4. Iteration

The FOR loop

“Verify” Prime Number Theorem:

“The number of primes between 1 and N is roughly N/log(N).”

But how can we test if any number J is a prime?

Is “j” a prime number?

Consider every possible divisor $1 < i < j$

If every possible division j/i has a nonzero remainder, then j is prime.

So if j is 100, do I have to write 98 separate MATLAB commands, to check the divisors 2, 3, 4,…,99?

The Program We Don't Want to Write

```
itsaprime = true
if ( mod ( j, 2 ) == 0 )
  itsaprime = false;
end
if ( mod ( j, 3 ) == 0 )
  itsaprime = false;
end
if ( mod ( j, 4 ) == 0 )
  itsaprime = false;
end
(...and so on and so on up to j-1)
```

We see a Pattern

```
if ( mod ( j, * ) == 0 )
  itsaprime = false;
end
```

where “*” will be 2, 3, 4, …, j – 1, in that order.

MATLAB allows us to collect all these statements into one set, with a variable taking the place of *, that will automatically take on each value.

Matlab’s FOR statement

```
itsaprime = true;     ← Assume j is prime

for i = 2 : j – 1        ← Check i=2,3,4,…,j-1
  if ( mod ( j, i ) == 0 )
    itsaprime = false;
  end
end
```

We haven't answered our original question

To "verify" the prime number theorem, we have to do something more complicated.

It amounts to having a PAIR of loops, one inside the other.

```
For every integer J from 1 to 100000
  For every possible divisor I from 2 to J-1
    If J is divisible by I, then J is not a prime
  end
  If J is a prime, increase the prime count
end
```

We will talk about these more complicated loops in a later class.

To repeat something N times:

```
N = ______

for i = 1:N

    Put the something here.

end
```

To repeat something N times:

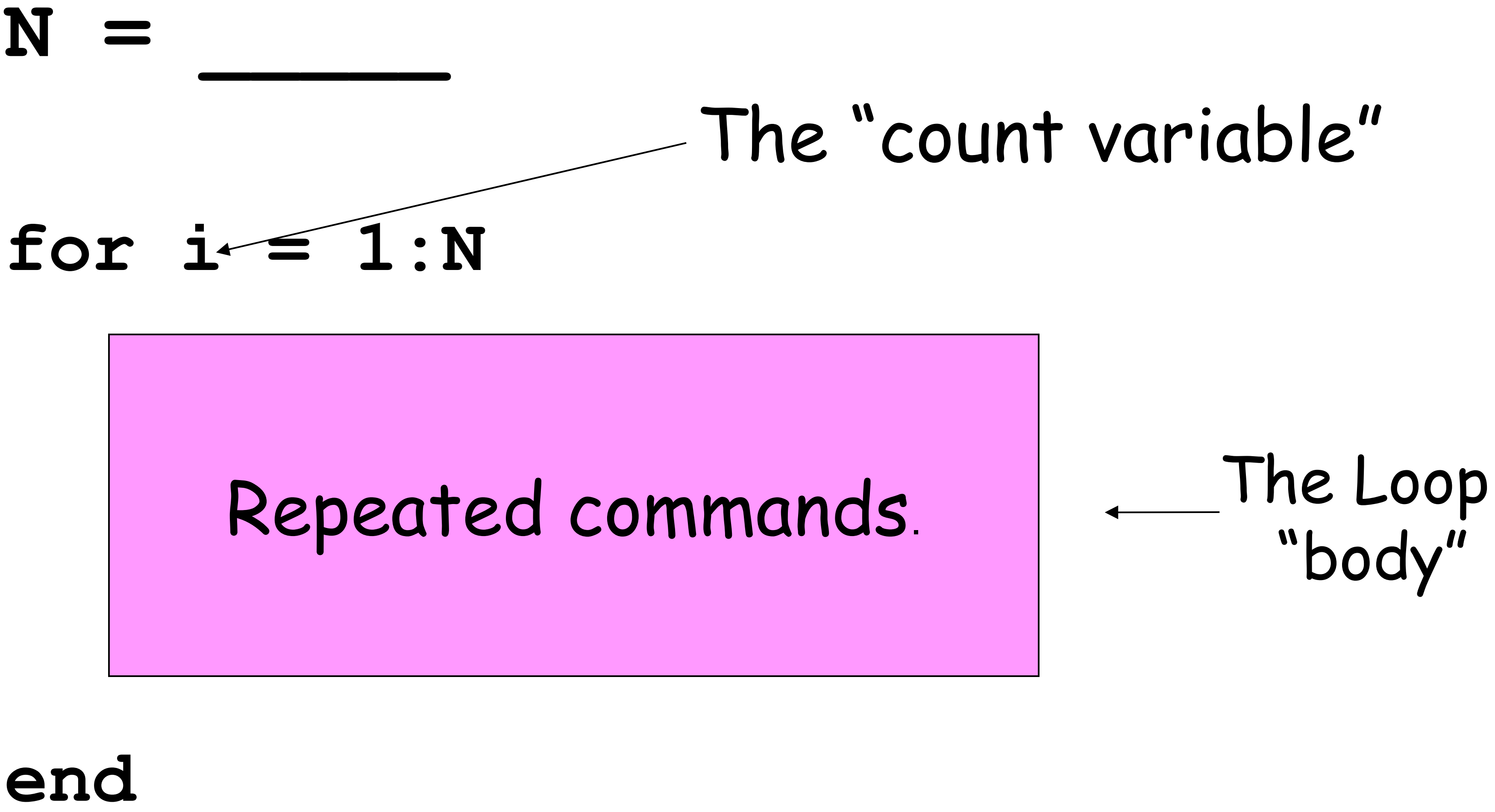
```
N = ______

for i = 1:N
```

Repeated commands here.

The Loop “body”

```
end
```

To repeat something N times:



```
N = ______

for i = 1:N

    Repeated commands.

end
```

The “count variable”

The Loop “body”

Using FOR Loops for sums

It is claimed that a formula for the sum of the integers from 1 to n is n*(n+1)/2.

Write a script that accepts a value n, sums the integers “the hard way”, and compares to the formula.

```
n = input ( ‘Enter the value of N:’ )

s1 = 0;
for i = 1 : n
  s1 = s1 + i;          ←  i can be used in formulas.
end

s2 = n * ( n + 1 ) / 2;

fprintf ( ‘  Sum = %d, Formula = %d\n’, s1, s2 );
```

Loop Index Options

Although names like i, j and k are common, the loop index can have any name.

And the range doesn’t have to start at 1.

```
%  4 percent interest rate
value = 1000;
for year = 2018 : 2037
  value = value + 0.04 * value;
  fprintf ( ‘  In %d, value is %g\n’, year, value );
end
```

Our Square Root Program

```
z = 2017;
x = z;
y = 1;
for i = 1 : 20        ← Maybe 20 steps is enough?
  x = ( x + y ) / 2;
  y = z / x;
end
fprintf ( '  Square root estimate = %g\n’, x );
```

FOR I = LOW : HIGH

The loop starts I at the LOW value. If I is greater than HIGH, then the loop stops, otherwise it executes the loop, then increases I by 1.

```
for I = 20 : 10
  →
(No steps will be taken!)

for I = 35 : 35
  →  I = 35
(just one value is generated)

for I = 99 : 103
  → I = 99, 100, 101, 102, 103
(5 values are generated)
```

FOR Low:Increment:High

A FOR loop with three values uses the middle value as a stepsize, which is otherwise 1 by default.

for I = 1 : 3 : 20

→ I = 1, 4, 7, 10, 13, 16, 19

(increase by 3, don't exceed 20).

for I = 100 : -5 : 0

→ I = 100, 95, 90, …, 5, 0

(decrease by 5, don't go below 0).

When the stepsize is negative, we really should write High:Decrement:Low.

More complicated examples

```
N = 5;
for i = N + 1 : 2 * N
  fprintf ( ‘%d\n’, i )
end

for i = 1 : 5
  for j = 1 : i - 1
    fprintf ( ‘%d’, j );
  end
  fprintf ( ‘\n’ );
end
```

Using Random Numbers

You can compute random numbers. Here are 3 versions of the same command:

```
x = rand;
x = rand ( );          ← I suggest using this!
x = rand ( 1, 1 );
```

MATLAB will return in x a random number between 0 and 1.

Random values change

```
for k = 1:10
   x = rand();
   fprintf(‘%10.6f\n’,x)
end
```

Displays 10 random numbers.

Our 10 random values might be this:

```
0.579736
0.609194
0.256451
0.246079
0.149936
0.564178
0.027311
0.790830
0.437630
0.997130
```

For loops and random values

Simulate flipping a coin. If rand() < 0.5, we got a head, otherwise a tail.

```
heads = 0;
for i = 1 : 1000
  x = rand ( );
  if ( 0.5 <= x )
    heads = heads + 1;
  end
end
```

Estimate Circle Area

Chapter 2.1 of our text estimates the area of a circle using a for loop. (Look at Eg2_1.m in today's "files" directory.)

It does this by dividing the region into tiny boxes and counting how many are inside the circle, whose formula is

x^2 + y^2 <= 1

A simpler approach

Instead, we will estimate the area of the unit circle using random numbers.

Using a for loop, we only have to write one set of statements, and then we can repeat them as many times as we want.

Pick random points (x,y) in [-1,+1]x[-1,+1].

Count how many points are in the circle.

Creating Random Values in a Box

We will need random points (x,y) in the box [-1,+1]x[-1,+1], but rand() gives us values in [0,1]. How do we fix this?

1) the interval [0,1] is 1 unit wide, but we want 2 units wide. Scale: x <= 2 * rand().

2) Now our interval [0,2] starts at 0, but we want to start at -1. Shift: x <= 2*rand() – 1.

3) Same for y.

The program circle_area.m

```
n = 1000;
inside = 0;
for i = 1 : n
  x = 2 * rand ( ) - 1;     ← put x and y into [-1,+1]
  y = 2 * rand ( ) - 1;
  if ( x^2 + y^2 <= 1 )
    inside = inside + 1;
  end
end
area = 4.0 * inside / n;
```

Getting Real Random Values in [A,B]

x = rand() will be $0 < x < 1$.

If we need real random values z in [a,b]:

* we “stretch” the values to size (b-a);
* and we shift the values to start at a;

z = a + (b – a) * rand ( );

To get random values between 35 and 40,

  z = 35 + 5 * rand ( );

To get random values between -1 and +1:

  z = -1 + 2 * rand ( );

Random Integers between A and B

For random integers between A and B,

```
    i = randi ( [ A, B ] )
```

To “roll” a die and get a random result:

```
  roll = randi ( [ 1, 6] )
```

To pick a random day of a (nonleap) year:

```
  day = randi ( [ 1, 365] )
```

To choose a number between -50 and 75

```
  n = randi ( [ -50, 75] )
```

Flipping a coin with randi()

We used rand() to simulate flipping a coin. It might be simpler to use randi(). We can ask for a random integer that is 0 or 1. In fact, if we think of 1 as meaning “heads”, then we can simply add every result.

```
heads = 0;
for i = 1 : 1000
  heads = heads + randi ( [ 0, 1] );
end
```

Splitting a Stick

Question:

A stick with unit length is split into two parts.

The breakpoint is randomly selected.

On average, how long is the shorter piece?

Split Stick Program Strategy

The value x = rand() will tell us where the stick is randomly broken.

The pieces have length x and 1-x.

We can use MATLAB's min() function to tell us which piece is shorter. (There is also a max() function!)

If we do this many times, and average the lengths of the short pieces, we approximate the mathematical answer.

stick_split.m

```
n = 1000;
s = 0.0;
for k = 1 : n
  x = rand ( );
  s = s + min ( x, 1.0 - x );
end
average = s / n
```

Future FOR Loop topics

FOR loops are used to compute sequences.

FOR loops can define vectors.

Nested FOR loops can set up a matrix.

FOR loops let us improve an approximation until we decide it is “close enough”.

We will see how to jump out of a FOR loop if we realize we’re done early.

Exercises

Print numbers divisible by 3 between 10 and 50.

Sum the even numbers between 1 and 100.

Print odd numbers between 20 and 60, but do not print 37!

Roll two dice 20 times, print the maximum sum.

What are the chances that the sum of two random numbers will be less than 1?

Estimate area between x-axis and the graph y=x^2 for 0 <= x <= 1.

New Concepts

```
for i = 1 : n
for i = Low : High
for i = Low : Increment: High
for i = High: Decrement: Low
z = min(x,y)
z = max(x,y)
x = rand ( )
n = randi ( [a,b] )
```

New Concepts

Determine if a number is prime.

Shift random number from [0,1] to a different range.

Simulate random process by averaging.

Estimate area by random sampling.

Estimate probability by averaging trials.