Eg. 1: “Gap N”

Keep tossing a fair coin until

\[|\text{Heads} - \text{Tails}| = N\]

Score = total \# tosses

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

The Packaging...

```matlab
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
```

The Header...

```matlab
function nTosses = Gap(N)

output parameter list
```

```
input parameter list
```

The Body

```matlab
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
```

Local Variables

```matlab
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.
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.

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 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.

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;
```

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
```

A more cumbersome implementation
nGames = 10000;
s = 0;
for k=1:nGames
    s = s + Gap(N);
end
ave = s/nGames;

for N = 1:30
    s = 0;
    for k=1:nGames
        s = s + Gap(N);
    end
    ave = s/nGames;
    fprintf('%3d   %16.3f\n',N,ave)
end

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

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.
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;
end

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);
end

Input & Output Parameters

function [a,b,c] = randomQuadratic
  A function can have more than one output parameter.
  Syntax: [v1,v2,...]
end

function [a,b,c] = rootsQuadratic(a,b,c)
  A function can have no input parameters.
  Syntax: Nothing
end
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);
end

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
\(-x^2 + 4 = 0\) \quad 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
\[
\text{for } k = 1:10 \\
\quad \text{Generate a random quadratic} \\
\quad \text{Compute its roots} \\
\quad \text{If the roots are real} \\
\quad \quad \text{then plot the quadratic and} \\
\quad \quad \text{show roots} \\
\quad \text{end}
\]

Script Pseudocode
\[
\text{for } k = 1:10 \\
\quad \text{Generate a random quadratic} \\
\quad \text{Compute its roots} \\
\quad \text{If the roots are real} \\
\quad \quad \text{then plot the quadratic and} \\
\quad \quad \text{show roots} \\
\quad \text{end}
\]

\[
[a,b,c] = \text{randomQuadratic};
\]

Script Pseudocode
\[
\text{for } k = 1:10 \\
\quad [a,b,c] = \text{randomQuadratic}; \\
\quad \text{Compute its roots} \\
\quad \text{If the roots are real} \\
\quad \quad \text{then plot the quadratic and} \\
\quad \quad \text{show roots} \\
\quad \text{end} \\
\quad [r1,r2] = \text{rootsQuadratic}(a,b,c); \\
\quad \text{If the roots are real} \\
\quad \quad \text{then plot the quadratic and} \\
\quad \quad \text{show roots} \\
\quad \text{end} \\
\quad \text{if imag(r1)==0 && imag(r2)==0}
\]

Script Pseudocode
\[
\text{for } k = 1:10 \\
\quad [a,b,c] = \text{randomQuadratic}; \\
\quad \text{[r1,r2] = rootsQuadratic(a,b,c);} \\
\quad \text{if imag(r1)==0 \&\& imag(r2)==0}
\]
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

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.

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.

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.

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

```matlab
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.
```

Complete Solution

```matlab
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
```