Number Systems III

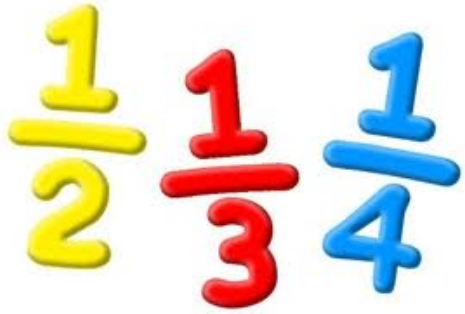
MA1S1

Tristan McLoughlin

December 4, 2013

http://en.wikipedia.org/wiki/Binary_numeral_system
http://accu.org/index.php/articles/1558
http://www.binaryconvert.com
http://en.wikipedia.org/wiki/ASCII

Converting fractions to binary



So far we have talked about integers both positive and negative.

We now look at a way to convert fractions to binary. You see if we start with, say

$$\frac{34}{5}$$

we can say that is $6 + \frac{4}{5}$. We know $6 = (110)_2$ and if we could work out how to represent $\frac{4}{5}$ as 0. something in binary then we would have

$$\frac{34}{5} = 6 + \frac{4}{5} = (110.\ \text{something})_2\ .$$

To work out what 'something' should be, we work backwards from the answer.

Say the digits we want are $b_1, b_2, b_3, \ldots$ and so

$$\frac{4}{5} = (0.b_1 b_2 b_3 b_4 \cdots)_2$$

We don't know any of $b_1, b_2, b_3, \ldots$ yet but we know they should be base 2 digits and so each one is either 0 or 1. We can write the above equation as a formula and we have

$$\frac{4}{5} = \frac{b_1}{2} + \frac{b_2}{2^2} + \frac{b_3}{2^3} + \frac{b_4}{2^4} + \cdots$$

If we multiply both sides by 2, we get

$$\frac{8}{5} = b_1 + \frac{b_2}{2} + \frac{b_3}{2^2} + \frac{b_4}{2^3} + \cdots$$

In other words multiplying by 2 just moves the binary point and we have

$$\frac{8}{5} = (b_1.b_2 b_3 b_4 \cdots)_2$$

Now if we take the whole number part of both sides we get 1 on the left and b_1 on the right. So we must have $\boxed{b_1 = 1}$. But if we take the fractional parts of both sides we have

$$\frac{3}{5} = (0.b_2b_3b_4\cdots)_2$$

We are now in a similar situation to where we began (but not with the same fraction) and we can repeat the trick we just did. Double both sides again

$$\frac{6}{5} = (b_2.b_3b_4b_5\cdots)_2$$

Take whole number parts of both sides: $\boxed{b_2 = 1}$. Take fractional parts of both sides.

$$\frac{1}{5} = (0.b_3b_4b_5\cdots)_2$$

We can repeat our trick as often as we want to uncover as many of the values b_1, b_2, b_3, etc as we have the patience to discover.

What we have is a method, in fact a repetitive method where we repeat similar instructions many times.

We call a method like this an *algorithm*, and this kind of thing is quite easy to programme on a computer because one of the programming instructions in almost any computer language is REPEAT (meaning repeat a certain sequence of steps from where you left off the last time).

In this case we can go a few more times through the steps to see how we get on.

Double both sides again.

$$\frac{2}{5} = (b_3.b_4b_5b_6\cdots)_2$$

Whole number parts: $\boxed{b_3 = 0}$. Fractional parts:

$$\frac{2}{5} = (0.b_4b_5b_6\cdots)_2$$

Double both sides again.

$$\frac{4}{5} = (b_4.b_5b_6b_7\cdots)_2$$

Whole number parts: $\boxed{b_4 = 0}$. Fractional parts:

$$\frac{4}{5} = (0.b_5b_6b_7\cdots)_2$$

This is getting monotonous, but you see the idea. You can get as many of the b's as you like.

In fact, if you look carefully, you will see that it has now reached repetition and not just monotony.

We are back to the same fraction as we began with $\frac{4}{5}$. If we compare

$$\frac{4}{5} = (0.b_5 b_6 b_7 \cdots)_2$$

to the starting one

$$\frac{4}{5} = (0.b_1 b_2 b_3 b_4 \cdots)_2$$

we realise that everything will unfold again exactly as before.

We must find $b_5 = b_1 = 1$, $b_6 = b_2 = 1$, $b_7 = b_3 = 0$, $b_8 = b_4 = 0$, $b_9 = b_5 = b_1$ and so we have a repeating pattern of digits 1100. So we can write the binary expansion of $\frac{4}{5}$ down fully as a repeating pattern

$$\frac{4}{5} = (0.\overline{1100})_2$$

and our original number as

$$\frac{34}{5} = (110.\overline{1100})_2$$

Floating point format storage

We have seen that in order to cope with numbers that are allowed to have fractional parts, computers use a binary version of the usual "decimal point". We called it a "binary point" as "decimal" refers to base 10.

Recall that what we mean by digits after the decimal point has to do with multiples of $1/10$, $1/100 = 1/10^2 = 10^{-2}$, etc. So the number 367.986 means

$$367.986 = 3 \times 10^2 + 6 \times 10 + 7 + \frac{9}{10} + \frac{8}{10^2} + \frac{6}{10^3}$$

We use the 'binary point' in the same way with powers of $1/2$. So

$$(101.1101)_2 = 1 \times 2^2 + 0 \times 2 + 1 + \frac{1}{2} + \frac{1}{2^2} + \frac{0}{2^3} + \frac{1}{2^4}$$

As in the familiar decimal system, every number can be written in binary using a binary point and as for decimals, there can sometimes be infinitely many digits after the point.

Binary Scientific Notation

What we do next is use a binary version of scientific notation.

The usual decimal scientific notation is like this

$$54321.67 = 5.432167 \times 10^4$$

We refer to the 5.4321 part (a number between 1 and 10 or between -1 and -10 for negative numbers) as the *mantissa*. The power (in this case the 4) is called the *exponent*. Another decimal example is

$$-0.005678 = -5.678 \times 10^{-3}$$

and here the mantissa is -5.678 while the exponent is -3.

This is all based on the fact that multiplying or dividing by powers of 10 simply moves the decimal point around. In binary, what happens is that multiplying or dividing by powers of 2 moves the 'binary point'.

$$\begin{aligned}(101)_2 &= 1\times 2^2 + 0\times 2 + 1\\ (10.1)_2 &= 1\times 2 + 0 + \frac{1}{2} = (101)_2 \times 2^{-1}\\ (1101.11)_2 &= (1.10111)_2 \times 2^3\end{aligned}$$

This last is an example of a number in the binary version of scientific notation. The mantissa is $(1.110111)_2$ and we can always arrange (no matter what number we are dealing with) to have the mantissa between 1 and 2. In fact always less than 2, and so of the form

$$1.\text{something}\ .$$

The exponent in this last example is 3 – the power that goes on the 2.

For negative numbers we would need a minus sign in front.

What can thus write every number in this binary version of scientific notation. That saves us from having to record where to put the binary point, because it is always in the same place. Or really, the exponent tells us how far to move the point from that standard place.

Computers then normally allocate a fixed number of bits for storing such numbers. The usual default is to allocate 32 bits in total (though 64 is quite common also). Within the 32 bits they have to store the mantissa and the exponent. The mantissa is already in binary, but we also need the exponent in binary.

So in

$$(1.10111)_2 \times 2^3$$

the mantissa is $+(1.110111)_2$ while the exponent is $3 = (11)_2$. Computers usually allocate 24 bits for storing the mantissa (including its possible sign) and the remaining 8 bits for the exponent.

In our example, 24 bits is plenty for the mantissa and we would need to make it longer to fill up the 24 bits: $(1.110111000\ldots)_2$ will be the same as $(1.110111)_2$. However, there are numbers that need more than 24 binary digits in the mantissa, and what we must then do is round off. In fact, we have to chop off the mantissa after 23 binary places (or more usually we will round up or down depending on whether the digit in the next place is 1 or 0).

Filling out the example number $(1.110111)_2 \times 2^3$ into 32 bits using this system, we might get:

0	1	1	1	0	1	1	1	0	...	0	0	...	0	1	1
1	2	3	4	5	6	7	8	9	...	24	25	...	30	31	32

We are keeping bit 1 for a possible sign on the mantissa and we also need to allow the possibility of negative exponents. For example

$$-(0.000111)_2 = -(1.11)_2 \times 2^{-4}$$

is negative and so has a negative mantissa $-(1.11)_2$. Because it is less than 1 in absolute value, it also has a negative exponent $-4 = -(100)_2$.

To be a bit more accurate about how computers really do things, they normally put the sign bit (of the mantissa) 'first' (or in their idea of the most prominent place), then put the 8 bits of the exponent next and the remaining 23 bits of the mantissa at the end.

So a better picture for $(1.110111)_2 \times 2^3$ is this:

0	0	$\cdots$	0	1	1	1	1	1	0	1	1	1	0	$\ldots$	0	0
1	2	$\cdots$	7	8	9	10	11	12	13	14	15	16	17	$\ldots$	31	32
$\pm$	exponent					mantissa less sign										

This is just explained for the sake of greater accuracy but is not our main concern.

The web site:
http://accu.org/index.php/articles/1558
goes into quite a bit of detail about how this is done.

What you get on
http://www.binaryconvert.com
(under floating point) tells you the outcome in examples but there are many refinements used in practice that are not evident from that and that also we won't discuss.

The method we have sketched is called *single precision floating point* storage. There are some details we have not gone into here(such as how to store 0).

Another common method, called *double precision*, uses 64 bits to store each number, 53 for the mantissa (including one for the sign) and 11 for the exponent.

Limitations of floating point

We can get an idea of the scope and accuracy allowed by these 'floating point numbers' (this means that the position of the binary point is movable, controlled by the exponent) .

The largest possible number we can store has mantissa

$$(1.111\ldots)_2$$

with the maximum possible number of 1's (in fact the usual system does not store the 1 before the binary point as it is always there) so we can manage 24 1's total, 23 places after the point and exponent as large as possible

Now $(1.111\ldots)_2$ (with 23 1's after the point) is just about 2 and the largest possible exponent is $2^7 - 1 = 127$. [We are allowed 8 bits for the exponent, which gives us room for $2^8 = 256$ different exponents. About half should be negative, and we need room for zero. So that means we can deal with exponents from -128 to $+127$.] Thus our largest floating point number is about

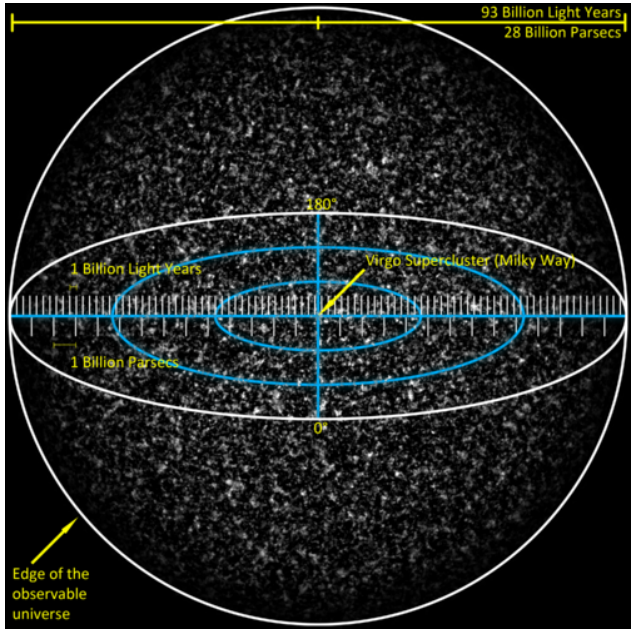
$$2 \times 2^{127} = 2^{128}$$

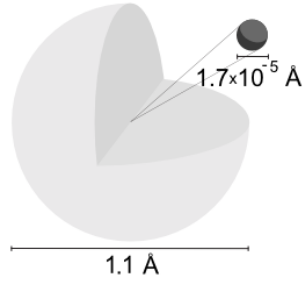
This is quite a big number and we can estimate it by using the $2^{10} \cong 10^3$ idea

$$2^{128} = 2^8 \times 2^{120} = 256 \times (2^{10})^{12} \cong 256 \times (10^3)^{12} = 256 \times 10^{36} = 2.56 \times 10^{38}$$

This is quite a bit larger than the limit of around 2×10^9 we had with integers. Indeed it is large enough for most ordinary purposes. For example the world GDP (in units of euro) is about 50 trillion (5×10^{13})

It is also large enough for most large numbers we find in science: for example an estimate of the number of hydrogen atoms in the observable universe is only 10^{80}.





That being said its easy to think of much larger numbers: a common one is the googolplex ($10^{10^{100}}$), while one number used by mathematician Stanely Skewes is

$$10^{10^{10^{34}}}.$$

While even larger numbers with even more iterated exponentiations, and iterated iterated exponentiation and so on, have appeared in mathematical proofs in graph theory but need a special notation to write conveniently.

We can also find the smallest positive number that we can store. It will have the smallest possible mantissa $(1.0)_2$ and the most negative possible exponent, -128. So it is

$$1 \times 2^{-128} = \frac{1}{2^{128}} = \frac{1}{2.56 \times 10^{38}} \cong \frac{4}{10} \times 10^{-38} = 4 \times 10^{-39}$$

or, getting the same result another way,

$$1 \times 2^{-128} = \frac{2^2}{2^{130}} = \frac{2^2}{(2^{10})^{13}} \cong \frac{4}{(10^3)^{13}} = 4 \times 10^{-39}$$

This is pretty tiny, tiny enough for many purposes.

In practice, there is a facility for somewhat smaller positive numbers with fewer digits of accuracy in the mantissa. These details are fairly well explained at http://accu.org/index.php/articles/1558.

Double Precision

If we use double precision (64 bits per number, requires twice as much computer memory per number) we get an exponent range from $-2^{10} = -1024$ to $2^{10} - 1 = 1023$. The largest possible number is

$$2^{1024} = 2^4 \times 2^{1020} = 16 \times (2^{10})^{102} \cong 16 \times (10^3)^{102} = 1.6 \times 10^{307}$$

and the smallest is the reciprocal of this.

In both single and double precision, we have the same range of sizes for negative numbers as we have for positive numbers.

So the limitation on size are not severe limitations, rather the key consideration is the limit on the accuracy of the mantissa imposed by the 24 bit limit (or the 53 bit limit for double precision).

The difficulty is essentially not with the smallest number we can store but with the next biggest number greater than 1 we can store. That number has exponent 0 and mantissa $(1.000\ldots01)_2$ where we put in as many zeros as we can fit before the final 1. Allowing 1 sign bit, we have 23 places in total and so we fit 22 zeros.

That means the number is

$$1+\frac{1}{2^{23}}=1+\frac{2^7}{2^{30}}=1+\frac{2^7}{(2^{10})^3}\cong 1+\frac{128}{(10^3)^3}=1+\frac{1.3\times 10^2}{10^9}=1+1.3\times 10^{-7}$$

(An accurate calculation gives $1+1.19209290\times 10^{-7}$ as the next number bigger than 1 that computers can store using the commonly used IEEE standard method.)

A consequence of this is that we cannot add a very small number to 1 and get an accurate answer, even though we can keep track of both the 1 and the very small number fine. For example the small number could be 2^{-24} or 2^{-75}), but we would be forced to round 1+ either of those numbers to 1. We can get a similar problem with numbers of different magnitude than 1. If we look at the problem relative to the size of the correct result, we get a concept called *relative error* for a quantity.

Relative Errors

The idea is that an error should not be considered in the abstract.

An error of 1 milimetre may seem small, and it would be quite small if the total magnitude of the quantity concerned was 1 kilometre but if the measurement is for the diameter of a needle, then a 1 milimetre error could be huge.

If we measure (or compute) a quantity where the 'true' or 'correct' answer is x but we get a slightly different answer $\tilde{x}$ (maybe because of inaccuracies in an experiment or because we made some rounding errors in the calculation) then the *error* is the difference

$$\text{error} = x - \tilde{x} = (\text{true value}) - (\text{approximate value})$$

Normally we don't worry about the sign and only concentrate on the magnitude or absolute value of the error. In order to asses the significance of the error, we *have to compare it to the size* of the quantity x.
The *relative error* is a more significant thing:

$$\text{relative error} = \frac{\text{error}}{\text{true value}} = \frac{(\text{true value}) - (\text{approximate value})}{\text{true value}} = \frac{x - \tilde{x}}{x}$$

It expresses the error as a fraction of the size of the thing we are aiming at. 100 times this give the percentage error.
Suppose we use $\frac{22}{7}$ as an approximation to π. Then the relative error is

$$\text{relative error} = \frac{(\text{true value}) - (\text{approximate value})}{\text{true value}} = \frac{\pi - \frac{22}{7}}{\pi} = 0.000402$$

or 0.04%.

Another way to look at the idea of a relative error will be to consider the number of significant figures in a quantity.

What happens with single precision floating point numbers is that we have at most 23 significant binary digits. When translated into decimal, this means 6 or 7 significant digits.

That means that a computer program that prints an answer 6543217.89 should normally not be trusted completely in the units place. (The 7 may or may not be entirely right and the .89 are almost certainly of no consequence.)

That is even in the best possible case. There may also be more significant errors along the way in a calculation that could affect the answer more drastically.

If a computer works in double precision, then there is a chance of more significant digits. In double precision, the next number after 1 is $1 + 2^{-52} \cong 1 + 2.6 \times 10^{-16}$ and we can get about 15 accurate digits (if all goes well).

Linear approximation

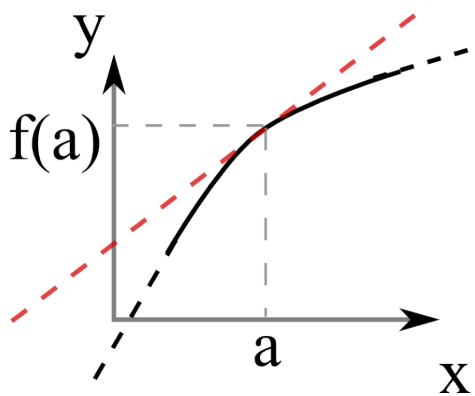
To think more about how approximations are made we recall briefly the linear approximation formula. It applies to functions

$$f = f(x)$$

(of a single variable x) with a derivative

$$f'(a)$$

that is defined at least at one point $x = a$.
The graph $y = f(x)$



of such a function has a tangent line at the point on the graph with $x = a$ (which is the point with $y = f(a)$ and so coordinates $(a, f(a))$). The tangent line is the line with slope $f'(a)$ and going through the point $(a, f(a))$ on the graph.

We can write the equation of the tangent line as

$$y = f(a) + f'(a)(x - a)$$

The linear approximation formula says that the graph $y = f(x)$ will follow the graph of the tangent line as long as we stay close to the point where it is tangent, that is keep x close to a.

It says

$$f(x) \cong f(a) + f'(a)(x - a) \qquad \text{for } x \text{ near } a$$

The advantage is that linear functions are easy to manage, much more easy than general functions. The disadvantage is that it is an approximation.

Condition Numbers

We can use linear approximation to understand the following problem.

Say we measured x but our answer was $\tilde{x}$ and then we compute with that to try to find $f(x)$ (some formula we use on our measurement). If there are no further approximation in the calculation we will end up with $f(\tilde{x})$ instead of $f(x)$. How good an approximation is $f(\tilde{x})$ to the correct value $f(x)$?

We assume that $\tilde{x}$ is close to x and so linear approximation should be valid.

We use the linear approximation formula

$$f(\tilde{x}) \cong f(x) + f'(x)(\tilde{x} - x) \qquad \tilde{x} \text{ near } x$$

So the final error

$$\text{(true value)} - \text{(approximate value)} = f(x) - f(\tilde{x}) \cong f'(x)(x - \tilde{x})$$

Notice that $x - \tilde{x}$ is the error in the initial measurement and so we see that the derivative $f'(x)$ is a magnifying factor for the error.

But we are saying above that relative errors are more significant things than actual errors. So we recast in terms of relative errors. The relative error in the end (for $f(x)$) is

$$\frac{f(x) - f(\tilde{x})}{f(x)} \cong \frac{f'(x)}{f(x)}(x - \tilde{x})$$

To be completely logical, we should work with the relative error at the start $\frac{x-\tilde{x}}{x}$ instead of the actual error $x - \tilde{x}$. We get

$$\frac{f(x) - f(\tilde{x})}{f(x)} \cong \frac{xf'(x)}{f(x)} \frac{x - \tilde{x}}{x}$$

or

$$\text{relative error in value for } f(x) = \frac{xf'(x)}{f(x)}(\text{relative error in value for } x)$$

Thus the relative error will be magnified or multiplied by the factor $\frac{xf'(x)}{f(x)}$ and this factor is called the *condition number*.

In summary

$$\text{condition number} = \frac{x f'(x)}{f(x)}$$

Examples

(i) Find the condition number for $f(x) = 4x^5$ at $x = 7$.

$$\frac{xf'(x)}{f(x)} = \frac{x(20x^4)}{4x^5} = \frac{20x^5}{4x^5} = 5.$$

So in this case it does happens not to depend on x.

(ii) Use the condition number to estimate the error in $1/x$ if we know $x = 4.12 \pm 0.05$.

If we take $f(x) = 1/x$ we can work out its condition number at $x = 4.12$:

$$\frac{xf'(x)}{f(x)} = \frac{x\left(\frac{-1}{x^2}\right)}{\frac{1}{x}} = \frac{\frac{-1}{x}}{\frac{1}{x}} = -1$$

This means it again does not depend on x in fact.

Now our initial value $x = 4.12 \pm 0.05$ means we have a relative error of (at most)

$$\frac{\pm 0.05}{4.12} \cong \pm 0.012$$

The relative error in $f(4.12) = 1/(4.12)$ is then going to be about the same (because the condition number is -1 and this multiples the original relative error). So we have, using $\tilde{x} = 4.12$ as our best approximation to the real x,

$$f(\tilde{x}) = \frac{1}{\tilde{x}} = \frac{1}{4.12} = 0.242718$$

and this should have a relative error of about 0.012. The magnitude of the error is therefore (at worst) about $(0.012)(0.242718) = 0.0029$ or about 0.003. So we have the true value of

$$\frac{1}{x} = 0.242718 \pm 0.003 \text{ or, more appropriately } 0.243 \pm 0.003$$

(no point in giving the 718 as they are not at all significant).

(iii) $f(x) = e^x$.

Condition number at $x = 10/3$ is

$$x\frac{f'(x)}{f(x)} = x\frac{e^x}{e^x} = x = 10/3 = 3.333\dot{3}.$$

So if we use $\tilde{x} = 3.33$ instead of $x = 10/3$ we would have a relative error to begin with

$$\text{relative error in } x = \frac{\frac{10}{3} - 3.33}{\frac{10}{3}} = 0.001$$

(that is an error of 0.1%). If we now compute $e^{3.33}$ while we ought to have computed $e^{10/3}$ we will have a relative error about 10/3 times larger (the condition number is 10/3, or roughly 3). So we will end up with a 0.3% error. In other words, still quite small.

If instead we were working with $x = 100/3$ and we took $\tilde{x} = 33.3$, we would have the same initial relative error

$$\text{relative error in } x = \frac{\frac{100}{3} - 33.3}{\frac{100}{3}} = 0.001$$

but the condition number for e^x is now $x \cong 33$. The error in using $e^{33.3}$ where we should have had $e^{100/3}$ will be a relative error about 33 times bigger than the initial one, or $33 \times 0.001 = 0.033$. This means a 3.3% error (not very large perhaps, but a lot larger than the very tiny 0.1% we began with.
In fact $e^{33.3} = 2.89739 \times 10^{14}$ and 3.3% of that is 9.56137×10^{12}.

There are many other important things we could say about numbers, there representations and their generalisations. One important one, which connects back to the early part of the course, is the notion of complex numbers.

Complex Numbers

We know that solutions of the equation

$$x^2 - x + 1 = 0$$

are

$$x = +\frac{1}{2} \pm \sqrt{-1}\frac{\sqrt{3}}{2} = +\frac{1}{2} \pm i\frac{\sqrt{3}}{2}$$

where i is the imaginary unit and numbers of the form $a + ib$ are called complex numbers.

We might wonder if other equations of the from

$$a_N x^N + a_{N-1} x^{N-1} + \cdots + a_1 x + a_0 = 0$$

require further generalisations however this is not the case. We won't prove it here but the so called Fundamental Theorem of Algebra tells us that complex numbers are sufficient to satisfy any such polynomial.

Complex Numbers

Let us denote individual complex numbers by the letter z and the set of complex numbers by $\mathbb{C}$. That is

$$z = a + ib$$

with a and b real numbers and $i^2 = -1$.

The real part of z is a and is often denoted by $\mathrm{Re}(z) = a$ while b is the imaginary part, $\mathrm{Im}(z) = b$. A complex number of the form $z = a + 0i$ is simply called a real number and so the real numbers can be viewed as a subset of the complex numbers. A complex number of the form $z = 0 + ib$ is called an imaginary number.

Two complex numbers are considered equal only if they have equal real and imaginary parts.

Complex Numbers

Complex numbers can be added, subtracted and multiplied by real numbers in the straightforward fashion

- $(a + bi) + (c + di) = (a + c) + i(b + d)$
- $(a + bi) - (c + di) = (a - c) + i(b - d)$
- $k(a + bi) = ka + kbi$ where $k \in \mathbb{R}$.

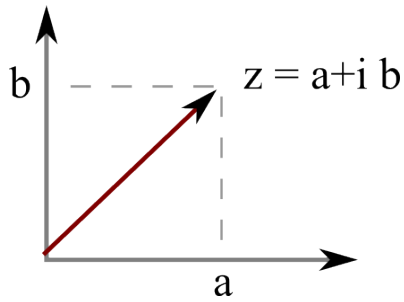
Multiplying numbers is also straightforward, but perhaps looks slightly different,

- $(a + bi)(c + di) = (ac - bd) + i(ad + bc)$

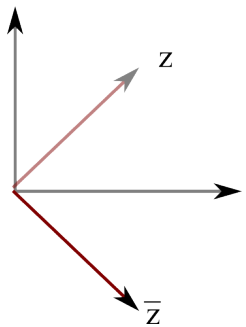
which follows from the the usual multiplication of real numbers and $i^2 = -1$.

Complex Plane

As is probably apparent from the familiarity of the rules we can think of the complex number with addition as a two-dimensional vector space. Moreover we can associate to each complex number, $a + ib$, an ordered pair (a, b) which can be represented graphically as an element of the complex plane.



The sum of two complex numbers can be calculated from the rules for vector addition (triangle law or parallelogram law). The complex conjugate of $z = a + ib$ is $z = a - ib$ and can be thought of as a reflection in the real axis



We can define the modulus of a complex number as

$$|z| = \sqrt{z\bar{z}} = \sqrt{a^2 + b^2} \ .$$

One important feature is the for complex numbers, unlike for vector spaces in general, we can define a reciprocal and hence division. If $z \neq 0$ we have

$$\frac{1}{z} = \frac{\bar{z}}{|z|^2} \ .$$

Quite obviously

$$z\left(\frac{1}{z}\right) = 1$$

as required. The quotient of two complex numbers is

$$\frac{z_1}{z_2} = \frac{z_1 \bar{z}_2}{|z_2|^2} \ .$$

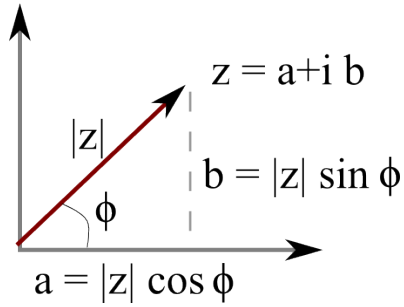
The properties of the conjugate can be summarised as

- $\overline{z_1 + z_2} = \bar{z_1} + \bar{z_2}$
- $\overline{z_1 - z_2} = \bar{z_1} - \bar{z_2}$
- $\overline{z_1 z_2} = \bar{z_1}\bar{z_2}$
- $\overline{z_1/z_2} = \bar{z_1}/\bar{z_2}$
- $\bar{\bar{z}} = z$

While the following results hold for the modulus

- $|\bar{z}| = |z|$
- $|z_1 z_2| = |z_1||z_2|$
- $|z_1/z_2| = |z_1|/|z_2|$
- $|z_1 + z_2| \leq |z_1| + |z_2|$

A useful form for complex numbers is called the polar form. Graphically this follows from



and can be written as

$$z = |z|(\cos\phi + i\sin\phi)$$

Here ϕ is called the argument of z. The argument is not unique as we could add any multiple of 2π and this would give the same z.

If ϕ is a real number then the complex exponential is defined to be

$$e^{i\phi} = \cos\phi + i\sin\phi \ ,$$

and so we can write $z = |z|e^{i\phi}$. This gives a trivial way to remember DeMoivre's formula

$$z^n = |z|^n(\cos\phi + i\sin\phi)^n = |z|^n(e^{i\phi})^n = |z|^n e^{in\phi} = |z|^n(\cos n\phi + i\sin n\phi)$$

Complex vector spaces

Much of this may be familiar but it also allows us to make a non-trivial generalisation of our earlier idea of a vector space.
Recall that we had several vector space axioms including

If $\mathbf{u}$ and $\mathbf{v}$ are vectors and k and l are scalars then

- $k(\mathbf{u}+\mathbf{v}) = k\mathbf{u} + k\mathbf{v}$.
- $(k+l)\mathbf{u} = k\mathbf{u} + l\mathbf{u}$
- $(kl)\mathbf{u} = k(l\mathbf{u})$

Before we had in mind k and l being real numbers, that is elements of $\mathbb{R}$, but they can equally well be complex numbers, that is elements of $\mathbb{C}$. In this case we find a complex vector space.