Intro Math Problem Solving
September 12

Questions

In-class Exercise

WHILE Loops

Questions

Question: I typed "n=input('Enter child's age:')
but MATLAB is confused by the apostrophe.

To use apostrophe or single quote in a string, say it TWICE:

```
    N = input ( 'Enter child''s age:' )
```

It looks strange, but will print correctly.

To use a percent sign in a string, say it TWICE.

```
    fprintf ( '  The interest rate is 4%%'\n' );
```

See "apostrophe.m"

In Class Exercise

Work on the following problem during class, and we will discuss our solutions at the end:

How many steps must I take, in summing up integers 1, 2, 3, 4, 5, …, until I reach a total of 100 or more?

To use a for loop for this problem, we'd have to guess an upper limit on the number of steps. We would like a solution that doesn't have to guess, but simply carries out the repetition – an indefinite number of times – until the desired goal is reached.

Sometimes a FOR Loop Doesn't Help

- We know a FOR loop allows us to specify that a set of commands are to be carried out repeatedly, with an index (such as "i") keeping track of which step we are on.
- We made FOR loops more flexible by adding a BREAK statement, that lets us decide to terminate the process early if we notice we have finished early.
- But there are simple, interesting problems that the FOR loop can't handle well.

Paying Debts with a FOR Loop

Suppose I want to play a game that costs $10 a round.

To simulate what's going to happen, I have to specify how many games I plan to play.

```
my_balance = 123.45;
cost = 10.0;
for round = 1 : 8
  my_balance = my_balance - cost;
end
fprintf ( 'I still have $%10.2f\n', my_balance );
```

Gambling til you're broke

- But if I'm gambling, and I plan to play until I double my money or go broke, then this is also a case where I want to repeat some commands many times.
- Instead of saying how many times, I need to say "until something happens"… that is, until I go broke or double.
- Let's sketch a pseudo-MATLAB code.

Double or Nothing (pseudo-MATLAB)

We will play over and over until we have made our goal (doubling), or can't pay for a round. We think of our program like this:

```
stake = 123.45;
goal = 2 * stake;
cost = 10.0;
AS LONG AS ( cost <= stake AND stake < goal )
  stake = stake - cost;              ← pay
  stake = stake + 20 * rand ( ); ← play
end
```

The WHILE Statement

The WHILE statement works as follows:

- If a condition is true, execute the statements inside the WHILE loop.
- If the condition is STILL true, execute the statements again, and again...
- WHILE lets us repeat an indefinite number of times, but requires us to control how the loop will terminate.

double_or_nothing.m (legal MATLAB)

The WHILE statement lets us play indefinitely, until we make our goal or can't pay for another round.

```
stake = 123.45;
goal = 2 * stake;
cost = 10.0;
while ( cost <= stake && stake < goal )
  stake = stake - cost;             ← pay
  stake = stake + 20 * rand ( ); ← play
end
```

What's the base-2 logarithm of X?

Can we find the smallest power of 2 that is greater than or equal to X? That's almost the logarithm base 2 of X (It's the integer part of the log, rounded up.) Let's call it "intlog2(X)":

X	intlog2(x)
1	0, because 1 <= 2^0 = 1
10	4, because 10 <= 2^4 = 16
100	7, because 100 <= 2^7 = 128
123456789	27, because 2^27 = 134217728

How would we compute intlog2(x)?

We seek P, the smallest power of 2 so that P >= X.

We could start with P=1, and set a counter I = 0.

(We don't know how many steps we will need so we can't use a FOR statement.)

"AS LONG AS" our P is smaller than X…

we need to double: P=P*2

and increase our counter I,

and try again.

When we reach a value P >= X, the value of I is our answer.

Sample Calculation

- X = 56
- I = 0, P = 1 < 56 so try again
- I = 1, P = 2 < 56
- I = 2, P = 4 < 56
- I = 3, P = 8 < 56
- I = 4, P = 16 < 56
- I = 5, P = 32 < 56
- I = 6, P = 64 >= 56, STOP,
- INTLOG2(56) = 6 because 2^5 < 56 <= 2^6

INTLOG2 (Pseudo-MATLAB)

```
Input X from user
Initialize counter: I = 0,
Initialize power of 2: P = 1

AS LONG AS ( P < X )
  P = P * 2;  ← double P
  I = I + 1;  ← increase I
end of statements to repeat

Then INTLOG2(X) is I
```

intlog2.m (legal MATLAB)

```
x = input ( '  Enter value of X: ' );
i = 0;
p = 1;
while ( p < x )
  p = p * 2;
  i = i + 1;
end
fprintf ( 'intlog2(%f) = %d\n', x, i );
```

What if X < 1?

If X is less than 1, then we need to look at negative powers of 2.

So we start at P=1, I=0, and DIVIDE P by 2 and subtract 1 from I, repeatedly,

As soon as P < X, we know 2*P >= X and is the smallest power to do that.

So INTLOG2(X) is I+1 (the previous I).

Sample Calculation for X < 1

X = 0.04

- I = 0, P = 1 > 0.4 so try again
- I = -1, P = 1/2 = 0.5 > 0.04
- I = -2, P = $\frac{1}{4}$ = 0.25 > 0.04
- I = -3, P = 1/8 = 0.125 > 0.04
- I = -4, P = 1/16= 0.0625 > 0.04
- I = -5, P = 1/32 = 0.03125 < 0.04, so STOP

INTLOG2(0.04) = -5+1=-4 (have to back up 1)

Because 2^(-5) < P <= 2^(-4).

intlog2.m (for ALL positive X)

```
x = input ( '  Enter value of X: ' );
i = 0;
p = 1;

if ( x < 1 )

  while ( x <= p )
    p = p / 2;
    i = i - 1;
  end
  i = i + 1;  %  Back up one level!

else if ( 1 < x )

  while ( p < x )
    p = p * 2;
    i = i + 1;
  end

end
```

When will my bank account be $1,000,000

In the sixth episode of “Futurama”, Philip J Fry discovers that his bank account of $0.93 in the year 1999 has grown to $4.3 billion in the 31st century.

A typical bank interest rate is 1.2%.

Each year, my savings multiply by 1.012.

If I have $1 in my account today, how soon will I have $1,000,000?

Strategy (pseudo-MATLAB)

```
Initially, I have $1 and the year is 2017

"AS LONG AS" I have less than $1,000,000
  Go to next year
  Add 1.2% to my current funds
End of statements to repeat

Print the year and amount
```

interest.m

```
amount = 1.00;
year = 2017;
while ( amount < 1000000 )
  year = year + 1;
  amount = 1.012 * amount;
end
fprintf ( 'Year %d, I have $%f\n', year,
   amount );
```

Alternating, Decreasing Series

Some mathematical functions can be defined by an infinite series, an endless sum of terms:

e^x = 1 + x + x^2/2 + x^3/6 + x^4/24 + ...

If successive terms have opposite signs, the series is said to alternate.

If each term is smaller (in absolute value) than the previous one, the series is said to be monotonically decreasing.

The error of a finite sum

Suppose a mathematical quantity can be expressed as an alternating, decreasing series, S = a1 - a2 + a3 – a4 + … - an …

We might estimate the value of S by adding the first n terms and stopping.

For alternating decreasing series, the error of such an estimate is no greater than the size of the next term.

An alternating decreasing series

S = 1 – 1/2 + 1/3 – 1/4 + 1/5 - ...

This series is alternating.

The terms decrease in size.

Its exact value is log(2).

Here are estimates for log(2):

1	= 1	error no more than 1/2
1-1/2	= 1/2,	error no more than 1/3
1-1/2+1/3	= 2/3,	error no more than 1/4

Estimate S with given accuracy

Suppose we want to estimate S, with an accuracy of at least 0.00001?
Adding no terms at all, our estimate S = 0, and our error is no more than 1.
Adding term #1, our estimate S = 1, and our error is no more than 1/2
Adding term #2, our estimate S = 1/2, and our error is no more than 1/3.
So a procedure to estimate S with accuracy ACCURACY might be:

```
Start S at 0, and set A to term #1.

AS LONG AS ( ACCURACY < | A | )
  S = S + A.
  A = next term.
end

Print S, the estimated value.
Print A, the estimated error.
```

Keeping track of the terms

In our loop, we compute the next term, *A*

On the N-th step, N is 1/N or -1/N.

One way to remember this is that, if N is odd, *A* will be positive, but if N is even, *A* will be negative.

```
if ( mod (n,2) == 0 )
  a = 1 / n;
else
  a = -1/n;
end
```

log2_series.m

```
acc = 0.00001;
s = 0.0;
n = 1;
a = 1 / n;

while ( acc < abs ( a ) )

  s = s + a;
  n = n + 1;
  if ( mod ( n, 2 ) == 1 )
    a = 1 / n;
  else
    a = - 1 / n;
  end

end

fprintf ( '  Estimate for log(2) = %16.10f\n', s );
fprintf ( '  Estimated  error = %16.10e\n', a );
```

In Class Exercise

```
sum = 0;                  % ← our sum;
i = 0;                    % ← number to add;
while ( sum < 100 )  %  ← Another step?
  i = i + 1;              % ← increase number
  sum = sum + i           % ← Add it.
end
fprintf ( '  Step %d, sum is %d\n', i, sum );
```

(We see that after step 14, the sum is 105.)