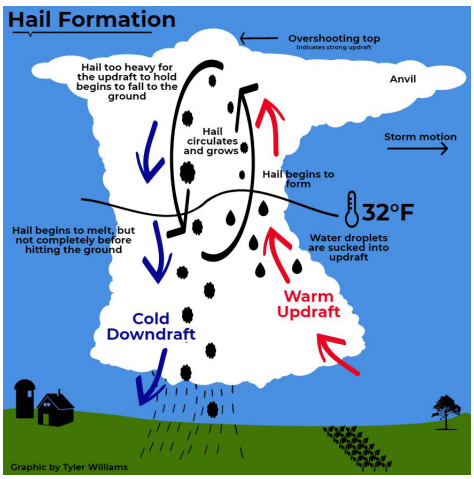
MATLAB Introduction

MATH1070: Numerical Mathematical Analysis

Location: http://people.sc.fsu.edu/~jburkardt/classes/math1070_2019/matlab_intro/matlab_intro.pdf

---



Hailstone numbers cycle for a while and then drop unpredictably!

MATLAB Intro

Let's model a particular problem, and see how MATLAB can help us.

1 The HUMP function

Suppose we are interested in the function:

$$y = \frac{1}{(x-0.3)^2 + 0.01} + \frac{1}{(x-0.9)^2 + 0.04} - 6.0$$

We can create a MATLAB function *hump.m*, so that we only have to type this formula correctly once. Note that division is done with a period-slash, squaring is done with a "period-caret", and we have to be careful about the parentheses:

```
function y = hump ( x )
  y = 1.0 ./ ( ( x - 0.3 ).^2 + 0.01 ) ...
    + 1.0 ./ ( ( x - 0.9 ).^2 + 0.04 ) ...
    - 6.0;
  return
end
```

Then we can ask for the value at 2.0 by writing `y=hump(2.0)`, or the value at a variable `x` by writing `y=hump(x)` and so on.

Suppose we wanted to know the value of the function at 6 equally spaced points in the interval $[0, 1]$? We could type the command 6 times, but a better option would be to somehow get all the answers at once. We can create a list of the x values and get our answers back like this:

```
x = [ 0, 0.2, 0.4, 0.6, 0.8, 1.0 ];
y = hump ( x )
```

We have just learned that we can create a list, or vector, by enclosing a sequence of values in square brackets. Now suppose we wanted to get a similar table of values between 1 and 2. We can write

```
x = x + 1.0;
y = hump ( x )
```

That is, we can do simple arithmetic to all the numbers in a vector, using the same commands we are familar with for normal arithmetic, like

```
y = 2 * x
z = x - sin ( x )
```

The other thing to notice is that $[0, 0.2, 0.4, 0.6, 0.8, 1.0]$ is a **row vector**. It has dimensions 1×6, 1 row and 6 columns. If we had wanted to create a column vector, we would separate our list by semicolons:

```
w = [ 0; 0.2; 0.4; 0.6; 0.8; 1.0]
```

Notice that you can't add the vectors `x` and `w`, because they have different shapes. You can use the transpose operator, an apostrophe, to flip between the row and column formats. Look at

```
v = w'
```

Suppose we wondered about the minimum and maximum values of our *hump(x)* function over the interval $[0, 2]$. A good way to estimate this would be to sample the function at a lot of places. Instead of creating a very long `x` vector, we can use the `linspace()` function:

```
x = linspace ( 0.0, 2.0, 101 );
y = hump ( x );
min ( y )
max ( y )
mean ( y )
```

Finally, to actually see our function, we can plot the `x` and `y` values:

```
plot ( x, y )
print ( '-dpng', 'hump.png' )  % Saves a copy of your plot.
```

There are several ways to make this plot nicer, which we will see in the plotting lab.

2 Let's make hail!

Start with 10. Because it's even, drop it to 5. Because that's odd, triple it and add 1 to get 16. Repeatedly divide by 2 to get 8, 4, 2, 1. We just did a hailstone sequence: start with any integer, get the next number by halving an even value, or if odd, tripling and adding 1.

This seems like a task the computer can do. It would be convenient to have a function `hail_sequence(n)` which is given our starting value and prints out the sequence as it is computed. How can we set this up? Our idea would be something like this:

```
function hail_sequence ( n )     % Note: we have NO output

  if n is 1, we are done

    if n is even, divide it by 2
    or if n is odd, multiply it by 3 and add 1.

  otherwise repeat
```

To determine if `n` is even, we can call `mod(n,2)`, which returns the remainder when `n` is divided by 2. If this is 0, then it's an even number.

We use `if(condition)` and `else` statements to control what we do to `n`.

How do we handle the repetition? The MATLAB `for` statement lets us repeat something a fixed number of times. But we want to repeat something until something happens, which could be any number of repetitions. For that, we need the `while()` statement, which has the form:

```
while ( condition )
  statements to be repeated
end
```

Simple conditions include:

```
n == 1    n is equal to 1
n ~= 2    n is NOT equal to 2      % That's the "twiddle" character
n < 3     n is less than 3
n >= 4    n is greater than or equal to 4
```

So our function looks like this:

```
function hail_sequence ( n )

  n                     % Print the first value

  while ( n ~= 1 )

    if ( mod ( n, 2 ) == 0 )
      n = n / 2
    else
      n = 3 * n + 1
    end

  end

  return
end
```

Does `hail_sequence()` work? Let's try it on $n = 7, 10, 12, 19$.

3 How long is a hailstone's journey?

It's interesting that the number of steps varies so much from one starting point to the next. Can we find an easy way to record this information? Suppose we modify *hail_sequence()* so that it doesn't print the values that are computed, but just counts them, and returns that as output?

```
function count = hail_count ( n )

  count = 0;

  while ( n ~= 1 )

    if ( mod ( n, 2 ) == 0 )
      n = n / 2;
    else
      n = 3 * n + 1;
    end

    count = count + 1;

```

```
  end

  return
end
```

Now we can use a `for` loop to make a table of these values.

We can even make a plot. We don't want to connect the successive values, but just show them as dots. We use a modified version of the plot command:

```
  x = zeros ( 150, 1 );
  y = zeros ( 150, 1 );

  for n = 1 : 150
    x(n) = n;
    y(n) = hail_count ( n )
  end

  plot ( x, y, 'bo' )     % 'bo' means plot data using blue circles.

  print ( '-dpng', 'hail_count_plot.png' )  % Save a copy of your plot
```