Intro Math Problem Solving
September 14

Questions

A Duel (Exercise)

Circle Approximation

Coin Tossing

Prime Factors

Gambler's Ruin

Exercise Discussion

Remind: Homework #2

Questions

Question: What's with the %d print format?

You should first know that MATLAB prefers to print all numbers using scientific notation: 1.23456789e+08

The %d prints integers so they look like integers:

```
i = 123456789;
fprintf ( 'i = %d\n', i );  i = 123456789
fprintf ( 'i = %e\n', i );  i = 1.234568e+08
fprintf ( 'i = %f\n', i );  i = 123456789.000000
fprintf ( 'i = %g\n', i );  i = 1.234568e+08
```

However, if your number is not actually an integer, MATLAB will ignore the %d and use "%e" instead!

```
i = 123456789.1;
fprintf ( 'i = %d\n', i );  i = 1.2346e+08
```

A Duel: Exercise

Persons A and B fight a duel to the death. They alternate turns, with A having the first shot, then B, and so on until someone is hit.

On any one shot, person A has a 30% chance of hitting B, while B has a 40% chance of hitting A.

B is a better shot, but A gets the first chance. Can you simulate one instance of this duel? If you run it 10 times, can you estimate A's chance of surviving?

We need a WHILE statement to run the duel, but we have two conditions to check, at different times: did A hit B?; did B hit A?

How do we stop the duel program on time? A BREAK statement is part of the answer.

Try to program this problem, and we will talk about it later today.

FOR versus WHILE

To use the FOR statement, we need to specify an index (such as “I”) and the range of values I will take. We specify the number of steps we expect to take, but a BREAK statement allows us to stop early if some condition is observed.

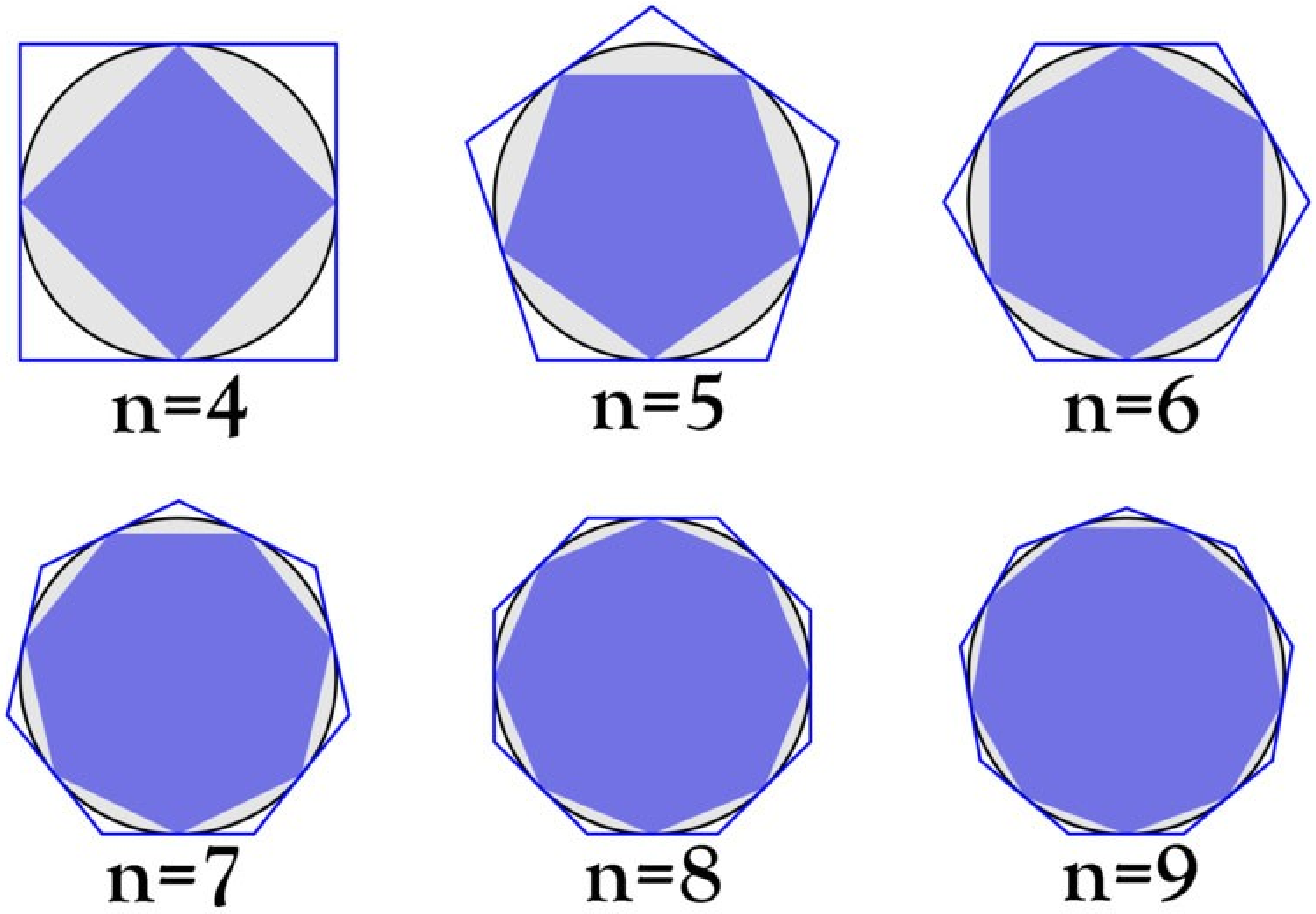
The WHILE specifies a condition that requires us to carry out the given steps, then check the condition again. If we want to count steps, we have to define our own counter (I=0; I = I+1;);

A Typical Use of WHILE: sum_100.m

```
s = 0
k = 0
while ( s < 100 )
  k = k + 1
  s = s + k
end

fprintf ( '1 through %d sum to %d\n', k, s );
```

Circle Approximation



Archimedes Approximates Pi

The Greeks knew exact formulas for the area of polygons, but not a circle. It's pi r^2, of course, but what value is pi?

They wanted to be able to find two numbers A and B, so that they were sure that A <= pi <= B, an underestimate and an overestimate.

This can be done by drawing polygons inside and around the circle.

Estimate Pi to a Given Tolerance

If we increase the number of sides of the polygons, the approximation gets better.

We could design a program that allows a user to specify a desired accuracy.

The program would use polygons with more and more sides. *As A* gets bigger and B gets smaller, we should reach the accuracy request.

Eg2_2.m (Van Loan, Chapter 2.2)

```
delta = input ( 'Enter the error tolerance:' );
nMax  = input ( 'Enter the iteration bound:' );
%
%  Initialize by approximating the circle with triangles.
%
n = 3;                          % Number of sides
A_n = (n/2)*sin(2*pi/n);       % Area of inside triangle
B_n = n*tan(pi/n);              % Area of outside triangle
ErrorBound = B_n - A_n;        % Maximum error in PI estimate.
%
%  As long as  Our estimated error is bigger than the tolerance
%  AND Our value of N hasn't reached the maximum allowed
%
while ( ErrorBound  > delta  && n < nMax )
  n = n + 1;
  A_n = (n/2)*sin(2*pi/n);
  B_n = n*tan(pi/n);
  ErrorBound = B_n - A_n;
end
%
%  Our iteration has stopped.  Report the results.
%
estimate = ( A_n + B_n ) / 2;
fprintf(' delta = %10.3e\n nStar = %1d\n nMax  = %1d\n\n', delta, n, nMax )  % ← How does this print 3 lines?
fprintf(' rho_nStar = %20.15f\n Pi        = %20.15f\n', estimate, pi )
```

A Coin Tossing Duel

Barbara tosses three coins. If she gets all three heads, she wins. Otherwise, Mary takes a turn doing the same thing. The players alternate until one wins.

Model this problem.

When tossing 3 fair coins, the probability of getting 3 heads is 1/8.

HHH ← 1 equally likely chance out of 8 possibilities;
HHT
HTH
HTT
THH
THT
TTH
TTT

Outline of the Model

It's natural to think of the game as a repetition of rounds.

It's natural to think of each round in terms of two turns, Barbara's and Mary's.

The game will end as soon as one player gets three heads, so it's natural to want to make this decision by checking immediately after each player's turn.

An outline of the program

```
While ( ??? )  % ← What can we put here?
  Do Barbara's turn
  If Barbara won, quit  % ← A BREAK
  Do Mary's turn
  If Mary won, quit  % ← A BREAK
end
```

The Details

Barbara's turn involves doing something that has a 1/8 chance of success.

We could model this by either:

A) choose a random integer between 1 and 8, and "win" if its value is 1;

OR

B) choose a random real in [0,1] and "win" if it's no greaer than 1/8.

Outline #2

```
while ( ??? )  % ← What can we put here?

  i = randi ( [ 1, 8 ] )
  if ( i == 1 )
    disp ( 'Barbara wins!' )
    break
  end

  i = randi ( [ 1, 8 ] )
  if ( i == 1 )
    disp ( 'Mary wins!' )
    break
  end

end
```

WHILE What?

The WHILE statement lets us play as many rounds as needed. Ordinarily, we put a condition in the WHILE statement so we know when to stop.

In this example, stopping is taken care of inside the WHILE loop.

To take the decision away from the WHILE loop, we just put the word “true” in the condition box. This means the WHILE will just do the repeating, letting us make the termination decision elsewhere.

Final Program: head_game.m

```
while ( true )  %  ← “Do forever”

  i = randi ( [ 1, 8 ] );
  if ( i == 1 )
    disp ( ‘Barbara wins!’ );
    break;
  end

  i = randi ( [ 1, 8 ] ) ;
  if ( i == 1 )
    disp ( ‘Mary wins!’ );
    break;
  end

end
```

Thinking about the In-Class Exercise

"while(true)" is useful in your duel program:

```
while ( true )
  Player A shoots at B
  If A hits B, A wins, break
  Player B shoots at A
  If B hits A, B wins, break
end
```

The FACTORS Problem

Given a number N, I want to see it written as a product of prime factors.

Examples:

N = 24: 2 * 2 * 2 * 3;

N = 25: 5 * 5;

N = 26: 2 * 13;

N = 27: 3 * 3 * 3;

N = 28: 2 * 2 * 7;

N = 29: 29;

How Would We Do This?

We can look at some numbers and see a factor right away. But to have it done by MATLAB, we have to do this in an organized fashion.

Where do we look, how do we report our results, and how do we know when to stop?

We can look for factors starting at 2 (1 is not considered interesting as a factor!), and we may have to go as high as N (if N is a prime!).

If we find that I is a factor of N, we can print it out immediately.

But what if I is a factor of N more than once?

And how do we know when to stop?

Divide Factor Out of N

Suppose we find that 2 is a factor of N. A logical thing to do is replace N by N/2 and look for factors of the smaller number.

Every time we find a factor, I, replace N by N/I.

As long as the current value of N is greater than 1, we still haven't found all the factors.

As soon as N is 1, we are done...in other words, "AS LONG AS (N is greater than 1)" keep looking! Now we are thinking in terms of a WHILE statement.

A Draft of the Algorithm

```
Get a value of N

AS LONG AS N is greater than 1
  Consider the possible factor I
  If I is a factor of N
    print I
    N = N / I;
  end
end
```

How Do We Search for ALL Factors?

If we walk through this slide, it's clear that we still have to figure out how to work with I. First, it makes sense to initialize I to 2, because that's the first possible factor.

But after we check 2, we need to check 3, and so on.

Why would the following code NOT do what we want? (If N = 12, would it give us 2*2*3?)

A Bad Second Draft

```
Get a value of N
I = 2
AS LONG AS N is greater than 1
  for I = 2 : N
    If I is a factor of N
      print I
      N = N / I;
    end
  end
end
```

There are several things wrong here!

We Need to Handle Repeated Factors

Even if we fix it up, the "bad second draft" idea will only report numbers that are factors of N once.

BUT:

1) We want each prime factor to appear as many times as it is used: 24=2*2*2*3;

2) non-prime factors could appear on the list. We don't want to see 4 as a (nonprime) factor of 24!

Rethinking

AS LONG AS 2 is a factor of N, we want to divide it out.

Adding another WHILE statement lets us repeat this check as many times as necessary.

Once there are no more 2's, we want to go to the next possible factor.

A Third Draft of the Algorithm

```
Get a value of N

I = 2
AS LONG AS N is greater than 1

  AS LONG AS N can be divided evenly by I
    print I
    N = N / I;
  End
  I = I + 1      % ← Time to try the next factor

end
```

Working Program: factors.m

```
% factors.m
%  Find and print the prime factorization of N.
%
n = input ( '  Enter the value of N > 1 to factorize: ' );

if ( n < 1 )
  error ( '  N must be greater than 1.' )
end

i = 2;        % Our first possible factor.


while ( 1 < n )
  while ( mod ( n, i ) == 0 )
    n = n / i;
    fprintf ( ' %d', i );
  end
  i = i + 1;
end
fprintf ( '\n' );
```

Improvements?

Could your code say something like
“24 has 4 factors: 2 2 2 3”

Could your code say:
“24 = 2 * 2 * 2 * 3”

Could your code say:
“24 = 2^3 * 3”

Could your code say:
“-24 = -1 * 2 * 2 * 2 * 3”

In Class Exercise: Player A shoots at B

To model what happens when A shoots at B, let's use rand() to get a number x between 0 and 1.

To model whether A hits B, we want to say something about x that is true 30% of the time. One such statement is that x <= 0.3.

If we decide A hits B, we break.

We do something similar on B's turn.

Gambler's Ruin

Here is a problem that is similar to a duel, but involves winning or losing cash.

Suppose Haley has $100 and Terry has $80, and they repeatedly toss a coin. Each head means Haley wins $10 from Terry, while a tail means Terry wins $10 from Haley. Play continues until someone is broke.

Model this game.

Outline of Gambling Program

Variables h_cash and t_cash track Haley and Terry's cash.

Our game will involve repetition for an unknown number of steps, so a while() is what we need.

We could use either version of the WHILE statement to play this game: while (true) + break, or while (condition).

On each turn, we need to:

* toss a random coin, getting 0=T or 1=H;
* take $10 from the loser's cash and add it to the winner's cash.
* decide if we are done.

HOW WOULD WE WRITE A PROGRAM TO MODEL THIS?

Ruin.m, using while(condition)

```
h_cash = 100;
t_cash = 80;
flips = 0;

while ( 10 <= h_cash && 10 <= t_cash )

  flips = flips + 1;
  ht = randi ( [ 0, 1 ] );

  if ( ht == 0 )
    t_cash = t_cash + 10;
    h_cash = h_cash - 10;
  else
    h_cash = h_cash + 10;
    t_cash = t_cash - 10;
  end

end

if ( h_cash < 10 )
  fprintf ( 'Terry won after %d flips!\n', flips );
else
  fprintf ( 'Haley won after %d flips!\n, flips' );
end
```

Ruin2.m Using a WHILE(TRUE)

We could also model the problem using a while(true) statement.

Then, if Terry won the toss, Haley loses $10. We need to check if this takes Haley's cash below $10, in which case she can't make another bet and the game is over (using a BREAK statement).

We make a similar check if Haley won the toss.

Ruin2.m, Using while(true)

```
h_cash = 100;
t_cash = 80;
flips = 0;

while ( true )

  flips = flips + 1;
  ht = randi ( [ 0, 1 ] );

  if ( ht == 0 )
    t_cash = t_cash + 10;
    h_cash = h_cash - 10;
    if ( h_cash < 10 )
      fprintf ( 'Terry won after %d flips!\n', flips );
      break;
    end
  else
    h_cash = h_cash + 10;
    t_cash = t_cash - 10;
    if ( t_cash < 10 )
      fprintf ( 'Haley won after %d flips!\n', flips );
      break;
    end
  end

end
```

Ruin3.m, Gambler's Ruin

This is an example of what, in probability, is called "gambler's ruin".

There are mathematical formulas that give the probability for either player to win.

The average number of steps can be determined.

How much advantage does a player with more money have?

Is a player with $20 twice as likely to win as an opponent with $10? NO, actually THREE times more likely. Program "ruin3.m" lets you input h_cash and t_cash, so you can check this!

Duel.m, In Class Exercise

```
a_acc = input ( 'Enter accuracy between 0 and 1 for A: ' );
b_acc = input ( '                          and for B: ' );

shots = 0;

while ( true )

  shots = shots + 1;
  x = rand ( );
  if ( x <= a_acc )
    fprintf ( '  Player A won after %d shots were fired!\n', shots );
    break;
  end

  shots = shots + 1;
  x =rand ( );
  if ( x <= b_acc )
    fprintf ( '  Player B won after %d shots were fired!\n', shots );
    break;
  end

end
```

In Class Exercise

How would you modify your duel program so that you could run N = 100 or 1000 duels?

Count the wins by A, the wins by B.

Then estimate:

Probability A wins = #A wins / N

Probability B wins = # B wins / N

Homework #2

hw0091: Using a FOR loop and the RAND function, estimate the average distance between two random points in the unit square.

hw016: print greatest common divisor tables using nested FOR loops.

hw023: Estimate the area under the curve y=humps(x), for x between 0 and 1;