Intro Math Problem Solving
September 7

Last time: FOR, RAND, RANDI

Estimate area under curve

The BREAK statement

Nested FOR loops

Exercises

Homework!

Your Questions

Q1: Difference between e, f, and g formats? What about d format?

A1: Demonstrate with "printing.m".

Q2: When using random sampling, how do you know that N=1000 tests is enough?

A2: You don't! So you try N=100, N=1,000, N=10,000, N=100,000 hoping answers settle down. Demonstrate with "stick_split.m".

integer_prime.m

```
itsaprime = true;


for i = 2 : j - 1
  if ( mod ( j, i ) == 0 )
    itsaprime = false;
  end
end
```

integer_sum.m

```
s = 0;

for i = 1 : n          ← Consider i=1,2,3,…,n
  s = s + i;
end
fprintf ( '  Sum is %d\n', s );
```

square_root.m

```
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 );
```

area_under_curve_random.m

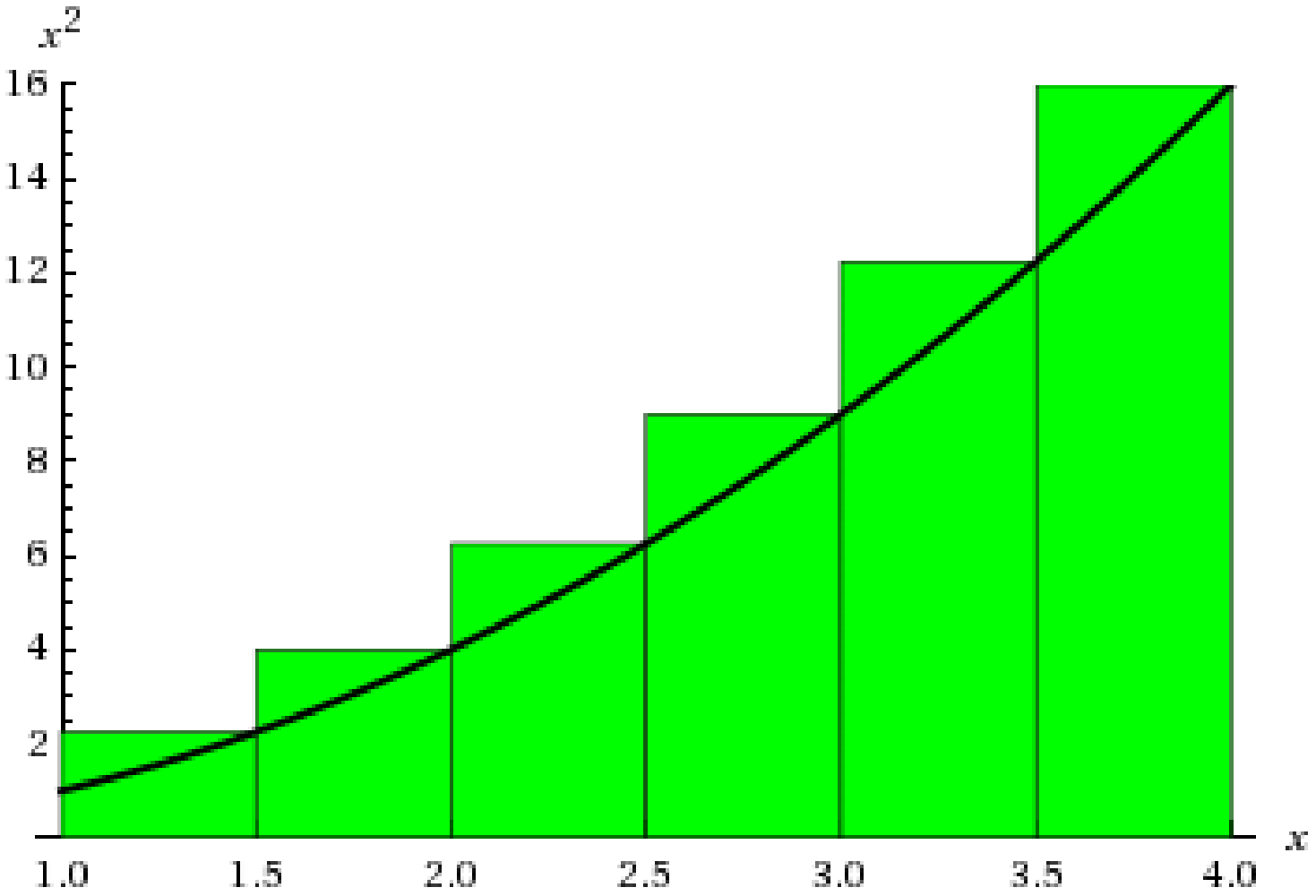
```
n = input ( '  Enter number of sample points: ' );
m = 0;
area_square = 1.0;

for i = 1 : n
  x = rand ( );
  y = rand ( );
  if ( y <= x^2 )
    m = m + 1;
  end
end

area_curve = ( m / n ) * area_square;
```

Area Under a Curve, Using a Grid

We estimated the area under the curve by putting a box around the picture and randomly sampling points in the box.

A more methodical way uses a grid. For this example, we divide the x-interval into N equal subintervals, and look at the area as a collection of `strips' with a raggedy top. We approximate each strip by a rectangle, and sum.

Approximate Area by Rectangles



Details of the grid

How could we express this in MATLAB?

To make N subintervals, we need N+1 points, 0=0/n, 1/n, 2/n, …, (n-1)/n, n/n=1.

Because the whole interval is of length 1, each subinterval is of length dx=1/n.

Interval I goes from x=(i-1)/n to x = i/n.

The "raggedy top" of the I-th strip is the values y=x^2 for (i-1)/n <= x <= i/n.

We approximate the strip by a rectangle, whose area is width*height=(1/n)*y, where we have to pick just one y value from the raggedy top.

area_under_curve_grid.m

```
n = input ( '  Enter the number of intervals to use: ' );

area = 0.0;

for i = 1 : n
  x = i / n;          ← This is the RIGHT endpoint of interval i
  y = x^2;            ← This is the height of the curve at x.
  area_strip = (1/n) * y;
  area = area + area_strip;
end

fprintf ( '  Estimated area under y=x^2 is %g\n', area );
fprintf ( '  Exact area is %g\n', 1 / 3 );
```

Change the problem

Estimate the area under the curve y = sin(x), going from 0 to 2 pi.

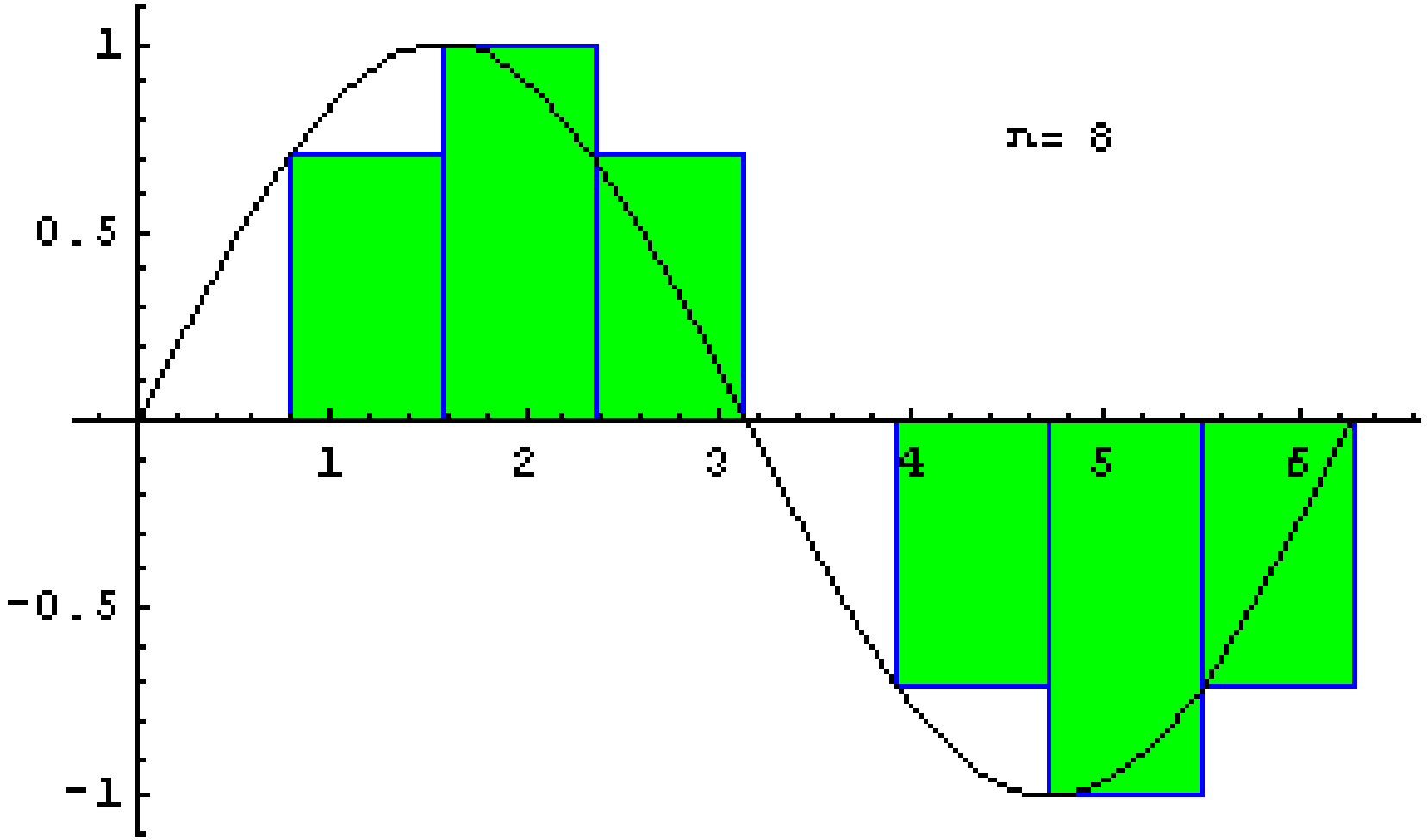
The interval has changed to [0,2pi], so each subinterval has width 2pi/n.

The function sin(x) can be negative, and so we count area below the x-axis as negative.

For this particular problem, the exact answer is 0, because the positive and negative areas will cancel out.

Instead of right endpoints, let's use left ones!

Left endpoint rectangle approximation



area_under_sine_grid.m

```
n = input ( '  Enter the number of intervals to use: ' );

area = 0.0;
width = 2 * pi / n;

for i = 1 : n
  x = 2 * pi * ( i - 1 ) / n;  ← This is the LEFT endpoint of interval i
  y = sin(x);                  ← This is the height of the curve at x.
  area_strip = width * y;
  area = area + area_strip;
end

fprintf ( '  Estimated area under y=sin(x) is %g\n', area );
fprintf ( '  Exact area is %g\n', 0.0 );
```

Make FOR Loops More Flexible

Using a for loop for square roots, we are always taking 20 steps. But sometimes we can tell that after just a few steps, the answer x is `good enough'.

When we check if a number is prime, we start by assuming it is. As soon as we find a single factor, we know it is not a prime. But we keep on, checking all possible factors.

Using the BREAK statement

```
for i = 1:N
   ...some commands...
    if ( something happens)
      ...maybe print a message
         or do something...
      break;
    end
...maybe more commands...
end
```

integer_prime2.m

```
itsaprime = true;

for i = 2 : j - 1
  if ( mod ( j, i ) == 0 )
    itsaprime = false;
    break;              ← no need for further steps!
  end
end
```

square_root2.m

```
z = 2017;
x = z;
y = 1;
for i = 1 : 20
  x = ( x + y ) / 2;
  y = z / x;
  if ( abs ( z – x^2 ) < 0.0000001 )   ← Close enough?
    break
  end
end
fprintf ( '  Square root estimate = %g\n', x );
```

BREAK is helpful

The break statement definitely improves the prime calculation.

For the square root calculation, we still have to wonder if 20 steps is enough for hard problems, and why we picked the value 0.0000001 as a tolerance for early stopping. (The WHILE statement will help us with the first issue.)

A Random Walk (with BREAK)

A person is standing in the dark, in the exact middle of a 10 foot board.

The person will take 20 steps, each randomly chosen to be of length -1 (to the left), 0 (stay), and +1 (to the right).

If the person goes beyond 5 feet, they FALL OFF.

Simulate this process.

Planning the Program

A "position" variable, which starts at 0, will keep track of where the person is.

A for loop will repeat 20 times.

The step size is a random integer between -1 and +1, so we can use randi([-1,+1]) to get the steps.

If the person FALLS OFF, then we can use break to terminate the calculation.

random_walk.m

```
falls_off = false;
position = 0;

for step = 1 : 20

  step_size = randi ( [ -1, +1 ] );
  position = position + step_size;

  if ( position < -5 || +5 < position )
    falls_off = true;
    break;
  end

end

if ( falls_off )
  fprintf ( '  Fell off on step %d\n', step );
else
  fprintf ( '  Safe at position %d\n', position );
end
```

Insight Through

Nested FOR Loops

A for loop lets us do a sequence of items on a list, with the for loop index telling us which item we're working on right now. The index, which might be "i", can be thought of as the row of the list we are working on.

Sometimes, it's more natural to work with a table, that is, a collection of items that are stored in rows and columns. In this case, if we use a pair of for loops, we can work on each item, always knowing what row and column we are working on.

It is common to use "i' for rows and "j" for columns.

An Example From Last Class

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

(First line is blank)

1

1 2

1 2 3

1 2 3 4

Printing a Table

```
for i = 1 : 3
  for j = 1 : 5
    fprintf ( ' %d', 10*i+j );
  end
  fprintf ( '\n' );
end
```

```
11 12 13 14 15
21 22 23 24 25
31 32 33 34 35
```

Printing Matrix Indices

```
for i = 1 : 3
  for j = 1 : 5
    fprintf ( ' (%d,%d)', i, j );
  end
  fprintf ( '\n' );
end
```

(1,1) (1,2) (1,3) (1,4) (1,5)
(2,1) (2,2) (2,3) (2,4) (2,5)
(3,1) (3,2) (3,3) (3,4) (3,5)

Print a Matrix

The Hilbert matrix H(i,j) = 1/(i+j-1);
Print rows 1:10 and columns 1:4.

```
for i = 1 : 10
  for j = 1 : 4
    fprintf ( ' %10.6f', 1/(i+j-1) );
  end
  fprintf ( '\n' );
end
```

Hilbert Matrix

```
>>
```

1.000000	0.500000	0.333333	0.250000	0.200000
0.500000	0.333333	0.250000	0.200000	0.166667
0.333333	0.250000	0.200000	0.166667	0.142857
0.250000	0.200000	0.166667	0.142857	0.125000
0.200000	0.166667	0.142857	0.125000	0.111111
0.166667	0.142857	0.125000	0.111111	0.100000
0.142857	0.125000	0.111111	0.100000	0.090909
0.125000	0.111111	0.100000	0.090909	0.083333
0.111111	0.100000	0.090909	0.083333	0.076923
0.100000	0.090909	0.083333	0.076923	0.071429

Printing Divisors of the number I

The number I is divisible by J if mod(I,J) is 0. To print the divisors of numbers between 10 and 20, we need two loops.

The outer loop chooses the value of I.

The inner loop considers all possible divisors J, 1 <= J <= I.

Jigsaw Puzzle – Put it together!

```
end
end
end
for i = 10 : 20
for j = 1 : i
fprintf ( '\n' );
fprintf ( ' %d', j );
fprintf ( 'Divisors of %d', i );
if ( mod ( i, j ) == 0 )
```

List Divisors of Numbers from 10 to 20

```
for i = 10 : 20
  fprintf ( 'Divisors of %d', i );
  for j = 1 : i
    if ( mod ( i, j ) == 0 )
      fprintf ( ' %d', j );
    end
  end
  fprintf ( '\n' );
end
```

Divisor Output

Divisors of 10: 1 2 5 10
Divisors of 11: 1 11
Divisors of 12: 1 2 3 4 6 12
Divisors of 13: 1 13
Divisors of 14: 1 2 7 14
Divisors of 15: 1 3 5 15
Divisors of 16: 1 2 4 8 16
Divisors of 17: 1 17
Divisors of 18: 1 2 3 6 9 18
Divisors of 19: 1 19
Divisors of 20: 1 2 4 5 10 20

Random Sampling Example

To do random sampling, we use a for loop to carry out the process many times.

In cases like our random walk, the process itself also involves a for loop.

A program would use nested loops:

For I = 1 : number of tests

For j = 1 : 20 (number of steps)

Is Falling Off Likely?

Running the random walk program, it seems like falling off the board is harder to do than we might imagine.

Suppose we wanted to estimate the probability of falling off? There are mathematical ways to do this, but even then, we could use a computation for comparison or reassurance.

We need to do N experiments, and count M, the number of times the person falls.

Need for TWO loops

We have seen other examples of averaging. N counts the trials, M the successes, and M/N the estimated probability of a success.

We wrap a FOR loop around the trial in order to carry out N of them.

Because the random walk already involves a FOR loop, now we have two loops, one "nested" inside the other.

The Outer Loop (J is the index)

Get N from user.

Set m = 0;

for j = 1 : n

  Do the J-th random walk, and increment M
  if the person fell.

End

prob = m / n;

The Inner Loop (STEP is the index)

```
position = 0;
for step = 1 : 20

  step_size = randi ( [ -1, +1 ] );
  position = position + step_size;

  if ( position < -5 || +5 < position )
    m = m + 1;
    break;
  end

end
```

The Averaging Loop + the Walking Loop

```
n = input ( 'Enter N:' )
m = 0;
for j = 1 : n          ← Average N walks

  position = 0;
  for step = 1 : 20     ← Each walk takes up to 20 steps.

    step_size = randi ( [ -1, +1 ] );
    position = position + step_size;

    if ( position < -5 || +5 < position )  ← Stop the walk if we have fallen.
      m = m + 1;
      break;
    end

end

end

prob = m / n;
```

Insight Through

The Prime Number Theorem

In the previous class, we talked about the prime number theorem, which estimates the number of primes you will find between 1 and N:

The number of primes between 1 and N can be estimated as N/log(N).

We can investigate using a pair of loops

```
N = 1000  (or get N as input from the user)
M = 0

For every integer I from 1 to N

 ITSAPRIME = true

 For every possible divisor J from 2 to I-1
  If I is divisible by J
   ITSAPRIME = false
   break
  end
 end

 If ( ITSAPRIME )
  M = M + 1
 End

end

Print N, M, log(N)/N
```

Insight Through

HOMEWORK

Homework #1 is due this Friday midnight.

Homework #2:

hw0091: distance between random points
(nested for loops and rand());
hw016: greatest common divisor table
(nested for loops)
hw023: area under curve y=humps(x)
(use "area under curve grid" ideas).

New Concepts

Estimating area under a curve using regular spaced x values;

Break statement;

Nested for loops;