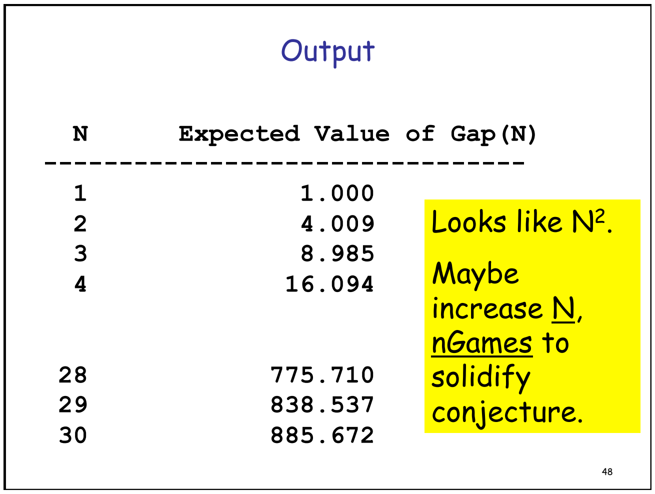
```
for N = 1:30
% Estimate the expected value of Gap(N)
   s = 0;
   for k=1:nGames
      s = s + Gap(N);
   end
   ave = s/nGames;
   disp(sprintf('%3d   %16.3f',N,ave))
end
```



Sol' n Involves a Nested Loop

```
for N = 1:30
% Estimate the expected value of Gap(N)
   s = 0;
   for k=1:nGames
      s = s + Gap(N);
   end
   ave = s/nGames;
   disp(sprintf('%3d   %16.3f',N,ave))
end
```

But during derivation, we never had to reason about more than one loop



Output

```
 N      Expected Value of Gap(N)
--------------------------------
 1                1.000
 2                4.009
 3                8.985
 4               16.094


28              775.710
29              838.537
30              885.672
```

Looks like N^2.

Maybe increase N, nGames to solidify conjecture.

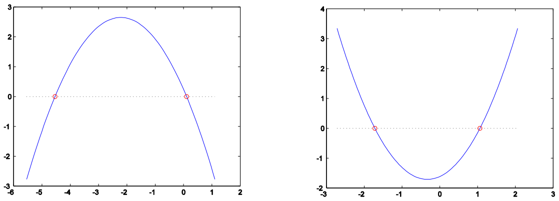
Eg. 2: Random Quadratics

Generate random quadratic

$q(x) = ax^2 + bx + c$

If it has real roots, then plot q(x) and highlight the roots.

Sample Output

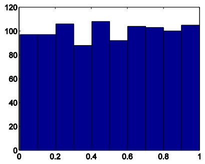


Script Pseudocode

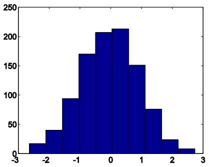
```
for k = 1:10
    Generate a random quadratic
    Compute its roots
    If the roots are real
        then plot the quadratic and
        show roots
end
```

Built-In Function: randn

```
% Uniform
for k=1:1000
   x = rand;
end
```



```
% Normal
for k=1:1000
   x = randn;
end
```



Built-In Functions:
imag and real

```
x = 3 + 4*sqrt(-1);
```

```
y = real(x)
```
Assigns 3 to y.

```
z = imag(x)
```
Assigns 4 to z.

Built-In Functions:
min and max

```
a = 3, b = 4;
```

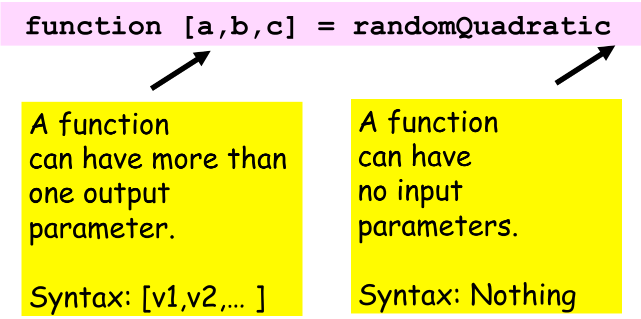
```
y = min(a,b)
```
Assigns 3 to y.

```
z = max(a,b)
```
Assigns 4 to z.

Packaging the Coefficient Computation

```
 function [a,b,c] = randomQuadratic
% a, b, and c are random numbers,
% normally distributed.

a = randn;
b = randn;
c = randn;
```

Input & Output Parameters



```
function [a,b,c] = randomQuadratic
```

A function can have more than one output parameter.

Syntax: [v1,v2,…]

A function can have no input parameters.

Syntax: Nothing

Computing the Roots

```
  function [r1,r2] = rootsQuadratic(a,b,c)

% a, b, and c are real.
% r1 and r2 are roots of
% ax^2 + bx +c = 0.

  r1 = (-b - sqrt(b^2 - 4*a*c))/(2*a);
  r2 = (-b + sqrt(b^2 - 4*a*c))/(2*a);
```

Question Time

```
function [r1,r2] = rootsQuadratic(a,b,c)
r1 = (-b - sqrt(b^2 - 4*a*c))/(2*a);
r2 = (-b + sqrt(b^2 - 4*a*c))/(2*a);
```

```
a = 4; b = 0; c = -1;
[r2,r1] = rootsQuadratic(c,b,a);
r1 = r1
```

Output?

A. 2 B. -2 C. .5 D. -.5

Answer is B.

We are asking `rootsQuadratic` to solve

$-x^2 + 4 = 0$ roots = +2 and -2

Since the function call is equivalent to [r2,r1] = rootsQuadratic(-1,0,4);

Script variable `r1` is assigned the value that `rootsQuadratic` returns through output parameter `r2`. That value is -2

Script Pseudocode

```
for k = 1:10
   Generate a random quadratic
   Compute its roots
   If the roots are real
       then plot the quadratic and
       show roots
end
```

Script Pseudocode

```
for k = 1:10
   Generate a random quadratic
   Compute its roots
   If the roots are real
       then plot the quadratic and
       show roots
end
```

```
[a,b,c] = randomQuadratic;
```

Script Pseudocode

```
for k = 1:10
   [a,b,c] = randomQuadratic;
   Compute its roots
   If the roots are real
       then plot the quadratic and
       show roots
end
```

```
[r1,r2] = rootsQuadratic(a,b,c);
```

Script Pseudocode

```
for k = 1:10
  [a,b,c] = randomQuadratic;
  [r1,r2] = rootsQuadratic(a,b,c);
  If the roots are real
       then plot the quadratic and
       show roots
end
```

```
if imag(r1)==0 && imag(r2)===0
```

Script Pseudocode

```
for k = 1:10
  [a,b,c] = randomQuadratic;
  [r1,r2] = rootsQuadratic(a,b,c);
  if imag(r1)==0 && imag(r2)==0
      then plot the quadratic and
      show roots
  end
end
```

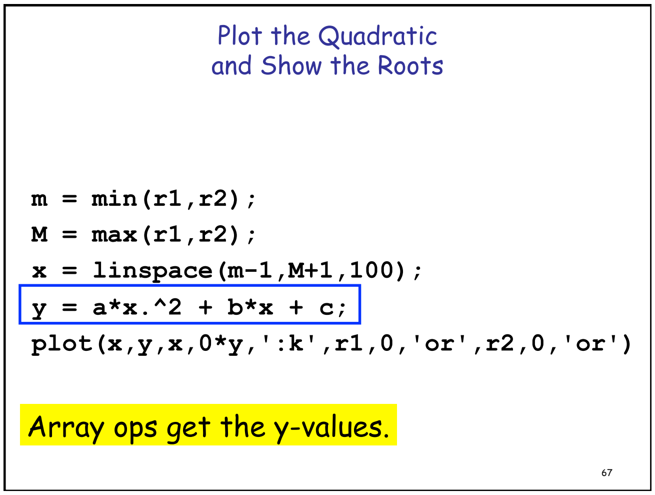
Plot the Quadratic and Show the Roots

```
m = min(r1,r2);
M = max(r1,r2);
x = linspace(m-1,M+1,100);
y = a*x.^2 + b*x + c;
plot(x,y,x,0*y,':k',r1,0,'or',r2,0,'or')
```

Plot the Quadratic and Show the Roots

```
m = min(r1,r2);
M = max(r1,r2);
x = linspace(m-1,M+1,100);
y = a*x.^2 + b*x + c;
plot(x,y,x,0*y,':k',r1,0,'or',r2,0,'or')
```

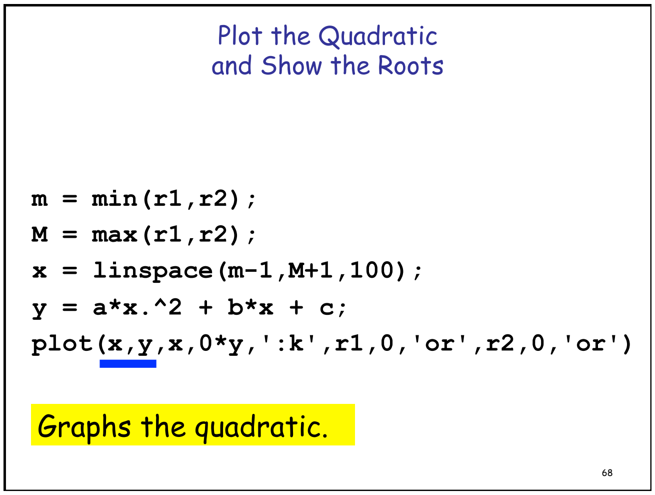
This determines a nice range of x-values.



Plot the Quadratic and Show the Roots

```
m = min(r1,r2);
M = max(r1,r2);
x = linspace(m-1,M+1,100);
y = a*x.^2 + b*x + c;
plot(x,y,x,0*y,':k',r1,0,'or',r2,0,'or')
```

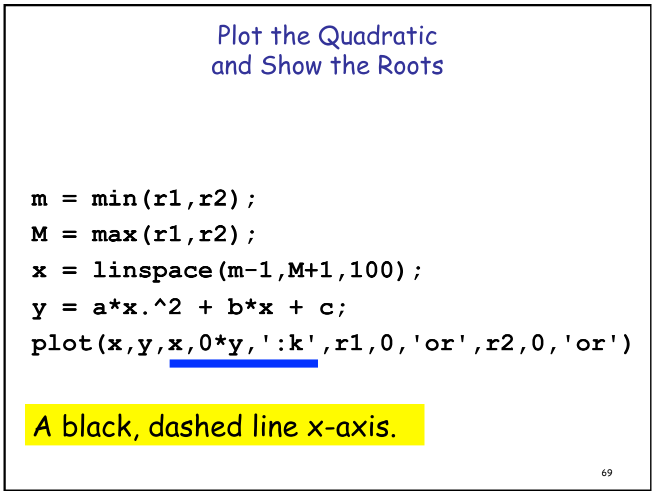
Array ops get the y-values.



Plot the Quadratic and Show the Roots

```
m = min(r1,r2);
M = max(r1,r2);
x = linspace(m-1,M+1,100);
y = a*x.^2 + b*x + c;
plot(x,y,x,0*y,':k',r1,0,'or',r2,0,'or')
```

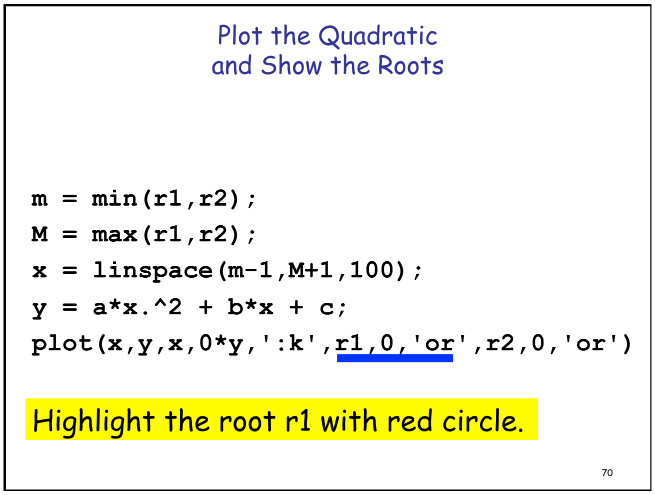
Graphs the quadratic.



Plot the Quadratic and Show the Roots

```
m = min(r1,r2);
M = max(r1,r2);
x = linspace(m-1,M+1,100);
y = a*x.^2 + b*x + c;
plot(x,y,x,0*y,':k',r1,0,'or',r2,0,'or')
```

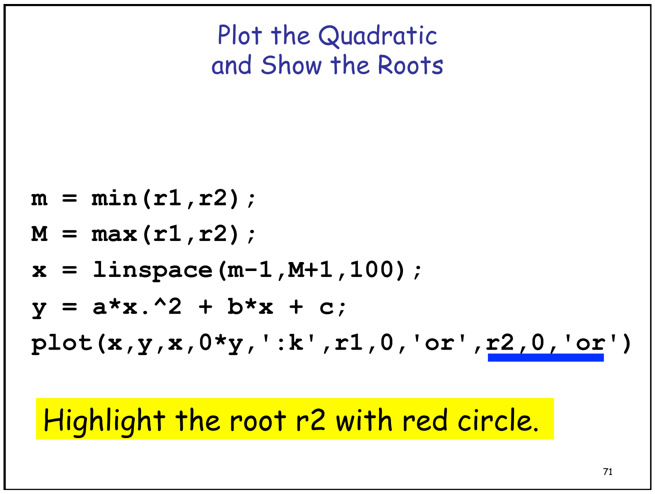
A black, dashed line x-axis.



Plot the Quadratic and Show the Roots

```
m = min(r1,r2);
M = max(r1,r2);
x = linspace(m-1,M+1,100);
y = a*x.^2 + b*x + c;
plot(x,y,x,0*y,':k',r1,0,'or',r2,0,'or')
```

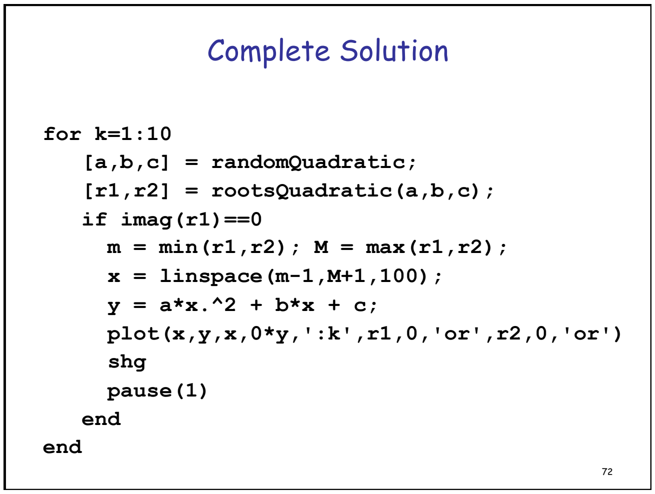
Highlight the root r1 with red circle.



Plot the Quadratic and Show the Roots

```
m = min(r1,r2);
M = max(r1,r2);
x = linspace(m-1,M+1,100);
y = a*x.^2 + b*x + c;
plot(x,y,x,0*y,':k',r1,0,'or',r2,0,'or')
```

Highlight the root r2 with red circle.

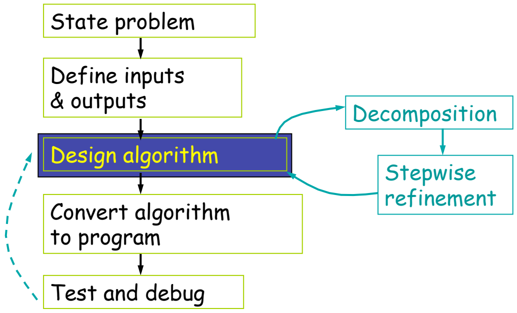


Complete Solution

```
for k=1:10
   [a,b,c] = randomQuadratic;
   [r1,r2] = rootsQuadratic(a,b,c);
   if imag(r1)==0
     m = min(r1,r2); M = max(r1,r2);
     x = linspace(m-1,M+1,100);
     y = a*x.^2 + b*x + c;
     plot(x,y,x,0*y,':k',r1,0,'or',r2,0,'or')
     shg
     pause(1)
   end
end
```

End of Material for Midterm

Top-Down Design



State problem

Define inputs & outputs

Design algorithm

Convert algorithm to program

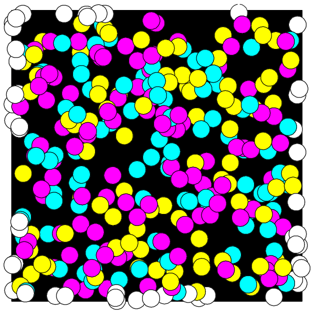
Test and debug

Decomposition

Stepwise refinement

An algorithm is an idea. To use an algorithm you must choose a programming language and implement the algorithm.

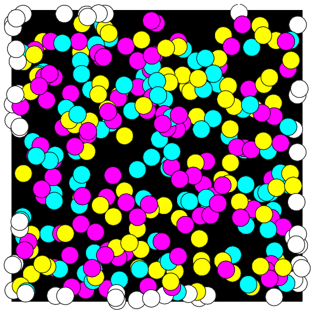
Problem 3: Paintball



Draw a black unit square with lower left corner at (0,0).

Draw a radius .03 disk with center randomly located in square.

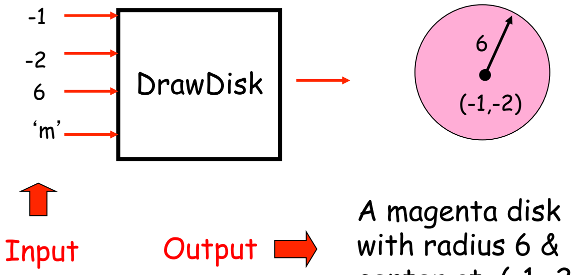
Problem 3: Paintball



If the disk is entirely in square, randomly color it 'c', 'y', or 'm' with equal probability. Otherwise, color it White.

Repeat this process until 50 white disks drawn.

```
DrawDisk(-1,-2,6,'m')
```

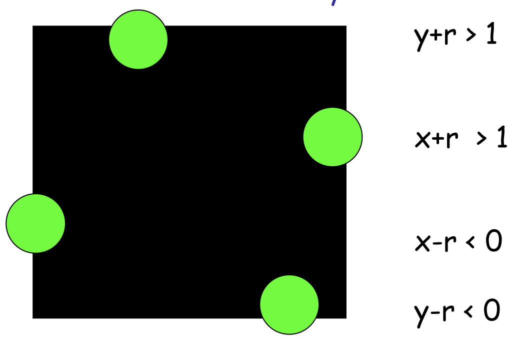


Input

Output

A magenta disk with radius 6 & center at (-1,-2)

Preliminary Notes



Dot: radius r, center (x,y)

"Edge Hits"

Preliminary Notes

How we simulate a 3-way random event?

If `ink = rand(1)`, then

1/3 the time we have: 0 < `ink` < 1/3
1/3 the time we have: 1/3 <= `ink` < 2/3
1/3 the time we have: 2/3 <= `ink` < 1

Check the inequalities and do the right thing.

Pseudocode

```
Draw black square.
Repeat until 50 white disks:
    Locate a random disk.
    If the disk is in the square then
        randomly color it'c', 'y', or 'm'.
    Otherwise,
        color it 'w'
    end
```

Refinement

"Draw the black square"

↓

Draw a unit black square
With lower left corner at (0,0)

↓

`DrawRect(0,0,1,1,'k')`

Pseudocode

```
DrawRect(0,0,1,1,'k')
EdgeHits = 0;
while EdgeHits < 50
    Locate a random disk.
    If the disk is in the square then
        randomly color it'c', 'y', or 'm'.
    Otherwise,
        color it 'w'
        EdgeHits = EdgeHits + 1;
    end
end
```

Variable Definition

We use a variable

`EdgeHits`

to keep track of the number of disks that intersect the square's boundary.

Refinement

"Locate a random disk"

↓

The center (x,y)
satisfies 0<x<1 and 0<y<1.

↓

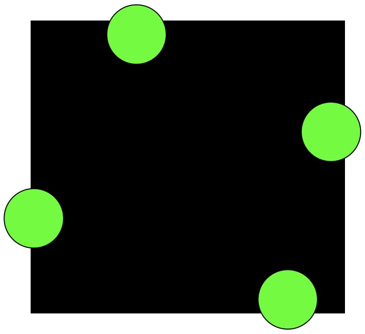
`x = rand; y = rand;`

Refinement

If the disk is in the square then
        randomly color it' c' , 'y' , or 'm' .
Otherwise,
        color it 'w'
        EdgeHits = EdgeHits + 1;
end

How do we check that?

None of these conditions hold.



y+r > 1

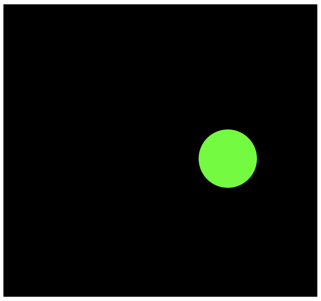
x+r > 1

x-r < 0

y-r < 0

Dot: radius r, center (x,y)

All of these conditions hold.



y+r <= 1

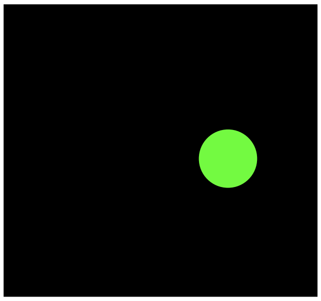
x+r <= 1

x-r >= 0

y-r >= 0

Dot: radius r, center (x,y)

All of these conditions hold.



y+r <= 1

x+r <= 1

x-r >= 0

y-r >= 0

```
y+r<=1  &&  x+r<=1  &&  x-r>=0  &&  y-r>=0
```

Question Time

Want to count upper right corner hits.
Which of these boolean conditions
guarantees that (1,1) is covered?

```
(i)    x + r >= 1   && y + r >= 1
(ii)   x + y >=  2 - 2*r
```

A. Neither   C. Both
B. (i) only   D. (ii) only

AnswerTime

```
 (i)  x + r >= 1   && y + r >= 1
(ii)  x + y >=  2 - 2*r
```

A. Neither   C. Both
B. (i) only   D. (ii) only

Consider 2rx2r disk in corner,
Consider x=1, y=1-r

Refinement

If the disk is in the square then
randomly color it ’c’, ‘y’, or ‘m’.
Otherwise,
color it ‘w’
EdgeHits = EdgeHits + 1;
end

How do we do that?

Refinement

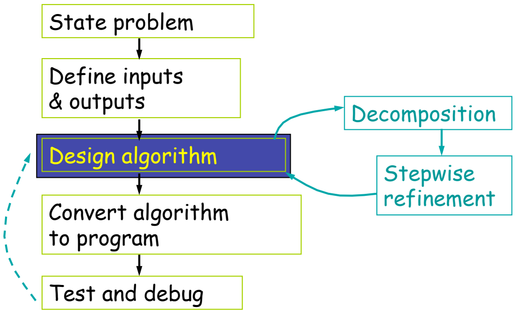
randomly color it ‘c’, ‘y’, or ‘m’

1/3 of the time the disk should be ‘m’
1/3 of the time the disk should be ‘y’
1/3 of the time the disk should be ‘c’

Refinement

```
ink = rand(1);
if ink < 1/3;
   DrawDisk(x,y,r,’m’)
elseif 1/3 <= ink && ink < 2/3
   DrawDisk(x,y,r,’y’)
else
   DrawDisk(x,y,r,’c’)
end
```

Top-Down Design



An algorithm is an idea. To use an algorithm you must choose a programming language and implement the algorithm.