Introduction to Mathematica

Applications to Cryptology

General Comments and Methods

- First moments

When you first start Mathematica, you will get a white window with nothing in it. Actually, at the very top of this window there is a horizontal line running all the way across: this is a cursor. Type: "This is a sampling of text."

This is a line of text.

Now cursor-down to get another horizontal cursor, and enter "2+2" (without the quote marks), and press either SHIFT+ENTER, or press ENTER on your numeric keypad. If you did that as I intended, you'll now see the answer, "4", and several blue cell-markers along the right edge of the screen:

\[
\text{In}[1]:= 2 + 2
\]

\[
\text{Out}[1]= 4
\]

Now cursor back up to your command "2+2" and change it to two-hundred-factorial: "200!". It will look something like this:

\[
\text{In}[2]:= 200!
\]

\[
\text{Out}[2]= 788657867364790503552363213932185062295135977687173263294742533244359449963403342
920304284011984623904177212138919638830257642790242637105061926624952829931113462
857270763317237396988943922445621451664240254033291864131227428294853277524242407
57390324032125740557956866022603190417032406235170085879617892222789623703897374
72000000000000000000000000000000000000000
\]

Mathematica tries very hard to answer with the same precision that you used in your input. Since "200" is an infinite precision number, an exact integer, Mathematica is loathe to answer with less than infinite accuracy. Fiddle around a bit, and try to find the largest number that Mathematica can take the factorial of.

Sometimes infinite precision is too much precision. You can convert numbers to a more conventional numerical representation using the function "N". Here's an example of the notation:

\[
\text{In}[3]:= N[200!]
\]

\[
\text{Out}[3]= 3.316275092450633 \times 10^{5735}
\]

This reveals that 200! has 5736 digits. And here are 100 digits of π:

\[
\text{In}[4]:= N[\pi, 100]
\]

\[
\text{Out}[4]= 3.1415926535897932384626433832795028841971693993751058209749445923078164062862089986280348
25342117068
\]
# Conventions

The most important conventions are that arguments are passed to functions using square brackets, "=" is used only to assign a value and not in equations, and all built-in Mathematica commands start with a capital letter. For example, the most important numbers in mathematics are Pi and E. The sine function (opposite over hypotenuse) is denoted Sin[x].

There are several ways to input things. Through the "Palettes" menu, select "BasicMathInput". A separate window should appear with around 100 buttons. Select the definite integral button to start inputing the following calculation. Press SHIFT+ENTER when you have it ready:

\[
\int_0^\pi \text{ArcTan}\left(x^3\right) \, dx
\]

\[
\frac{1}{12} \left( 6 \sqrt{3} \text{ArcCot}\left(\frac{-1 + 2 \pi^2}{\sqrt{3}}\right) - 4 \pi \left(\sqrt{3} - 3 \text{ArcTan}\left[\pi^3\right]\right) + 6 \log\left[1 + \pi^2\right] - 3 \log\left[1 - \pi^2 + \pi^4\right]\right)
\]

I don't know about you, but I couldn't do that integral. Notice that the output is in infinite precision, just like the input. I wonder how big that number is...

\[
\text{N}[%] = 3.17166
\]

There's another nice convention: the percent sign always refers to the most recent output.

# Typesetting

One of the nicest things about using Mathematica is that it can intersperse text with calculations. You can always scroll back up and change your input, press SHIFT+ENTER, and continue on. Select the blue cell-marker on the right of your "This is a line of text." cell. Then, through the Format/Style menu, select "Text". You'll see the font and spacing improve. When we read, we like letters more dense than formulas.

Also, take note of the "100%" on the lower right of your window. Change that to "125%" and your eyesight will be relieved, depending on your monitor.

# Plotting functions

Here're a couple of examples of how to plot a function with Mathematica.

First, we define the function with the following notation:

\[
f[x_] = x^2 + \sin[10 \, x]
\]

\[
x^2 + \sin[10 \, x]
\]

Yes, that is an underscore after the x; it signals that "x" is a dummy variable.
That's nifty. We can see the effect of changing the frequency by fiddling with the slider bar in the following image.

- Modular Arithmetic

*Mathematica* can do modular reduction, but it doesn't do it intelligently. You and I should see immediately that $2601^{13432} + 26 \times 3456723^{39234563} \mod 26$ is 1. The following *Mathematica* command does the exponents first, and then the reduction:

\[
\text{In[10]:=} \quad \text{Mod}[2601^{13432} + 26 \times 3456723^{39234563}, 26] // \text{Timing}
\]

\[
\text{Out[10]=} \quad \{57.329, 1\}
\]

It takes my laptop 60.218 seconds to get the answer.

Since modular exponentiation is such a common task that can go so much faster, there's a special command for it: PowerMod
\[\text{In}[11] = \text{Mod}[\text{PowerMod}[2601, 13432, 26] + 26 \times \text{PowerMod}[345623, 39234563, 26], 26] // \text{Timing}\]

\[\text{Out}[11] = \{5.82867 \times 10^{-16}, 1\}\]

Instantaneous.

It is worth noting that if the first argument to Mod is a list, then Mod is applied to each element of the list.

\[\text{In}[12] = \text{Mod}[\{10, 20, 30, 40, 50\}, 26]\]

\[\text{Out}[12] = \{10, 20, 4, 14, 24\}\]

This means that we can implement Caesar encryption in the following way: convert the message to a list of numbers, add something to the list (it will add to each entry of the list), and then Mod26 the list (one command will handle the entire list), and then convert the list of numbers back to letters.

We could define modular arithmetic so that, instead of 0,1,...,25, we get a number in 1,2,...,25,26. We can do that in Mathematica by specifying one more number in Mod:

\[\text{In}[13] = x = \text{Range}[30]\]

\[\text{Mod}[x, 26]\]

\[\text{Mod}[x, 26, 1]\]

\[\text{Out}[13] = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30\}\]

\[\text{Out}[14] = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30\}\]

\[\text{Out}[15] = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 1, 2, 3, 4\}\]

### Lists and Strings

A list looks like the following

\[\text{In}[16] = \text{myfirstlist} = \{11, 290, -311, x^2 + \text{Exp}\[x\], "alphabet"\}\]

\[\text{Out}[16] = \{11, 290, -311, e^x + x^2, \text{alphabet}\}\]

Specifically, there's a brace, a sequence of things (could be numbers, expressions, functions, strings, whatever), followed by a closing brace. Here are the basic commands:
In[17]= Length[myfirstlist]
 myfirstlist[[4]]
 Position[myfirstlist, -311]
 MemberQ[myfirstlist, "alphabet"]
 MemberQ[myfirstlist, 200]
 Table[k (3 k + 1)
 2, {k, -3, 5}]

Out[17]= 5

Out[18]= e^x + x^2

Out[19]= {{3}}

Out[20]= True

Out[21]= False

Out[22]= {12, 5, 1, 0, 2, 7, 15, 26, 40}

A string is a bunch of characters, surrounded by quote marks.

In[23]= myfirststring = "Good night to all, and to all a good night!"

Out[23]= Good night to all, and to all a good night!

Some useful commands are the following (make sure you understand what each does):

In[24]= Characters[myfirststring]
 StringLength[myfirststring]
 ToLowerCase[myfirststring]
 StringPosition[myfirststring, "n"]
 StringPosition[myfirststring, "night"]

Out[24]= {G, o, o, d, , n, i, g, h, t, , t, o, , a, l, l, , ,
 a, n, d, , t, o, , a, l, l, , a, , g, o, o, d, , n, i, g, h, t, !}

Out[25]= 43

Out[26]= good night to all, and to all a good night!

Out[27]= {{6, 6}, {21, 21}, {38, 38}}

Out[28]= {{6, 10}, {38, 42}}

---

Cryptography

- Basic Manipulations

To do cryptography (and more interestingly, cryptanalysis), we need to handle messages. First, let's decide on the alphabet we'll use. Note that the whoever is setting up the cryptosystem (Alice and Bob) chooses the alphabet, and Eve has to figure out what they chose (usually it's obvious, though). We'll use the lower case english alphabet, in the usual order. (The dollar sign at the beginning of the variable name is customary to denote global variables.)
One
Message).

Now I'll define a function that will convert a string to a list of numbers. Here's the algorithm: convert the message to a list of individual letters using Characters, use Position to convert each letter to a number, and then use Flatten to get rid of the junk coming from characters that are in the message but aren't in our alphabet.

EncodeMessage[message_] :=
Module[{chars},
chars = Characters[message];
Flatten[Table[Position[$alphabet, chars[[i]]], {i, 1, Length[chars]}]]

DecodeMessage[listofnumbers_] :=
StringJoin[Table[$alphabet[[listofnumbers[[i]]]], {i, 1, Length[listofnumbers]]}]

Note that for decoding we don't need the Module construction between we don't use any local variables (like "chars" in Encode Message). It wouldn't hurt, but we don't need it.

- Example

First, some text. A fictional character of Heinlein's said something like this:

quotation = "Anyone who is not comfortable with mathematics is not fully human. At best, he is a tolerable subhuman who has learned to not make messes in the house."

Anyone who is not comfortable with mathematics is not fully human. At best, he is a tolerable subhuman who has learned to not make messes in the house.

First, we fix the upper case letters!

quotation = ToLowerCase[quotation]

anyone who is not comfortable with mathematics is not fully human. at best, he is a tolerable subhuman who has learned to not make messes in the house.

eq = EncodeMessage[quotation]

dq = DecodeMessage[eq]

It works!

One last command to convert a string to 4-grams.

StringGram[t_, blocklength_] :=
StringInsert[t, " ", Table[k blocklength + 1, {k, Ceiling[StringLength[t]/blocklength]}]]
In[37]:=  StringGram[dq, 4]

Out[37]=  anyo newh oisn otco mfor tabl ewit hmat hema tics isno tful lyhu mana tbes thei sato lera bles ubhu manw hoha slea rned tono tmak emes sesi ntthe hous e

Easier to copy, at least.

### Caesar's Cipher

Here's encryption: encode the message, add the key mod 26, then decode.

In[38]:=  EncryptCaesar[message_, key_] := Module[{numbers, encryptednumbers},
   numbers = EncodeMessage[message];
   encryptednumbers = Mod[numbers + key, 26, 1];
   DecryptMessage[encryptednumbers]];

In[39]:=  EncryptCaesar[quotation, 0]

Out[39]=  EncryptCaesar[quotation, 1]

Out[40]=  anyonewhoisnotcomfortablewithmathematicsisnotfullyhumanatbestthesatis tolerablesubhumanwhoas l earnedtonotmakemessesinthehouse

And now for decryption:

In[41]:=  DecryptCaesar[message_, key_] := EncryptCaesar[message, -key];

In[42]:=  eq = EncryptCaesar[quotation, 10]

Out[42]=  kxixyogryscxydmywpbydksvogdsrdkwdrkmdscscxydpevvirewxxkdlocdrosckdvyobklvocelrewkxgrykcv okbxondyxydwkuowocccosxdroryeco

Out[43]=  anyonewhoisnotcomfortablewithmathematicsisnotfullyhumanatbestthesatis tolerablesubhumanwhoasl earnedtonotmakemessesinthehouse

Bingo.

Now for the trickiest one: cryptanalysis of Caesar. 10 Challenge points for an explanation of why this works. The first thing we will need is a frequency table:

In[44]:=  FrequencyTable[text_] := Table[StringCount[text, $alphabet[[1]]], {i, 1, 26}];

In[45]:=  FrequencyTable[eq]

Out[45]=  {0, 3, 11, 12, 5, 0, 3, 0, 2, 0, 12, 4, 2, 1, 14, 2, 0, 10, 6, 0, 1, 6, 7, 9, 11, 0}

In[46]:=  AnalyzeCaesar[message_] := Module[{ourfrequencytable, basicfrequencytable, dots},
   ourfrequencytable = FrequencyTable[message];
   (* this is the frequency table coming from the US Constitution *)
   dots = Table[RotateLeft[ourfrequencytable, i].basicfrequencytable, {i, 1, 26}];
   Flatten[Position[dots, Max[dots]]]

In[47]:=  AnalyzeCaesar[eq]

Out[47]=  {10}