Intro Math Problem Solving
September 19

Question

Exercise

Formula Sequences

Interest and Population

The Cannonball Problem

Lists and Plots

Computing "Until" an Event

Question

Q: In our homework, we used the function humps(x). I'd really like to see this function!

Mathematical answer:

y=1/((x-0.3)^2+0.01)+1/((x-0.9)^2+0.04)–6

Computational answer:

Plot y=humps(x) for 0 <= x <= 1.

See: humps_plot.m

In Class Exercise

For years, Mexico had a population growth rate of about 3%.

Write a script that uses the growth rate and the population in 1950, to approximate data at 1960, 1970, and 1980.

Instead of 3%, can you find a rate like 2.9% or 3.1%, which matches the data better?

Using that rate, in what year would Mexico pass 100,000,000?

1950	25,791,017
1960	34,923,129
1970	48,225,238
1980	66,846,833

Sequences

Sequences are an important mathematical idea, inspired by the sequence of natural numbers:

1, 2, 3, 4, 5, …

A typical sequence is an infinite list of numbers. Even though we can't write down the complete list, we understand that there is some rule or pattern that describes every entry.

Sometimes, it is enough to write the beginning of the sequence, and anyone can understand the pattern.

Simple (but boring) Sequences

1, 2, 3, 4, 5, …

2, 4, 6, 8, 10, …

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, …

1, 4, 9, 16, 25, …

1, 3, 9, 27, 81, 343, …

1892, 1896, 1904, 1908, 1912, …

Puzzling Sequences

PAS: 1, 7, 21, 35, 21, 7, 1, (END)

+N*N: 1, 2, 2, 4, 8, 11, 33, 37, 148, 153, …

FIB: 1, 1, 2, 3, 5, 8, 13, 21, 34, …

PAD: 1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12, 16, 21, …

ONE1: 1, 11, 21, 1211, 111221, 312211, …

Formula Sequences

a(i) = 2^(i-1): 1, 2, 4, 8, 16, 32, …

a(i) = (i+1)/(2i^2): 1, 3/8, 2/9, 5/32, …

a(i) = (i+1)*i/2: 1, 3, 6, 10, 15, 21, …

a(i) = i^2+3i-10: -6, 0, 8, 18, 30, …

a(i) = 1*2*…*i: 1, 2, 6, 24, 120, 720,…

a(i) = 1+i*2^i: 3, 9, 25, 65, 161, …

a(i) = i*(i-1)*(i-2)/6: 0,0,1,4,10,20,…

Computing a Formula Sequence

To study a sequence defined by a formula, we begin by printing the first 5 or 10 entries.

Some sequences start with entry a(0), others with entry a(1), so we need to know that.

To print entry a(i), we simply evaluate the formula. To evaluate entries a(0) or a(1) through a(n), we can use a FOR loop.

Assuming N is small, we can print the values on one line (print a NEWLINE at the end).

powers_of_2_sequence.m

Formula: a(i) = 2^i

Start at a(0), compute to a(10).

```
n = input ( 'Enter maximum index: ' );
for i = 0 : n
  fprintf ( ' %d', 2^i );
end
fprintf ( '\n' );
```

ichoose3_sequence.m

a(0) = 0

a(i) = i*(i-1)*(i-2)/6

Number of ways to choose 3 items from set of I items.

```
n = input ( 'Enter maximum index: ' );
for i = 0 : n
  ichoose3 = i*(i-1)*(i-2)/6;
  fprintf ( ' %d', ichoose3 );
end
fprintf ( '\n' );
```

interest_sequence.m

```
rate = input ( 'Enter interest rate: ' );
final_year = input ( 'Enter final year: ' );
initial_year = 2017;
initial_value = 1000.0;


for year = initial_year : final_year
  value = initial_value * ( 1.0 + rate )^(year-initial_year);
  fprintf ( '%d  %.2f\n', year, value );
end
```

Exercise

The interest rate example is very similar to the Mexican population example.

The population growth rate works the same as the interest rate.

The 1950 population works like the initial value in the bank account.

You should be able to run your program from 1950 to 1980, setting rate to 0.03, and approximate the data.

Sequence of Cannonball Positions

Suppose a cannonball is fired at an angle alpha, with a velocity V0 ft/sec, from an initial position (x0,y0) ft.

For t=0, 1, 2, 3, ... seconds, we can produce a sequence of positions:

$x(t) = x0 + v0 * \cos(alpha) * t$

$y(t) = y0 + v0 * \sin(alpha) * t - 16\ t^2$

Now we have a pair of values to track.

Planning the Program

On each step, we want to print T, and (X,Y).

We need a lot of input from the user:

- The cannon angle in degrees;
- The initial velocity in feet per second.
- The initial position (x0,y0) in feet;
- The number of time steps, in seconds.

We will need to convert the angle from degrees to radians, because MATLAB's sin and cos expect radian measure.

To get radians, divide degrees by 180 and multiply by pi.

cannon_sequence.m

```
degrees = input ( 'Enter cannon angle (degrees): ' )
angle = degrees * pi / 180;
v0 = input ( 'Enter initial velocity (ft/sec): ' );
x0 = input ( 'Enter initial x position (ft):' );
y0 = input ( 'Enter initial y position (ft):' );
tmax = input ( 'Enter max time (secs): ' );

for t = 0 : tmax
  x = x0 + v0 * cos ( angle ) * t;
  y = y0 + v0 * sin ( angle ) * t - 16 * t^2;
  fprintf ( '%d  %g  %g\n', t, x, y );
end
```

Detecting Maximum Value

Most of the sequences we've looked at so far simply increase, but we know that the height (y value) of the cannonball probably goes up and then down.

Therefore, it may be important to know the maximum y value observed.

To do this, we have to:

* create a new variable: "y_max"
* initialize it somehow: y_max = y0;
* compare it to each y as we go: ymax=max(ymax,y);
* print it at the end

cannon_max.m

```
ymax = y0;

for t = 0 : tmax
  x = x0 + v0 * cos ( angle ) * t;
  y = y0 + v0 * sin ( angle ) * t - 16 * t^2;
  fprintf ( '%d  %g  %g\n', t, x, y );
  ymax = max ( ymax, y );
end

fprintf ( 'Maximum height = %g\n', ymax );
```

Think About This!

Suppose I also want the time step when the maximum height occurred, which we might call **tpeak**.

To do this, we probably have to replace

"ymax = max (ymax, y)"

by something like:

```
if ( condition )
  do some things
end
```

Can you make this happen?

What about a Picture?

The cannonball sequence is like a series of snapshots of position. Seeing a plot of these positions would help us to believe the numbers we computed.

MATLAB has a build in PLOT() function that is perfect for this, but "officially" we don't know enough to use it.

Let's get a preview, learning just enough to make our plot.

MATLAB Lists

In Matlab, you can make a lists of data.

A list is enclosed in square brackets: [].

The list must have a name, the same way that variables have names: "xlist".

The simplest list contains nothing:

```
xlist = [];
```

If you know what goes inside, you can do this:

```
xlist = [ 1, 3, 43, 12 ];
```

But for our sequences, we will start with an empty list, compute a new value and add it to the end:

```
xlist = [ xlist, x ];
```

List the Cannonball data!

```
xlist = [];          ← start a list called "xlist"
ylist = [];          ← start a list called "ylist"
for t = 0 : tmax
  x = x0 + v0 * cos ( angle ) * t;
  y = y0 + v0 * sin ( angle ) * t - 16 * t^2;
  fprintf ( '%g  %g  %g\n', t, x, y );
  xlist = [ xlist, x ];   ← add this "x" to the list
  ylist = [ ylist, y ];   ← add this "y" to the list
end
```

Lists can be printed

Matlab treats xlist and ylist just like other variables.

In particular, we can print out their values just by typing their names.

Actually, the range of a FOR loop, like 1:n or initial_year:final_year is also a list.

See what happens if we ask MATLAB to print "0:tmax".

Lists can be plotted!

The MATLAB plot command has the form

plot(xlist,ylist)

where x and y are lists of equal length.

Three plots worth making:

```
plot ( 0:tmax, xlist );     ← Just the X history
plot ( 0:tmax, ylist );     ← Just the Y history
plot ( xlist, ylist );      ← The path of the cannonball.
```

You can label a plot by adding:

```
title ('What you want the title to be' )
```

cannon_plot.m

```
xlist = [];
ylist = [];

for t = 0 : tmax

  x = x0 + v0 * cos ( angle ) * t;
  y = y0 + v0 * sin ( angle ) * t - 16.0 * t^2;

  fprintf ( '%d seconds,  (%g, %g) feet\n', t, x, y );
  xlist = [ xlist, x ];
   ylist = [ ylist, y ];

end

plot ( xlist, ylist );
title ( 'Cannonball trajectory' );
```

Computing a Sequence "Until"

We'd like a way to compute the sequence of cannonball positions until it reaches y=0.

We have seen that some kind of WHILE statement is the way to do this.

Let's see if we can modify our program to compute just as many steps as necessary.

Because we don't use a FOR loop, we have to initialize and update T, and we have to use a BREAK statement to terminate.

Compute until "impact"
cannon_until.m

```
t = 0;                          ← We have to initialize T

while ( true )                  ← Keep repeating, we'll check elsewhere

  x = x0 + v0 * cos ( angle ) * t;
  y = y0 + v0 * sin ( angle ) * t - 16 * t^2;
  fprintf ( '%d  %g  %g\n', t, x, y );

  if ( y <= 0.0 )               ← Check for impact.
    break;
  end

  t = t + 1;                    ← We have to increase T
.
end
```

Working with the New Code

Assuming we have set things up correctly, our "cannonball_until" program will accept the user data, compute a sequence of (x,y) values, and stop as soon as a negative y is noticed.

So the results for the input

angle = 45, v0 = 500, x0 = 0, y0 = 200

will show a nice arc up and down. Suppose the cannon is firing from a pit, say at elevation y=-20? How does our program work? Can we fix it?

Formula Sequences Summary

A formula sequence allows us to compute the i-th element of a sequence a(i).

To print up to the N-th element, a FOR loop I=0:N or I=1:N is a good choice.

If we are looking for the first sequence element that has some property, we need to use a WHILE statement, and a WHILE(TRUE) may be the better choice.

If we store sequence elements in a list, we can plot them:

plot (0:n, ichoose3_list) ← our subset sequence

plot (xlist, ylist) ← our cannonball sequence

Did You Get It?

Can you modify the ichoose3 code to

A) save the first 6 elements in a list, and plot them?

B) compute the first value of I for which the sequence element is more than 1000?

```
n = input ( 'Enter maximum index: ' );
for i = 0 : n
  ichoose3 = i*(i-1)*(i-2)/6;
  fprintf ( ' %d', ichoose3 );
end
fprintf ( '\n' );
```

Exercise

For years, Mexico had a population growth rate of about 3%.

Write a script that uses the growth rate and the population in 1950, to approximate data at 1960, 1970, and 1980.

Instead of 3%, can you find a rate like 2.9% or 3.1%, which matches the data better?

Using that rate, in what year would Mexico pass 100,000,000?

1950 25,791,017
1960 34,923,129
1970 48,225,238
1980 66,846,833
1990 81,249,645
2000 97,483,412
2010 113,580,528