Canadian Journal of Pure and Applied Sciences
Vol. 15, No. 2, pp. 5201-5219, June 2021
Online ISSN: 1920-3853; Print ISSN: 1715-9997
Available online at www.cjpas.net

REIMAGINING COMPLEX NUMBERS

Christopher C. O'Neill
University College Dublin (UCD), Belfield, Dublin, Ireland

ABSTRACT

This work starts by examining the square root problem, i.e. $(-1)^{1/2}$. By looking for practical solutions to this problem, it arrives at a new mathematical space, where imaginary numbers can be reinterpreted free from their algebraic context and therefore from an entirely different perspective. This new mathematical space is based on XNOR logic gates and deals strictly with operators. Further permutations of this method lead to a total of 16-dimensional gate operator spaces, which may have some application to Quantum Mechanics.

Keywords: Quaternions, octonions, sedenions, complex plane, mathematical operators, quadratic equation, logic Gate algebra.

INTRODUCTION

Imaginary numbers are invaluable in many areas of mathematics, physics, and, in particular, quantum mechanics (Karam, 2020), see also in ("Hermitian Operators and their Applications" by AS. Hicks) and ("Quantum physics needs complex numbers" by Marc-Olivier Renou *et al*., arXiv:2101.10873v1). They allow us to extend our understanding of the real numbers into the abstract realm of the Complex Plane. Complex numbers are in the form of (a + bi), where 'a' and 'b' are real numbers (like 1, 2, 3…) and 'i' is the square root of –1. The Complex Plane can be further extended into the 4-dimensional realm of the Quaternions, Octonions, and Sedenions, etc. In 1843, the famous Irish mathematician and physicist William Rowan Hamilton extended the complex numbers into the 4-dimensional realm of the Quaternions (Hamilton, 2000). Shortly thereafter, Hamilton's friend John T. Graves discovered the Octonions.

Since then, a 16-dimensional hypercomplex, known as the Sedenions has been discovered. But with each successive extension of the Reals into the imaginary realms, some functionality has been lost. The Real numbers are very good at describing and modelling the world. This is because they are ordered, commutative, and associative. When we move to the Complex Plane, the numbers are no longer ordered. This is because the value of $(-1)^{1/2}$ is not a known value. When we reach the Quaternions, we lose the property of commutation. And the situation gets worse as we go to the Octonions where the property of associativity is lost. Now, $a\times(b\times c)$ no longer equals to $(a\times b)\times c$. This makes it difficult to do actual calculations in these mathematical realms. By the time, we get to the Sedenions all hell breaks loose and nontrivial division by zero is allowed. Perhaps that is why John Baez asked; 'If octonions are the crazy uncle that no one lets out of the attic, would sedenions be the serial killer maximum-security prison escapee that no one even lets in the house?'(https://en.wikipedia.org/wiki/Talk%3ASedenion)

Nowadays, scientists, mathematicians, and physicists like to think of complex numbers as variables over finite fields (Campello de Souza *et al*., 2002). This produces another layer of abstraction which can used to model electrical fields and has a great deal of application in String Theory (Moosavian and Pius, 2019). But how does an increase in abstraction help us clear up the confusion surrounding an already abstracted problem? It doesn't. Therefore, the author proposes going the other way, and stripping the imaginary of the abstract to revealing the true form it has been hiding from us, all along.

Real or Imaginary?

When $(-1)^{1/2}$ was first encountered in quadratic equations, its usefulness wasn't considered outside of how it could be used for equations to resolve them (https://en.wikipedia.org/wiki/Cubic_equation). The 16th Century Italian poly-mathematician Gerolamo Cardano was the first to take notice of it. But it wasn't until Rafael Bombelli, 'the father of complex numbers' that the problem was really understood and their potential truly known (https://en.wikipedia.org/wiki/Rafael_Bombelli).

The problem can be stated as $x^2 = -1$. We have to solve for x. If we substitute +1 for x we get +1, so that can't be right. If we use –1, we still get plus one. So that can't be right either. In fact, no matter what number we use, we will never get $x^2 = -1$.

Corresponding author e-mail: chris.ozneill@gmail.com

One way to get around this problem is to rewrite x as i and set it as a 'complex number' ($a + b$i). Then, we multiply the complex number ($a + b$i) by its conjugate ($a - b$i) to obtain the absolute value of any number in the complex plane. This result is possible because squaring of $(-1)^{1/2}$ gives –1.

But is there another solution to this problem? Perhaps a solution that doesn't persist in giving more and more abstract interpretations? In order to find this solution, we need to examine how operators work. What does the author mean by operators in this context?

In the mathematical language, operators are those like {+, –, ×, ÷}. When we multiply a positive number by a positive number, we get a positive real number. When we multiply a negative by a positive, we get a negative and when we multiply a negative by a negative, we get a positive number. It is the first and last rules that prevent us from solving the problem of $(-1)^{1/2}$. So, what do we do? We change the order of the operators and how they work.

There are many ways to do this. The current arrangement of our operators is isomorphic with the XOR logic rule set, also known as 'exclusive or' shown in Figure 1.

0	0	0
1	0	1
0	1	1
1	1	0

+	+	+
–	+	–
+	–	–
–	–	+

Fig. 1. The XOR logic rule set.

As the reader can see in Figure 1, the 0s and 1s of the XOR exactly match the '+' and '–' of our operators. And this works equally well for the divisors. Why is our mathematics based on the XOR system, the reader might ask? Why not on some other more familiar logic system, like OR, AND, NOR, or NAND? It seems sort of arbitrary, doesn't it?

While there are good logical reasons for why a minus multiplied by a minus equals to a plus, it would appear that in the case of $(-1)^{1/2}$ and the Complex Plane, this property no longer holds true (Marks-Tarlow *et al.*, 2015). A very similar idea was put forward by Martin A. Hays, who has conceived of six different number systems using +0, –0 and the shape of the Necker Cube to generate a new mathematical system. ('Real and Imaginary Numbers', available online: http://chiralkine.com/real-and-imaginary-numbers/).

If we examine the logic chain for XOR, which is 0110, we see that this is the binary number, which is itself equal to 6 (or 7 depending on how one counts them.) Since there are 16 variations on a four-bit binary number, it is conceivable that we have 16 variations of our number bit operators and it is certain that our new set of operators are to be found there. In fact, N. J. Wildberger posits the same idea in his video the "Implication and 16 logical operations" (https://www.youtube.com/watch?v=XkqmuUg_yFs). For an alternative perspective on this, see also the work by the author (C. O'Neill, "Logic Gate Arithmetic and Quaternions" DOI: 10.13140/RG.2.2.10320.12809).

Notice how the 16 logical operations correspond to the 16-dimensions of the Sedenion Numbers. But unlike, the n-dimensional division algebras of the imaginary numbers, these systems will be both well-behaved (ordered), commutative, and associative. Furthermore, they will permit extensions into odd and singly-even dimensions. In this sense, 5, 6, and even seven-dimensional algebra will be possible in a manner that will be relatively simple and easy for anyone to understand. But before we do that, we must choose our operator set. So, what is the best choice? Obviously one where positive square numbers and negative square numbers equal to negative numbers. XNOR will easily achieve that is shown in Figure 2.

0	0	1
1	0	0
0	1	0
1	1	1

+	+	–
–	+	+
+	–	+
–	–	–

Fig. 2. The XNOR logic rule set.

With this set of operators, we can now easily prove $(-1)^{1/2}$ to be equal either to +1 or –1. Our next step is to privilege this result; meaning that from now on all of our mathematics will be done in the XNOR (!Δ) operator set. From here on out, all of our calculations will take place using the XNOR ruleset. We have already determined what the rules for multiplication and division, but in order to carry out even the most basic of calculations, we need to know the rules of addition and subtraction as well.

Basic Arithmetic

How does our new XNOR rule set deal with something simple like (a + b = c)? The answer is more difficult than we would at first imagine. Let's take a look at the rules for addition in XOR:

$$A + (+B) = A + B$$
$$A + (-B) = A - B$$
$$A - (+B) = A - B$$

A – (–B) = A + B

As we can see, these are identical to the multiplication rules of XOR. Now let's look the same rules in XNOR:

A + (+B) = A – B
A + (–B) = A + B
A – (+B) = A + B
A – (–B) = A – B

They are just the opposite. In light of this, we can do arithmetic at all stages across the divide of the XNOR and XOR dimensional axes or gateways. But how does this work with some simple real-world examples? If we have the sum 4 + 6 = 10 (in XOR) and we exchange the operators, we get:

4 + (+6) = +10
4 + (–6) = –2
–4 – (+6) = –10
4 – (–6) = +10
–4 – (–6) = +2

In XNOR, we get the complete inverse:

4 + (+6) = –2
4 + (–6) = +10
–4 – (+6) = +2
4 – (–6) = –2
–4 – (–6) = –10

Now that we have our new rule set of arithmetic we can apply it to more complex examples. When we do this, we inevitably find the somewhat perplexing value of $(+1)^{1/2}$, which can't be solved with our current operators. Our new XNOR equation for the quadratic formula looks like this:

$$y = \frac{b \pm \sqrt{-b^2 - 4(ac)}}{2a}$$

Using this to solve the equation $2x^2 + 4x + 20 = 0$ will require us to use the ruleset for basic arithmetic seen in the last table. This results in part of the expression being $(-16-160)^{1/2}$, which based on above ruleset is $(144)^{1/2}$. However, this can be confusing. Therefore, we can multiply by –1, which is the method employed in the computer programs in the **Appendices**

$$y = \frac{4 \pm \sqrt{-16 - (160 \times (-1))}}{-4}$$

$$y = \frac{4 \pm \sqrt{144}}{-4}$$

$$y = \frac{4}{-4} \pm \sqrt{\frac{144}{-4}}$$

which is equal to 1 – 3i or 1 + 3i. In XOR, we know that i = ±1 and so our answer becomes –2 or 4. Now, we have a complete and ordered set of operators for the complex plane and we can do algebra with it. We will call this algebra, Logic Gate Algebra.

Imaginary Numbers are Real

In a very popular video by Welch Labs, entitled 'Imaginary Numbers Are Real [Part 1: Introduction]' (https://www.youtube.com/watch?v=T647CGsuOVU), there is a demonstration of the concept of imaginary numbers and how they can be used to find the roots of quadratic equations, which otherwise don't appear to have any roots.

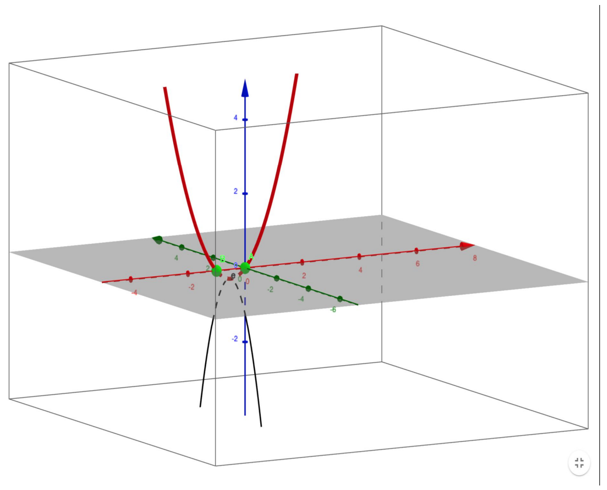


Fig. 3. This graph showing the polynomial roots of $x^2 + 1$ generated on the online graphing application Geogebra (https://www.geogebra.org/m/U2HRUfDr).

An example of what is being described in the video can also be seen in Figure 3. Here we have a parabola $x^2 + 1$ and we are interested in the roots of this function. According to the video, there are no roots to this function, as the parabola does not cross the x-axis. However, according to Gauss's Fundamental Theorem of Algebra, any polynomial of degree n must have n roots. When we look we see that our equation has degree 2, therefore it has 2 polynomial roots.

The solution can be once again found using the Quadratic Formula. The answer can be partially seen in Figure 3. There is a grey line (representing $x^2 + 1$ and a red line, which is the imaginary curve and can be see crossing the x-axis at two points.

In the video, we are shown an interesting animation, where the function is stretched into a 2nd dimension and pulled down below the x-axis (See Fig. 4). This shows a kind of analytic continuation between the Real and imaginary part of the equation, like a map between the two different functions. In some sense, then this new surface is the true form of the equation $x^2 + 1$, as it contains the most information about the surface and where it connects, too.

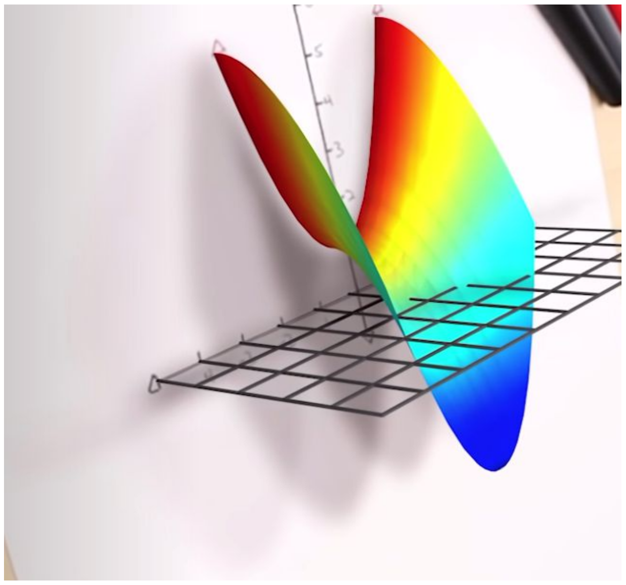


Fig. 4. The animation showing the complex roots of $x^2 + 1$. Image credit: Welch Labs (https://www.youtube.com/watch?v=T647CGsuOVU).

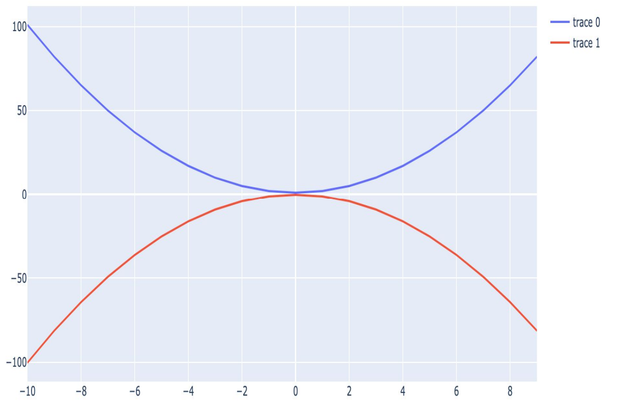


Fig. 5. The blue trace for $x^2 + 1$ and the red trace for $– x^2 – 0.25$.

The question is: Is it possible to create a function or a program which actually generates this surface and can we use the new XNOR ruleset to help us in doing so? If we attempt to graph our original function $x^2 + 1$ and the imaginary roots of function $(– x^2 – 0.25)$, we get something not unlike Figure 5.

We need some way to extend these two surfaces so that they interact with one another. As we known, $x^2 + 1$ creates a parabola, which is itself the result of multiplying two lines together. Therefore, we may be able to extend these two parabolas by multiplying them together, in a special way. The result of this simple process can be seen in Figure 6 and the code which generated this image can be found in **Appendix G**. The reader will note that it looks nothing like what is presented in the animation in Figure 4.

Instead, what we have is a kind of singularity, not unlike the kind of 2-dimensional singularities used to model black holes in a popular sense. Similar structures also presented in (Teh, 2010). This would appear to reinforce the connection between Riemannian Geometry and the curvature of spacetime, since we have quickly arrived at a model of a 2-dimensional blackhole, simply by muddling around with functions and their imaginary counterparts.

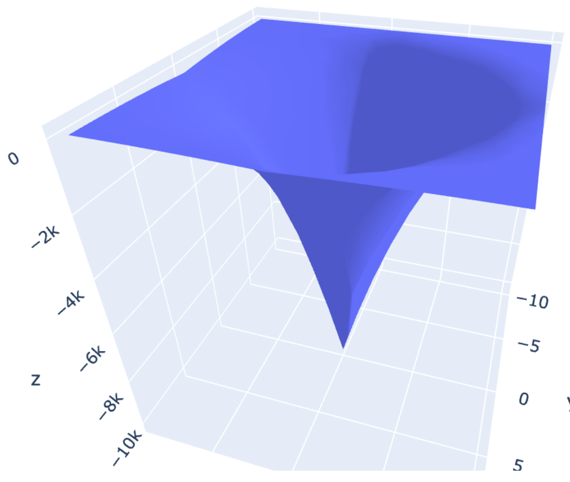


Fig. 6. The result of the parabolas $x^2 + 1$ and $– x^2 – 0.25$ being multiplied together.

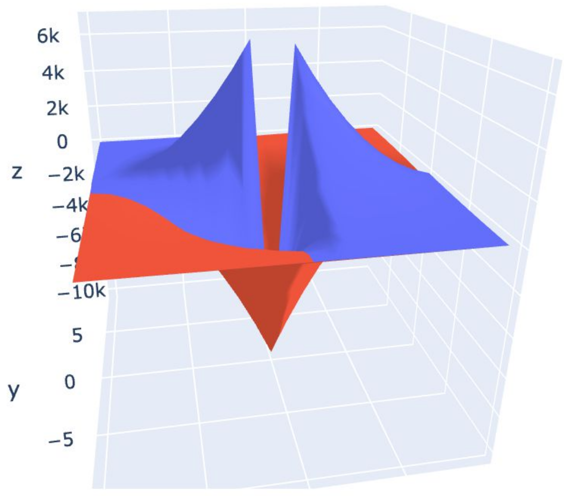


Fig. 7. An attempt to incorporate the XNOR ruleset into the program (**Appendix G**).

The reader will notice that there is no mention of any of the Logic Gate Algebra in this method. We can attempt to implement a kind of Logic Gate Algebra (also demonstrated in the code in Appendix G) but the methods and the results are dubious, at best, and require refinement.

So how do we go about creating maps between the Real numbers and the imaginary numbers, like the ones see in the animation in Figure 4? The simplest and most obvious route would be to simply transform one parabolic curve by the other. But that is also not very interesting.

Undiscovered things

Quadratic equations are equations in the form $ax^2 + bx + c = 0$. This form holds true for both the Real numbered part and the imaginary part, but we can't always expect it to hold true for all parts of our mapping and in fact, it will likely only be the case at very specific places where the like terms are being multiplied by one another. But before we can do that, we need to look at some of the basics of linear algebra in the Logic Gate Algebra system.

Suppose we want to plot the expression: 2(3 + i)(4 + i). Under the original Complex Plane method this would lead to:

$$2(3 + i)(4 + i) = 22 + 14i$$

Under the Logic Gate Algebra, our original expression would be rendered in the form 2(!Δ + 3)(!Δ + 4); the terms are reversed, because we have privileged the use of !Δ(XNOR) over Δ(XOR). The result of this is:

$$2(!\Delta + 3)(!\Delta + 4) = 22$$

This is much simpler and well ordered. But what is going on here? When multiplying through the two different systems XNOR and XOR, we have to multiply the terms twice, once in each system. This generates two equal results of opposite signs, which then cancel each other out.

$$2(!\Delta+3)(!\Delta+4)$$
$$2(-1 + (4-4) + (3-3) + 12)$$
$$2(-1+12) = 22$$

This is equivalent to multiplying FL instead of the full mnemonic: FOIL, which stands for (First, Outer, Inner, Last). But we can't expect this to work with larger expressions like $(!\Delta + 1)^3$ or $(!\Delta + 1)^4$ and so on. A more general approach can be seen in **Appendix C**. Alternatively, they can be placed into matrices where the values cancel out to zero and the remainders are summed to give the answer. If we were to simply graph 2(x + 3)(x + 4) on a normal Cartesian graph, we would get the familiar quadratic curve that is shown in Figure 8.

Plotting the same function in the three-dimensional form (3D) with a plotting program gives similar but not so amazing result shown in Figure 9. Graphing the same function using XNOR and XOR on a 3-dimensional plane however produces interesting and beautiful results shown in Figures 10a and 10b.

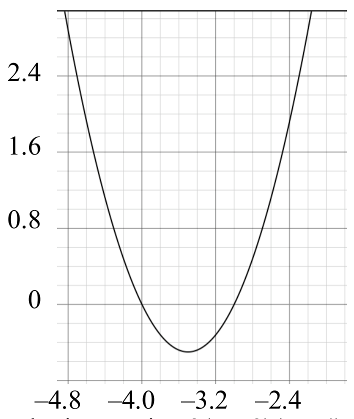


Fig. 8. The quadratic equation 2(x + 3)(x + 4).

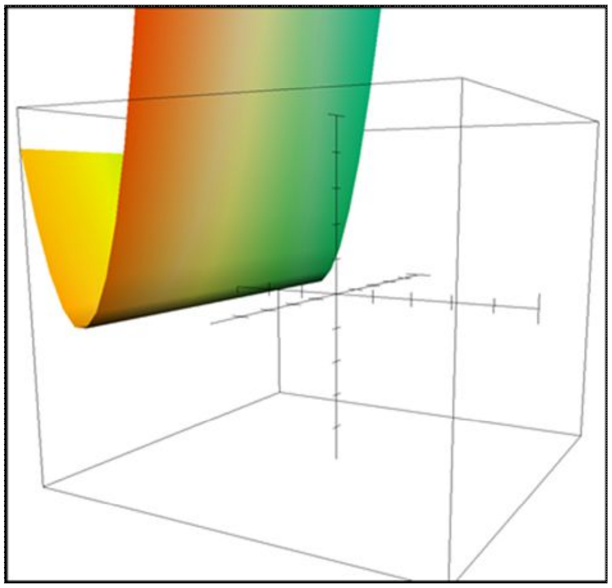
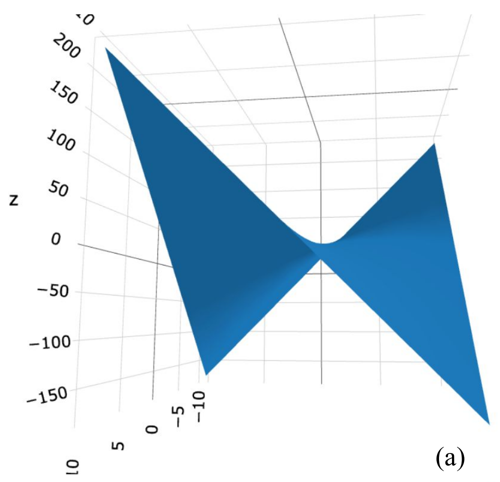


Fig. 9. The three dimensional form of 2(x + 3)(x + 4).

This result is much closer to the kind of map that we see in Figure 4. But it does not appear to be exact and may be skewed. If so, then this is most likely the result of the double for loop needed to generate it. While the resulting surface succeeds in plotting the map between the Real and Imaginary functions, whatever this function is, it is definitely not the original function 2(x + 3)(x + 4). The author has yet to devise a method of discovering what the new function is [in the conventional sense] and how to translate between the two in all cases. This will have to be left until subsequent research, as the research in this paper continues on.


(a)

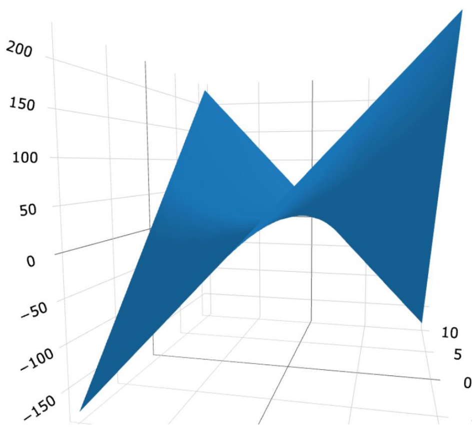

(b)

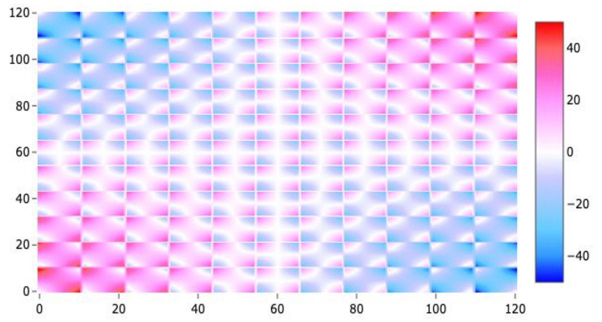
Fig. 10. The more accurate 3-dimensional depiction (a) and (b) of $2(x + 3)(x + 4)$.

Plotting domains

Now that we have our new rules for Logic Gate Algebra or Order 2, so called because it deals with two coordinates, we can begin plotting some functions. To start with, the author plotted results of all functions for (x!Δ + yΔ)(x1Δ + yΔ) over a finite field to produce the polynomial distribution on the complex plane shown in Figure 11.

We can increase or decrease the range and increase the step value to produce many such graphs. Interestingly, they all exhibit the same properties at every scale, much like how magnetic fields exhibit the same properties no matter if they are produced by a single atom or a whole storm of atoms nested in a magnetized block of metal.

Graphing a single function is more revealing. In this case it is the function from earlier: 2(!Δ+3)(!Δ+4) that is shown in Figure 12. With this much simpler plot it is easy to see that what we are graphing here is simply the parabolic curve from earlier (see in Figures 13, 10a, and 10b). As we know, this is the general shape of polynomial multiplication in the Logical Gate Space (LGS). But what about polynomial division algebras?


(a)

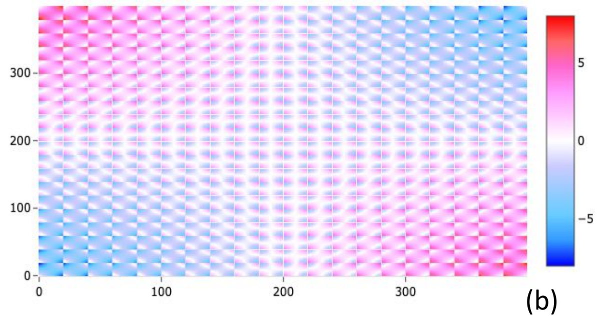

(b)

Fig. 11. The complex plane: Polynomial distribution (a) for the range {–5, 5} and (b) {–2, 2} step 20.

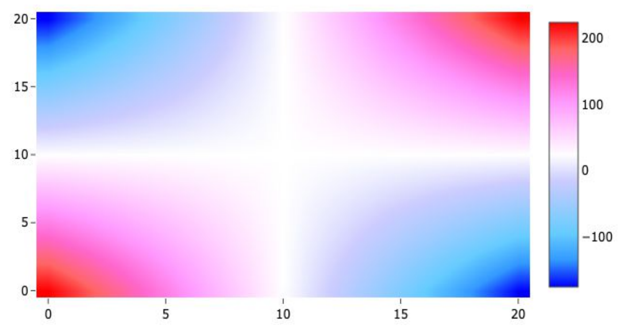


Fig. 12. The complex plane: Polynomial distribution, 2(!Δ + 3)(!Δ + 4).

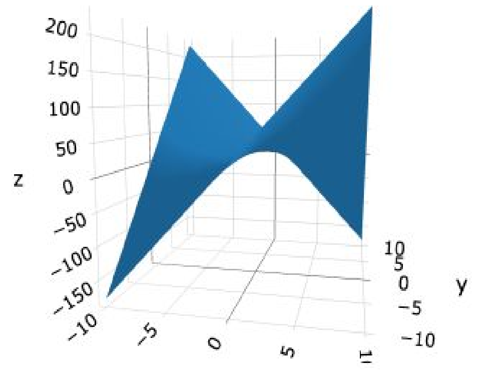
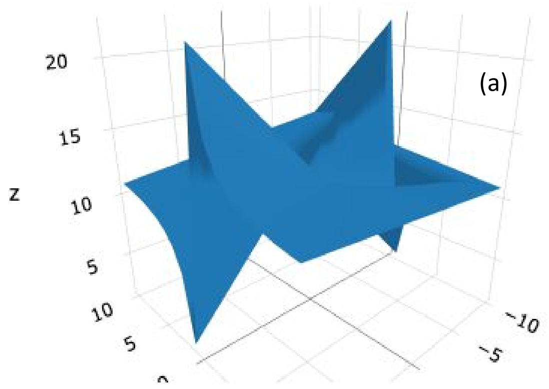


Fig. 13. The hyperbolic quadratic, 2(!Δ + 3)(!Δ + 4).



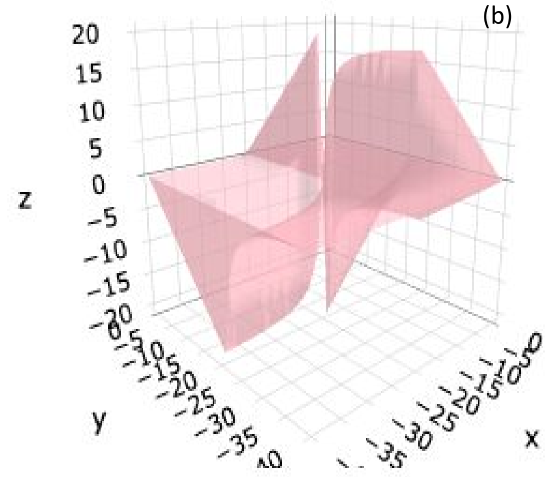
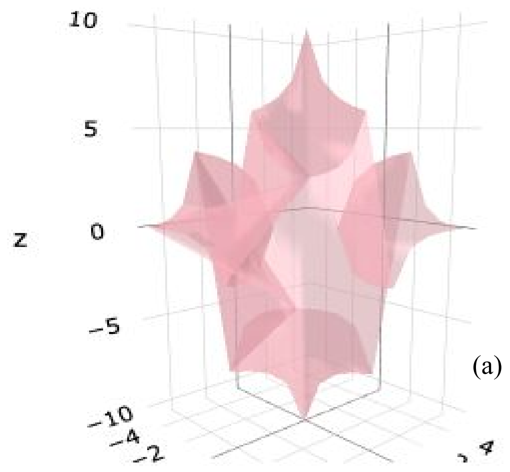


Fig. 14. The hyperbolic Division Quadratic (a), 2(!Δ + 3)/(!Δ + 4), and the polynomial division (b).

Here we have (x!Δ + 3Δ)/(y!Δ + 4Δ), which produces this unusual 'saw-toothed' graph shown in Figure 14. The results for all functions of this kind, over a particular finite range can be plotted in 3D and reveal the same structure (see **Appendix A** for the code). This is interesting and shows that these functions are somehow embedded in themselves. The author has used a less opaque surface here (Fig. 14b) to make the resulting structure more apparent.

The major difference between more general division functions like these and the previous multiplication functions is revealed when the range of the field is increased. These functions appear to show repeat patterns extending out across the plane, whereas the other one just stays in place and increases to infinity. Below, we have one section of the above function (a) and then an extended version (b) shown in Figure 15. Figure 15a is embedded in the graph of Figure 15b, although it helps to be able to rotate the two graphs to compare the shapes to see exactly where it is embedded. The other peaks and troughs hint that this structure repeats outside the limits of the field, but exactly how and in what way is not yet known, as it requires a large amount of processing power to generate such graphs.



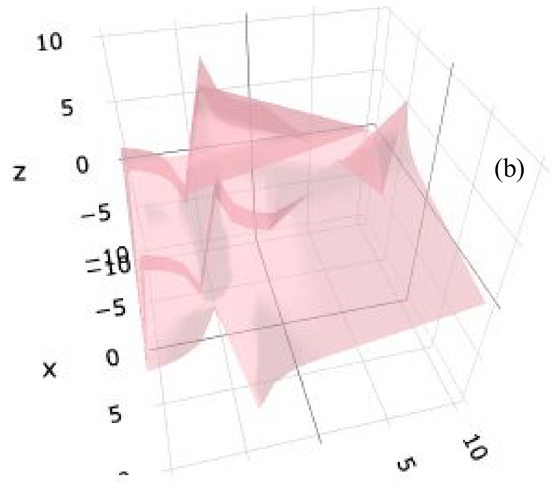


Fig. 15. The embedded (a) and extended (b) graphs.

3-Dimensional cubic functions are enabled by $(x + y)^2$, which was unexpected because the function is squared rather than cubed. The Plotly graphing library used to generate this has a bug in it that glues some of the faces together with unwanted polygons. The author has had to angle this graph so they remain hidden. Unfortunately, this may not be the best angle to view the cubic properties from but it should give the reader some idea. Fortunately, the code for this example is available in **Appendix B**. On the plus side, there is a way of altering the parameter alphahull (this is equivalent to convex hull for coordinate points) which when set to zero joins up all the faces of this cubic function to reveal a cuboctahedron; an object with 6 square and eight triangular faces.

The complex plane cubic equation is shown in Figure 16. Graphing $(x!\Delta + 3)^3$ results in the hyperbolic cubic function shown in Figure 17. The alphahull = 0 of this graph equals a nicely skewed octahedron. The author doesn't know if this means that there is a relationship between 3D cubic functions and platonic solids, but it is interesting to speculate about. Plotting (Δ, !Δ, Δ)^2 gives the familiar saddle graph shown in Figure 18.

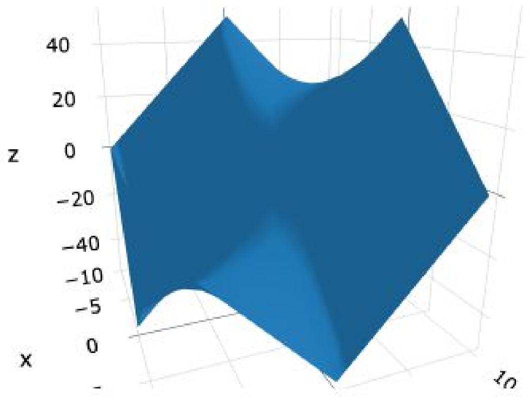


Fig. 16. The complex plane cubic equation.

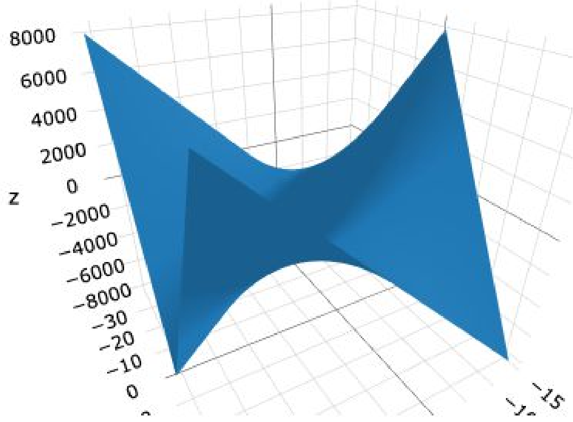


Fig. 17. The hyperbolic cubic function, (x!Δ + 3)^3.

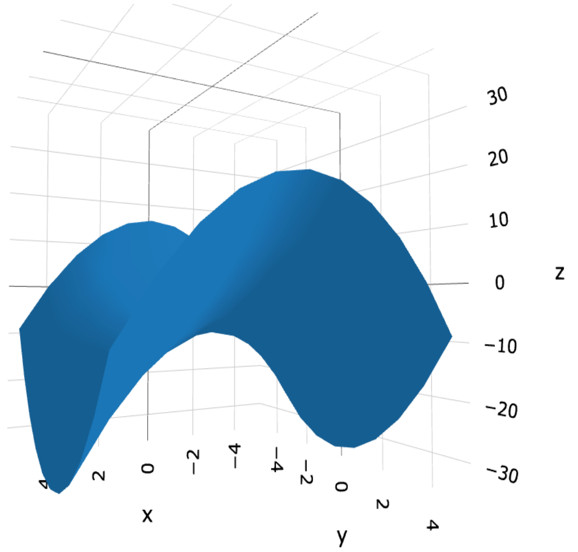


Fig. 18. The saddle graph.

For the most part, representations of the complex plane are done in two-dimensional heat maps. But two extra real space dimensions can always be added to these spaces making them 4-dimensional in total.

It is impossible for the average human to visualize or imagine the fourth dimension but there are ways and means around this. For instance, we can plot three of our values as real space coordinate values and allow the fourth to be some kind of vector coordinate, denoting direction or flow. This works quite well. Another method is simply move each of the four-dimensions in and out of the three real space dimensions at a time, giving one a 'slice' or a window into what the entire function might actually look like. This is not very satisfactory but for us mere 3-dimensional mortals, it is often the best that we can do. Graphing the saddle plot from Figure 16 in the vector coordinates gives the results shown in Figure 19.

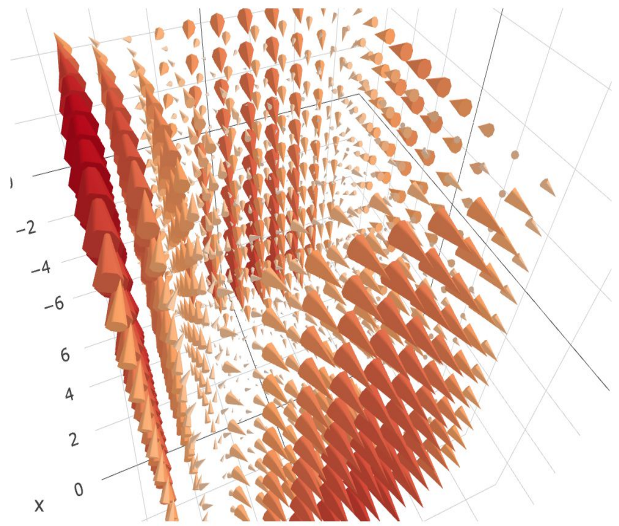


Fig. 19. The saddle graph using XNOR vectors.

Complex Plane

How does the XNOR ruleset work on the Complex Plane? For example, how does it deal with something like the Riemann Zeta Function? The Riemann Zeta function contains an infinite series of the following kind:

$$\mathrm{Re}(z) = 1/1^{(a+bi)} + 1/2^{(a+bi)} + 1/3^{(a+bi)} + 1/4^{(a+bi)} + 1/5^{(a+bi)} \ldots$$

where *a* and *b* are some Real numbered values and 'i' is of course the imaginary number. Working out the first three terms of the Riemann Zeta function results in

$$1/1^{(2+3i)} + 1/2^{(2+3i)} + 1/3^{(2+3i)}$$
$$(\Delta 1 + (-\,!\Delta 1)) + 1/(4 + (-6)) + 1/(6 + (-9)$$
$$\Delta: 0 + 1/(-2) + 1/(-3)$$
$$!\Delta: 2 + (+1/10) + (+1/15)$$
$$\Delta: -0.5 + (-0.333333\ldots)$$

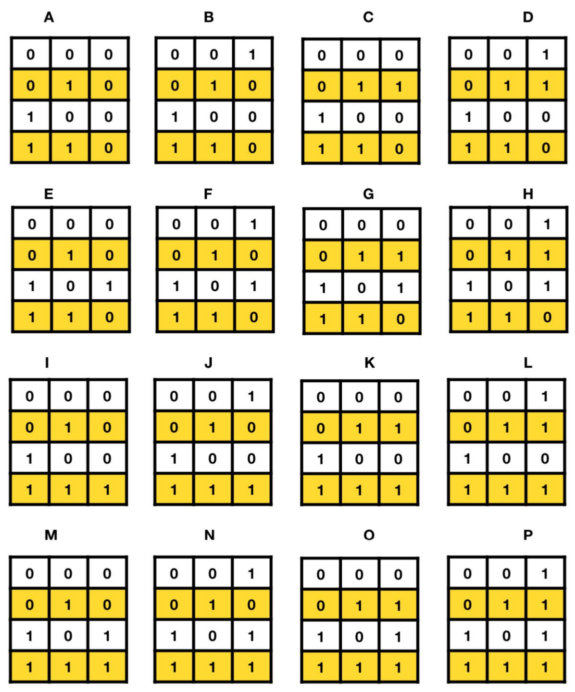
$$! \Delta: 2 - 0.1 - 0.0666666\ldots$$
$$\Delta: -0.833333\ldots$$
$$!\Delta: 1.833333\ldots$$
$$\Delta: -0.833333\ldots + 1.833333\ldots = 1$$
$$!\Delta: -0.833333\ldots + (+1.833333\ldots) = 2.666666\ldots$$

Therefore, the result in coordinates is (1, 2.666666). This is another level of LGA and is fully stocked for all arithmetic, not just multiplication and division. As such this will be labelled LGA-1, whereas our previous system will be labelled as LGA-2. Arithmetic in LGA-1 is admittedly confusing. More work needs to be done on which operators pertain to XOR and which to XNOR and how they interact, otherwise the calculations will result in inconsistent results.

Other series that are useful to do real mathematics include the Taylor Expansion for trigonometric functions for the sine, cosine, and exponential functions. By applying the same rules of arithmetic to these functions, as above, we see that they equate to $\sin(x) = -y$, $\cos(x) = y$ and the exponential function $\exp(x) = y - x$.

16-Dimensional spaces

The LGA are arranged from '0000' to '1111' and ascribed letters 'A' through 'P' as shown in Figure 20. Here 0 = '+' and 1 = '–'. Notice that 'G' is equivalent to XOR, 'J' with XNOR. There are other logic gates here, for example, including; 'I', 'B', and 'O', which correspond to AND, NOR, and NAND, respectively.


A

0	0	0
0	1	0
1	0	0
1	1	0

B

0	0	1
0	1	0
1	0	0
1	1	0

C

0	0	0
0	1	1
1	0	0
1	1	0

D

0	0	1
0	1	1
1	0	0
1	1	0

E

0	0	0
0	1	0
1	0	1
1	1	0

F

0	0	1
0	1	0
1	0	1
1	1	0

G

0	0	0
0	1	1
1	0	1
1	1	0

H

0	0	1
0	1	1
1	0	1
1	1	0

I

0	0	0
0	1	0
1	0	0
1	1	1

J

0	0	1
0	1	0
1	0	0
1	1	1

K

0	0	0
0	1	1
1	0	0
1	1	1

L

0	0	1
0	1	1
1	0	0
1	1	1

M

0	0	0
0	1	0
1	0	1
1	1	1

N

0	0	1
0	1	0
1	0	1
1	1	1

O

0	0	0
0	1	1
1	0	1
1	1	1

P

0	0	1
0	1	1
1	0	1
1	1	1

Fig. 20. The LGA arranged from '0000' to '1111' and ascribed letters 'A' through 'P'.

If we can think of these 16 systems as distinct rules for addition in multi-dimensional space, then we notice some interesting properties. For instance, no matter the operator in 'A' the answer is always a positive. The opposite is true for 'P'. In many of these spaces, like 'C', 'D', 'K' and 'L', a positive number by a negative number yields different results depending on how they are multiplied together. For instance:

$$3C \times (-3C) = -9$$
$$-3C \times 3C = 9$$

This means that they are non-commutative in this respect. But in other respects, when the operators are the same, 'C' is commutative. Graphs similar to those seen in Figures 16, 17, and 18 can be generated using these 16 gates. The code to do this can be found in **Appendix C**.

Hyperbolic curves

One of the most famous demonstrations of the imaginary roots of a quadratic equation comes in the form of $x^2 + 1 = 0$. Ordinarily, we think of this function as having no factors, i.e. it cannot be written as a polynomial. However, if we rewrite this equation in XNOR and XOR, we see that we can generate a polynomial as follows:
$(\Delta!x + 1\Delta)(-\Delta!x + 1\Delta)$
Plotting this gives us the usual hyperbolic polynomial curve that we have seen before. But what about other equations? Are there other equations that are considered to be completely factorized, which can be factorized further with this method? The author doesn't know but we can look at other equations that are close to being factorized, like:

$$2x^2 + 2x + 10 = 0 \qquad (1)$$

If we pull the common factor out, we get:

$$2(x^2 + x + 5) = 0 \qquad (2)$$

If we make use of the 16-dimensional spaces (shown above) we can rewrite equation (1) as:

$$(1\mathrm{A} - x\mathrm{A})(-5\mathrm{I} + x\mathrm{I}) \qquad (3)$$

where 'A' and 'I' are LGA spaces: '0000' and '1000' or AND. But how do we determine, which spaces we are to use?

The method the author employed makes use of matrixes, cartesian products and set theory to narrow down the likely candidates. Once we have the candidates, we can start applying them to the quadratic matrixes to see if they satisfy the result. But there is a much simpler method, which we will also look at in this section, using tables and Numpy matrices.

The result (1A – xA)(–5I + xI) can then be plotted. In Figure 21, we see what is usually meant by a hyperbolic curve in algebra using the same coefficients as in the earlier examples. As we can see this is just a saddle graph. Whereas graphing (1A – xA)(–5I + xI) gives us what we have been calling a hyperbolic graph shown in Figure 22.

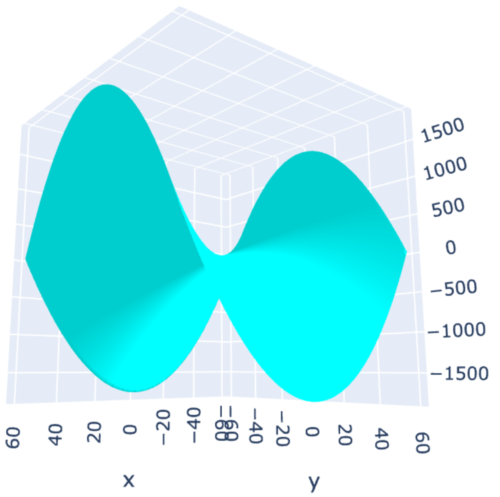


Fig. 21. What is ordinarily thought of as a Hyperbolic Quadratic, $z = (x^2 + y^2)/2 + 5$.

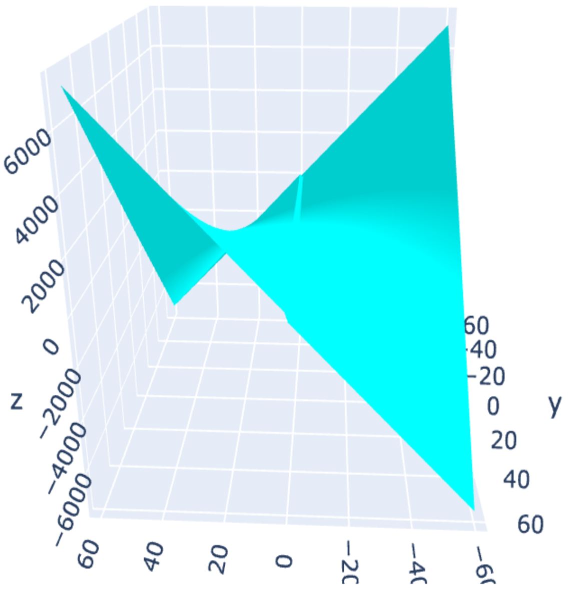


Fig. 22. The actual Hyperbolic Quadratic made from Logic Gates.

We can plot them all together, to show that (1A – xA)(–5I + xI) produces a much better match for the curve than the saddle graph does (**Appendix D**). This is shown in Figure 23. But notice that the aqua-colored graph is skewed in relation to our original surface from equation (1). This result suggests that we have not merely graphed connection between two orthogonal parabolic curves previously shown in Figure 3, but that we have transformed the original curve, before doing so.

The roots of equation (1) are at (0.5, 2.1794) and (–0.5, –2.1794). These coordinates are located in the center of the green patch shown in Figure 24.

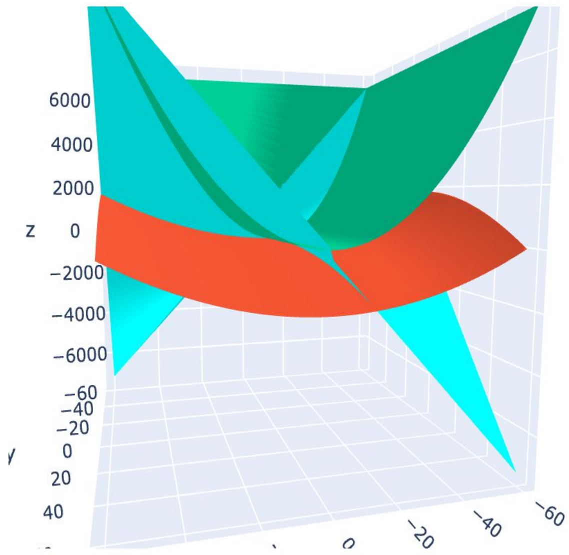


Fig. 23. The comparison of various related quadratic shapes.

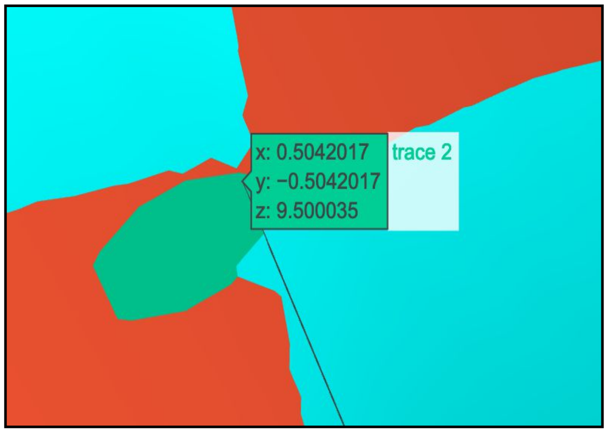


Fig. 24. The roots of quadratic functions.

What can we say about expressions, which are lacking imaginary roots that are close to being factorized or completely factorized? Can these be graphed in higher dimensional space? The answer is 'yes'. Let's take $2x^2 + 4x + 2$. Its roots are $x = –1$. Taking out the common factor, we have:

$$x^2 + 2x + 1$$

If we were to take the factors of this quadratic, we get $(x + 1)(x + 1)$. We can make a table out of this, to produce:

	x	**1**
x	x^2	x
1	x	1

When we add all of the like terms in this table together we return to $x^2 + 2x + 1$. In order to make it fit the function $2x^2 + 4x + 2$, we need to apply logic gates 'A' and 'P' to the matrix. This will ensure that the elements in the table are routinely positive or negative or cancel each other out. We will call these tables AA and AP:

	A	**A**
A	+	+
A	+	+

	P	**A**
A	0	+
P	–	0

We multiply the results of both tables AA and AP by original matrix. This is a direct multiplication of like numbered index values, not the more complex version of matrix multiplication most people are accustomed to. A.A = +2 (i.e. double the original value) whereas P.P = –2 and A.P = 0. Therefore:

$$AA \times (x+1)^2 = \begin{bmatrix} 2x^2 & 2x \\ 2x & 2 \end{bmatrix} \tag{4}$$

$$AP \times (x+1)^2 = \begin{bmatrix} 0 & 2x \\ -2x & 0 \end{bmatrix} \tag{5}$$

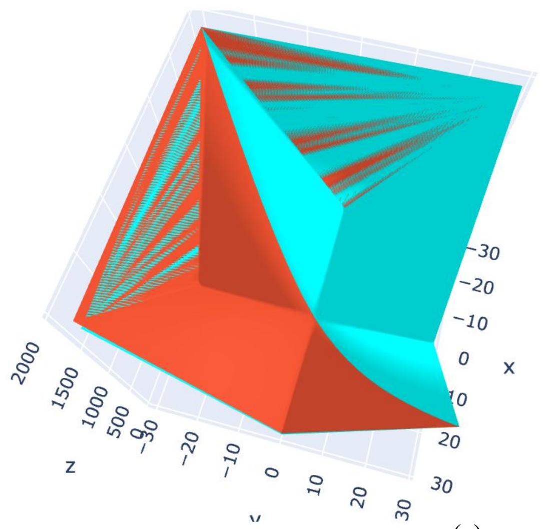
Then we simply sum the products of equation (4) and (5):

$$\begin{bmatrix} 2x^2 & 2x \\ 2x & 2 \end{bmatrix} + \begin{bmatrix} 0 & 2x \\ -2x & 0 \end{bmatrix} = \begin{bmatrix} 2x^2 & 4x \\ 0 & 2 \end{bmatrix}$$

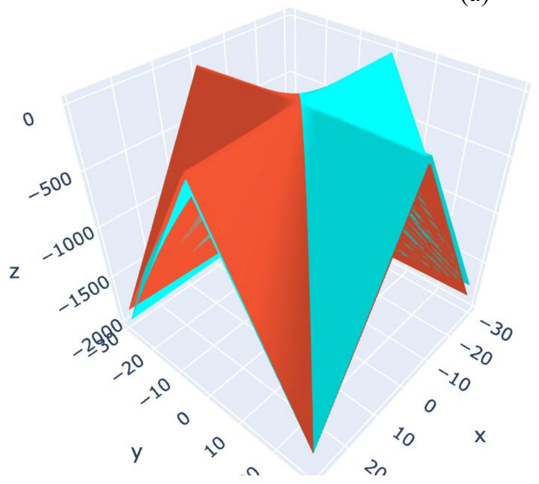
This gives us our original expression: $2x^2 + 4x + 2$. Now, all that is required is to divide by 2 and we have $x^2 + 2x + 1$, or:

(xA + A1)(–xA – 1A) + (xA + P1)(–xP – 1A)

Conceivably, a similar process could be applied to create any quadratic without imaginary roots. Once we have our expression, we can plot it over a range to produce a graph. Previously, these graphs included imaginary numbers and were, in some sense, considered 4-dimensional. Our newest expression was created using the 3 real-space dimensions and two other imaginary dimensions 'A' and 'P'. As a result, we may consider the totality of the output to be 5-dimensional. Although, this is debatable as 'A' and 'P' are constructed entirely out of pluses and minuses respectively, which when summed produces 'G' or XOR. If this is the case, then 'i' doesn't represent a 4-dimensional axis, merely another way of looking at our 3-dimensional axes.



(a)



(b)

Fig. 25. (xA + A1)(–xA – 1A) + (xA + P1)(–xP – 1A) (a) and its inverse (meaning multiplied by –1) (b).

When several of these dimensions are graphed simultaneously, we see the shapes above. Figure 25b is the inverted version of Figure 25a. The line bisecting the graph (a) is our quadratic curve. It is also possible to invert the two graphs and place them one on top of the other. For the code for this see **Appendix E**.

This is further demonstrated in Figure 26, where the red trace equals to ($x^2 + 2x + 1$) and the aqua-colored trace equals to (xA + A1)(–xA – 1A) + (xA + P1)(–xP + 1A). We can plot our simple 4D parabolic equation in a vector space to get a better understanding of what it might look like in higher-dimensions. The author decided to play around with this and generated a 5- or 6-dimensional graph illustrating a field based on our quadratic formula.

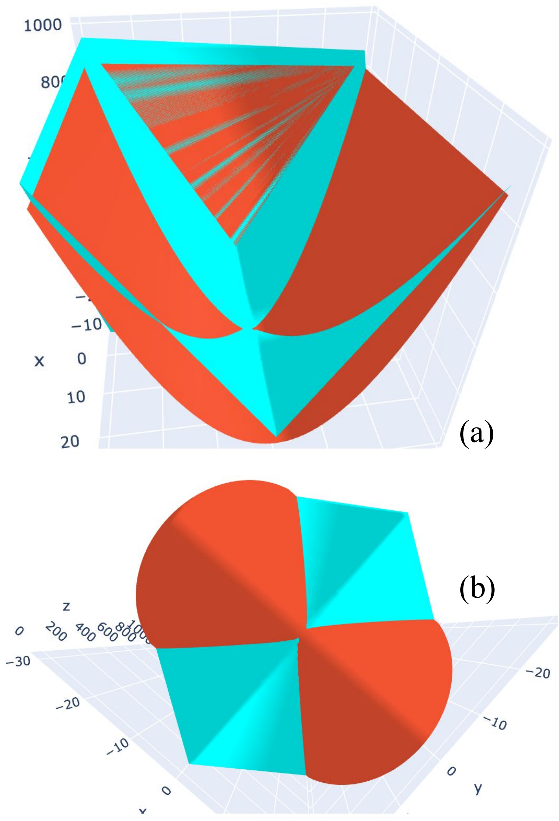


Fig. 26. (xA + A1)(–xA – 1A) + (xA + P1)(–xP – 1A) is the aqua-colored graph. It is plotted alongside the garden variety (x2 + 2x + 1) in red. Both graphs (a) and (b) are the same, merely viewed from different angles.

The graph (Fig. 27) is simply four spatial coordinates, with the '*y*' coordinate repeated in the '*v*' coordinate and the quadratic; (xA + A1)(–xA – 1A) + (xA + P1)(–xP + 1A). The result is interesting, because it shows that each point in the vector space has between 2 and 7 vector cones. All electro-magnetic fields fluctuate slightly (by about the mass of a photon). This method appears to produce such a fluctuation as a matter of course, without the need for any extra parameters. The code that produced this graph can be found in **Appendix F**.

Rotation groups

One of the most important features of complex numbers is there use in rotation groups. The Complex Plane represents a field of numbers that can turn and overwrite itself. These groups are closed under addition and multiplication and form a subring of C_2.

$$i^n = 1, i, -1, -i, 1, i, -1, -i\ldots$$

Can Dimensional Gate Operators do the same thing? It turns out; Yes, they can. For Z^2 rotation group, we can rewrite the above in the following way (Fig. 28):

$$\mathbf{\Delta(1)^2 = !\Delta(1)}$$
$$\mathbf{!\Delta(1)^2 = \Delta(-1)}$$
$$\mathbf{!\Delta(-1)^2 = \Delta(-1)}$$
$$\mathbf{\Delta(-1)^2 = \Delta(1)}$$

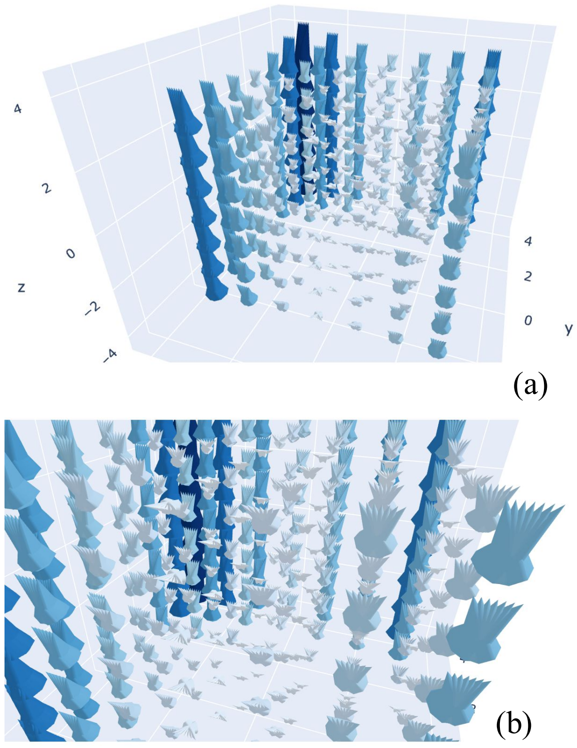


Fig. 27. (xA + A1)(–xA – 1A) + (xA + P1)(–xP – 1A) as a 4-dimensional system (a) and the enlarged area (b).

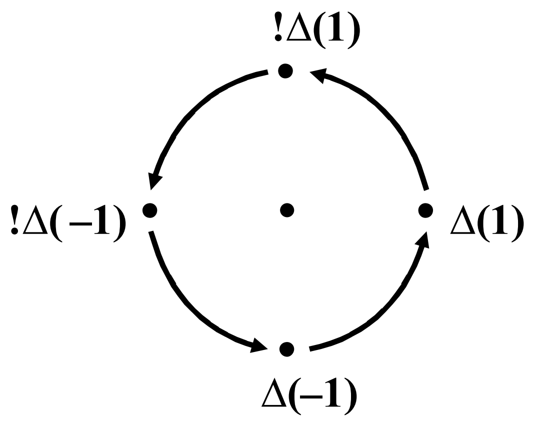


Fig. 28. Z^2 rotation group using Dimensional Gate Operators (DGO) instead of imaginary numbers.

The same cannot be done, as far as the author knows, for A_2, i.e. those lattices that makes use of the Euler or Eisenstein integers, because they are predicated on the abstract algebraic technique that we are reexamining from a new perspective.

The implication of this reexamination is that whenever a sum is calculated in Δ, the answer is immediately transported to !Δ and therefore undergoes a rotation of 90 degrees around some kind of computational axis. Here the rules of arithmetic work differently. But each time we take that answer and preform a new sum with it, it is as if we were rotating it back by an angle of 90 degrees, back to the identity.

But how would this effect normal everyday computations? If we add something to one side of the equation, we must add it to the other. But what we are really doing is subtracting it from the right side, as the rules are inverted. As we can see, the results are entirely the same, so there is no reason to rewrite all of mathematics and start changing the rules of the game. The same must be said of the above diagram. The author doesn't believe that current 'i' notation or method should be replaced with the 'Δ' notation or method since they are ultimately equivalent and may even prove confusing since Δ is used elsewhere in Quantum Mechanics to stand for various items including uncertainty, 'change in' and the discriminant. But it is also used in logic for XOR and XNOR, so we won't be changing that either.

While the author thinks it is important to preserve order and notation in mathematics. Also, it is important to explore new areas and perspectives on even basic mathematical principles and it is this that Logic Gate Algebras (LGAs) can provide.

Trigonometry and Earlier Research

During the course of writing and researching the subject of imaginary numbers, the author encountered several other papers on a similar theme. The first of these is Martin A. Hays, who understood that the XNOR logic gate can be used to explain the rule set of imaginary numbers and has developed several complex systems of numbers based on them. Mostly his work appears to focus on quaternions, neuroscience, economic issues and voting, so the rest of his work though interesting is not strictly applicable here (Marks-Tarlow *et al*., 2015; Hay, 2016).

Another author and mathematician, by the name of Deepak Bhalchandra Gode discovered the same principle. His work shows that complex arithmetic can be undertaken without the use of complex numbers and he uses complex trigonometric functions to do this (Gode, 2014). However, since complex numbers were introduced in large part to avoid the complexity of trigonometry in physics, the utility of this work is questionable (Renou *et al*. 2021. Quantum physics needs complex numbers. arXiv:2101.10873v1). Be that as it may, Gode appears to be the first person to write and publish his work on the subject of the real multiplicative rule sets of imaginary numbers, as far as the author can tell.

CONCLUSION

Logic Gate Algebra (LGA) produces a well-ordered, commutative, and associative algebra that doesn't rely on complex conversions to Lie Groups or Clifford Algebras. It should be noted that one LGA space is not necessarily commutative with another, but this is seen as a strength of the system, rather than a weakness, and has many applications in numerous fields outside of LGA, such as Quantum Mechanics. For more on this, see, my work entitled "The shape of a photon" will discuss. However, it is also worth noting that whatever the LGA is graphing it is not the initial real valued function, or its complex roots function (if one exists). We can tell this because of the results of our Quadratic Formula, where 1 + 3i and 1 – 3i became 2 and 4 instead. The Real function is our parabolic curve, the imaginary curve are the roots provided by the quadratic and the LGA curve appears to be a map between these two with its own set of roots and coordinates. This suggests that the LGA may represent some kind of hereto unknown change of basis or coordinate system.

ACKNOWLEDGMENT

The author would like to thank Dr. Laurence Cuffe, Noirin and Declan O'Neill.

REFERENCES

Campello de Souza, RM., de Oliveira, HM. and Silva D. 2002. The Z transform over finite fields. Proceedings of the IEEE/SBrT International Telecommunication Symposium. pp.362-367. https://arxiv.org/pdf/1502.03371.pdf

Gode, DB. 2014. Complex number theory without imaginary number (i). Open Access Library Journal. 1(7):e856 (13 pages). DOI: http://dx.doi.org/10.4236/oalib.1100856.

Hamilton, WR. 2000. On Quaternions, or on a new system of imaginaries in algebra. The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science. XXV-XXXVI(3rd Series): pp. 92. (Hamilton, WR. 1847. XXXVI. On Quaternions, or on a new system of imaginaries in algebra. Proceedings of the Royal Irish Academy. Philosophical Magazine Series 3. 31(207):214) DOI: https://doi.org/10.1080/14786444708645826.

Hay, MA. 2016. Recursive distinctioning, tetracoding and the symmetry properties of chiral Tetrahedral molecules. Journal of Space Philosophy. 5(2):28-55. http://keplerspaceinstitute.com/wp-content/uploads/2017/11/JSP-Fall-2016-11_Hay-Final.pdf

Karam, R. 2020. Why are complex numbers needed in quantum mechanics? Some answers for the introductory level. American Journal of Physics. 88(1):39-45. DOI: https://doi.org/10.1119/10.0000258.

Marks-Tarlow, T., Hay, MA. and Klitzner, H. 2015. Quaternions, chirality, exchange interactions: A new tool for neuroscience? Society for Chaos Theory in Psychology and Life Science Newsletter (SCTPLS Newsletter). 32(1):8-14. Available online: http://www.markstarlow.com/wp-content/uploads/2017/03/QuaternionFeature-2015.pdf.

Moosavian, SF. and Pius, R. 2019. Hyperbolic geometry and closed bosonic string field theory. Part II. The rules for evaluating the quantum BV master action. Journal of High Energy Physics. 2019(8):177. DOI: https://doi.org/10.1007/JHEP08(2019)177.

Teh, R. 2010. Generalized Jacobi elliptic one-monopole - Type A. International Journal of Modern Physics A. 25(31):5731-5746. DOI: http://dx.doi.org/10.1142/S0217751X10051062.

Received: March 26, 2021; Revised: April 7, 2021;
Accepted: May 14, 2021

Appendix A.

Python code for cubic equation (LGA). For a more general case, see **Appendix C** below.

```
from plotly.graph_objs import *
import plotly
import  plotly.graph_objs as go
import math

#a more cubic form (x+x)(y+y)
space = []
for i in range(-5, 6):
    for c in range(-5, 6):
        y = (i, c)
        space.append(y)

n = []
v = []
h = []
for i in space:
    for c in space:
        u = (i[0]*c[0])*-1
        p = i[1]*c[1]
        g = u+p
        n.append(g)
        v.append(c[0]+i[1])
        h.append(i[0]+c[1])

n1 = np.array(n).reshape(int(math.sqrt(len(n))), int(math.sqrt(len(n))))
n1 = np.rot90(n1, -3)

pz = int(math.sqrt(len(n)))
x = list(range(pz))

data = [go.Mesh3d(
            x = h,
            y = v,
            z = n,
            #alphahull=0,
            colorscale=colourscales[-1],
        ),
    ]
layout = go.Layout(dict(title='Complex',
            titlefont= {"size": 14},
             font={'color':'black'},
             paper_bgcolor= 'white',
             plot_bgcolor= "white",
             hovermode='closest'))
figure = dict(data=data, layout = layout)
iplot(figure)
```

Appendix B.

Python code for the division of (x!Δ+3Δ)/(y!Δ+4Δ).

```
#(x(xnor)+3)/(x(xnor)+4) Division Logic Gate Algebra graphed in 3-D
n = []
v = []
h = []
for i in range(-10, 11):
    for c in range(-10, 11):
        if c != 0:
            u = (i/c)*-1
        else:
            u = 0
        p = 3/4
        g = (u+p)
        n.append(g)
        v.append(i)
        h.append(c)

data = [go.Mesh3d(
            x = h,
            y = v,
            z = n
        ),
    ]
layout = go.Layout(dict(title='Complex',
            titlefont= {"size": 14},
             font={'color':'black'},
             paper_bgcolor= 'white',
             plot_bgcolor= "white",
             hovermode='closest'))

figure = dict(data=data, layout = layout)
iplot(figure)
```

Appendix C.

This python code will generate any number of 3-dimensional surface graphs based on the 16-dimensional Dimensional Gate Operators shown in Figure 20. When you run the code, you will be prompted to enter in a set of logic gates. The logic gates are labelled 'sedens' in the code and are the exact inverse correspondence to those shown in Figure 20.

The one other variable that can be altered in this code is the 'power' variable. If the power variable is set to '2', then we have a quadratic, '3' (as in the example below) gives a cubic curve and so on.

```
from plotly.graph_objs import *
import plotly
import  plotly.graph_objs as go

b = input("Input a Logic Gate: ")

sedens = {'A': [1,1,1,1], 'B': [1,1,1,-1], 'C': [1,1,-1,1], 'D':[1,1,-1,-1], 'E':[1,-1,1,1], 'F':[1,-1,1,-1], 'G':[1,-1,-1,1], 'H':[1,-1,-1,-1], 'I':[-1,1,1,1], 'J':[-1,1,1,-1], 'K':[-1,1,-1,1],
‘L':[-1,1,-1,-1], 'M':[-1,-1,1,1], 'N':[-1,-1,1,-1], 'O':[-1,-1,-
1,1], 'P':[-1,-1,-1,-1]}

power=3
def replace1(boz):
   try:
      boz = boz.replace(',', '')
      boz2 = boz.replace(' ', '')
      boz = boz.split()
      return(boz, boz2)
   except:
      pass

def sed2(list1):
   got=[]
   for i in list1:
      for c in list1:
         if i == c:
            got.append(1)
         else:
            p = (sum(sedens[i])+sum(sedens[c]))
            got.append(p)
   return(got)
funcs = []
srce = (replace1(b))
(list1, list2) = srce
lens = len(list1)
go = (sed2(list1))

if lens <= 2:
   start= -6
   end = 7
elif 2 < lens < 4:
   start= -5
   end = 6
elif 3 < lens < 5:
   start= -4
   end = 5
elif 4 < lens < 6:
   start= -2
   end = 3
else:
   start = -1
   end = 2

print(start, end)
p = product(list2, repeat=power)
print(list1)

t = []
for i in p:
   x = list(i)
   t.append(x)

r0 = []
for i in t:
   x = '\']*perms[\''.join(i)
   r0.append(x)

r = []
for i in r0:
   jk = 'perms[\'' + i + '\']'
   r.append(jk)

perms1 = product(range(start, end), repeat=lens)
#use the perms to build a dictionary with keys and values
from the list,
test=[]
for cc in perms1:
   gg = list(zip(list1, cc))
   test.append(gg)

past = []
for chi in test:
   perms={}
   for rei in chi:
      perms[rei[0]]=rei[1]
   res = [eval(i) for i in r]
   res2 = [res[a]*go[a] for a in range(len(go))]
   piece = sum(res2)
   past.append(piece)

perms2 = product(range(start, end), repeat=lens)

vee=[]
hee=[]
for i in perms2:
   hee.append(i[0])
   vee.append(i[1])

print(len(past), len(vee), len(hee))

data = [go.Mesh3d(
            x = hee,
            y = vee,
            z = past
         ),
      ]
layout = go.Layout(dict(title='Complex Parabolic Curve',
         titlefont= {"size": 14},
           font={'color':'black'},
           paper_bgcolor= 'white',
           plot_bgcolor= "white",
           hovermode='closest'))

figure = dict(data=data, layout = layout)
iplot(figure)
```

Two examples of the above code using a power value of 3 and 4 respectively produces the following outputs:

```
Input a Logic Gate: G, C, C
-5 6
```

```
['G', 'C', 'C']
1331 1331 1331
```

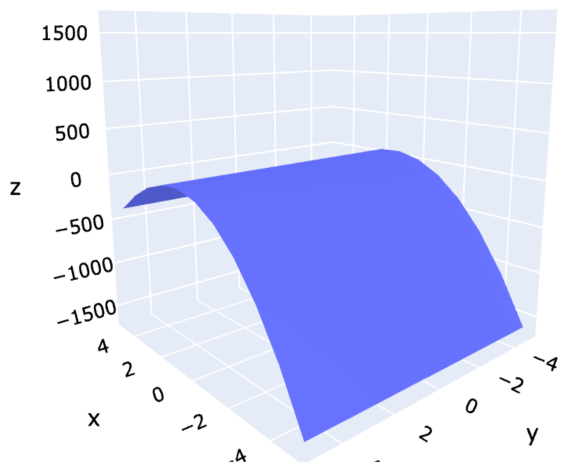


Fig. 29. The complex parabolic curve.

```
Input a Logic Gate: N, N, L
-5 6
['N', 'N', 'L']
1331 1331 1331
```

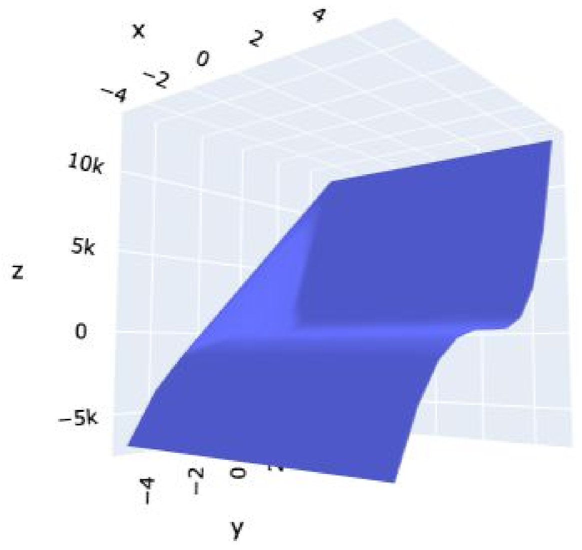


Fig. 30. The complex parabolic curve.

Appendix D.
This python code reproduces the three colored plots shown in Figure 23. None of the variable names have any meaning, they are simply chosen so as not to conflict.

```
import plotly.graph_objects as go
import numpy as np

da1 = []
ma1 = []
so1 = []
la1 = []
bla1 = []
a = 1
b = -5
t = list(range(-50, 50))

for cc in t:
    for yc in t:
        pc = ((cc**2)/2)-((yc**2)/2)+10
        po = (2*(cc**2))+(2*yc)+10
        d = [a, -cc, b, yc]
        e = d[0]*d[2]
        if e < 0:
            e = e*-1
        e1 = d[0]*d[3]
        if e1 < 0:
            e1 = e1*-1
        e2 = d[1]*b
        if e2 < 0:
            e1 = e1*-1
        e3 = (d[1]*b)*-1
        e4 = (d[1]*d[3])*-1
        e5 = (e+e)+(e1+e1)+(e2+e3)+(e4+e4)
        da1.append(e5)
        ma1.append(cc)
        so1.append(yc)
        la1.append(pc)
        bla1.append(po)

fig = go.Figure(data=[go.Mesh3d(x=ma1, y=so1, z=da1, color='cyan')])
fig.add_trace(go.Mesh3d(x=ma1, y=so1, z=la1))
fig.add_trace(go.Mesh3d(x=ma1, y=so1, z=bla1))
fig.show()
```

Appendix E.
This python code reproduces the colored plots shown in Figure 19.

```
import plotly.graph_objects as go
import numpy as np

da2 = []
ma2 = []
so2 = []
la2 = []
bla2 = []

for i in range(-30, 30):
    for x in range(-30, 30):
        y = np.array([i, 1])
        c = np.array([x, 1]).reshape(2, -1)
        g = y*c
        d = (np.abs(g)*2)
        d1 = np.sum(d)
        f = np.array([[0, 2],[-2, 0]])
        m = (g*f)
        m1 = np.sum(m)
        kk = np.sum((d+m)/2)
        bb = ((x**2) + i + 1)
        d2 = d1*-1
        da2.append(i)
        ma2.append(x)
        so2.append(kk)
        la2.append(d2)
        bla2.append(d1)

da3 = np.ravel(da2).tolist()
ma3 = np.ravel(ma2).tolist()
so3 = np.ravel(so2).tolist()
la3 = np.ravel(la2).tolist()
bla3 = np.ravel(bla2).tolist()

fig = go.Figure(data=[go.Mesh3d(x=ma3, y=da3, z=bla3,
color='cyan')])
fig.add_trace(go.Mesh3d(x=da3, y=ma3, z=la3))
fig.show()
```

Appendix F.
This python code recreates the graph in Figure 27.

```
so4 = []
la4 = []
ta4 = []
so5 = []
la6 = []
ta7 = []

for im in range(-3, 4):
    for xs in range(-3, 4):
        for gs in range(-3, 4):
            for km in range(-3, 4):
                y = np.array([im, 1])
                c = np.array([xs, 1]).reshape(2, -1)
                g = y*c
                d = (np.abs(g)*2)
                d1 = np.sum(d)
                f = np.array([[0, 2],[-2, 0]])
                m = (g*f)
                m1 = np.sum(m)
                kk = np.sum((d+m)/2)
                bb = ((xs**2) + im + 1)
                so5.append(kk)
                la6.append(km)
                ta7.append(d1)
                so4.append(im)
                la4.append(xs)
                ta4.append(gs)

ta5= []
for im2 in range(-30, 30):
    for xs2 in range(-30, 30):
        ps = ((im2**2)/2)-((xs2**2)/2)+10
        ta5.append(ps)

fig = go.Figure(data = go.Cone(
    x=so4,
    y=la4,
    z=ta4,
    u=la6,
    v=la4,
    w=so5,
    colorscale='Blues',
    sizemode="absolute",
    sizeref=40))

fig.update_layout(scene=dict(aspectratio=dict(x=1, y=1,
z=0.8), camera_eye=dict(x=1.2, y=1.2, z=0.6)))
fig.show()
```

Appendix G.
This code produces the plot in Figure 5 but can also be modified to produce Figures 3 and 4 as well.

```
import plotly.graph_objects as go
import numpy as np

do2 = []
re2 = []
so2 = []
la2 = []
di2 = []
ta2 = []

t = list(range(-10, 10))

for i in t:
    d = (i**2 + 1)
    do2.append(d)
    di2.append(i)

for x in t:
    e = (((x**2)*-1) - 0.25)
    so2.append(x)
    re2.append(e)

fa2=[]
sa2=[]
ba2=[]
fla2 = []
sre2 = []
fra2 = []

t = list(range(len(do2)))

for i in di2:
    for k in so2:
        e = (do2[i]*re2[k])
        fla2.append(e)
        sre2.append(i)
        fra2.append(k)
        if i < 0 and k >0:
            b = (do2[i]*re2[k])*-1
            fa2.append(b)
            ba2.append(i)
            sa2.append(k)
        elif i > 0 and k < 0:
            b = (do2[i]*re2[k])*-1
            fa2.append(b)
            ba2.append(i)
            sa2.append(k)
        elif i > 0 and k > 0:
            b = (do2[i]*re2[k])
            fa2.append(b)
            ba2.append(i)
            sa2.append(k)
        else:
            b = (do2[i]*re2[k])
            fa2.append(b)
            ba2.append(i)
            sa2.append(k)

fa3 = np.ravel(fa2).tolist()
sa3 = np.ravel(sa2).tolist()
ba3 = np.ravel(ba2).tolist()
fla3 = np.ravel(fla2).tolist()
sre3 = np.ravel(sre2).tolist()
fra3 = np.ravel(fra2).tolist()

fig = go.Figure(data=[go.Mesh3d(x=ba3, y=sa3, z=fa3)])
fig.add_trace(go.Mesh3d(x=sre3, y=fra3, z=fla3))
fig.show()
```