6.S965
Digital Systems Laboratory II

Lecture 8:

IQ and related topics

Administrative

- Week 4 due Monday
- Week 5 will, I’m trying, come out on Friday at noon.

Generalized CORDIC

- The three equations we're iterating on can be generalized to this format

z is our angle accumulator

μ is settable constant

d_i is our control/feedback function for locking into a target

$$x_{i+1} = x_i - \mu d_i y_i 2^{-i}$$

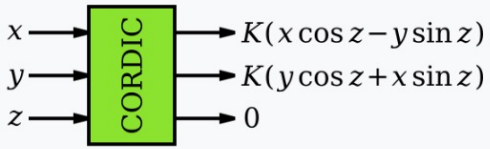
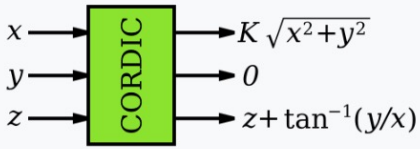
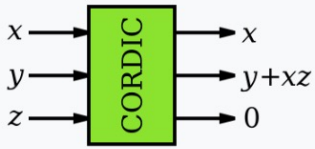
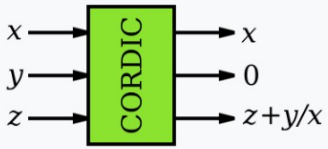
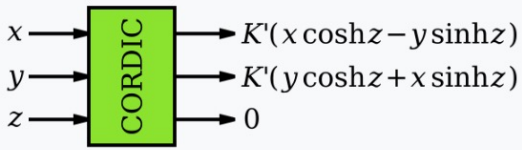
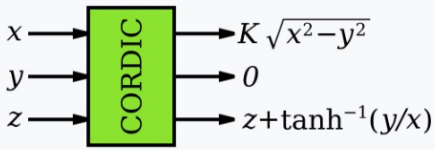
$$y_{i+1} = y_i + d_i x_i 2^{-i}$$

$$z_{i+1} = z_i + d_i x_i 2^{-i}$$

$\text{sgn}(\theta)$ in our walkthrough example

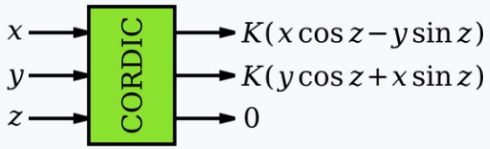
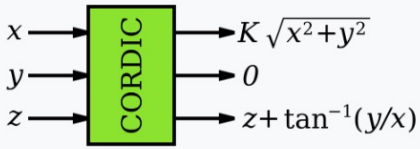
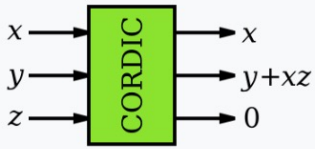
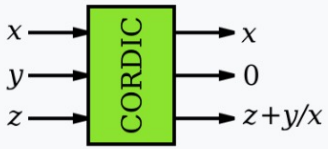
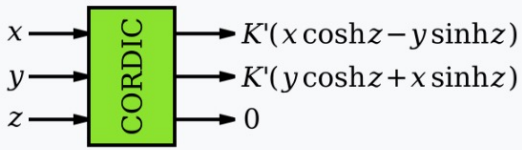
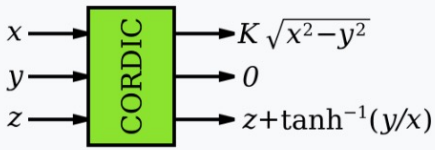
2^{-i} are the $\tan(\alpha_i)$ from our original example

Different Modes

Mode	Rotation	Vectoring
	$d_i = \operatorname{sgn}(z_i), \quad z \to 0$	$d_i = -\operatorname{sgn}(y_i), \quad y \to 0$
Circular $\mu = 1$ $\alpha_i = \tan^{-1}2^{-i}$		
Linear $\mu = 0$ $\alpha_i = 2^{-i}$		
Hyperbolic $\mu = -1$ $\alpha_i = \tanh^{-1}2^{-i}$		

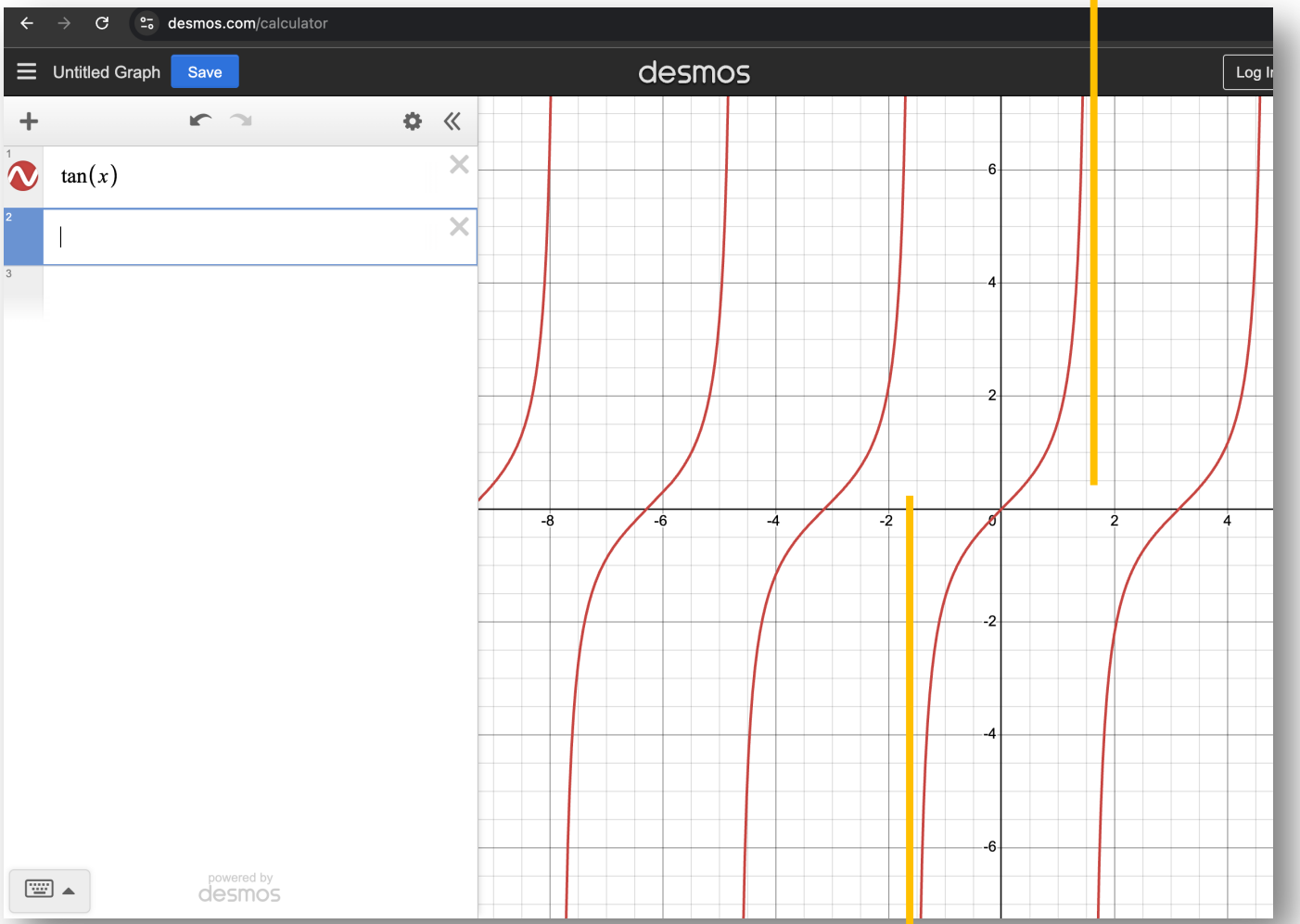
- In hyperbolic mode, iterations 4, 13, 40, 121, ..., j, $3j+1$,... must be repeated. The constant K' given below accounts for this.
- $K = 1.646760258121...$
- $1/K = 0.607252935009...$
- $K' = 0.8281593609602...$
- $1/K' = 1.207497067763...$

How to Actually Get $\sqrt{a}$?

Mode	Rotation	Vectoring
	$d_i = \mathrm{sgn}\,(z_i), \quad z \to 0$	$d_i = -\mathrm{sgn}\,(y_i), \quad y \to 0$
Circular $\mu = 1$ $\alpha_i = \tan^{-1}2^{-i}$		
Linear $\mu = 0$ $\alpha_i = 2^{-i}$		
Hyperbolic $\mu = -1$ $\alpha_i = \tanh^{-1}2^{-i}$		

- In hyperbolic mode, iterations 4, 13, 40, 121, ..., j, $3j+1$,... must be repeated. The constant K' given below accounts for this.
- $K = 1.646760258121...$
- $1/K = 0.607252935009...$
- $K' = 0.8281593609602...$
- $1/K' = 1.207497067763...$

Observe $\tan(x)$

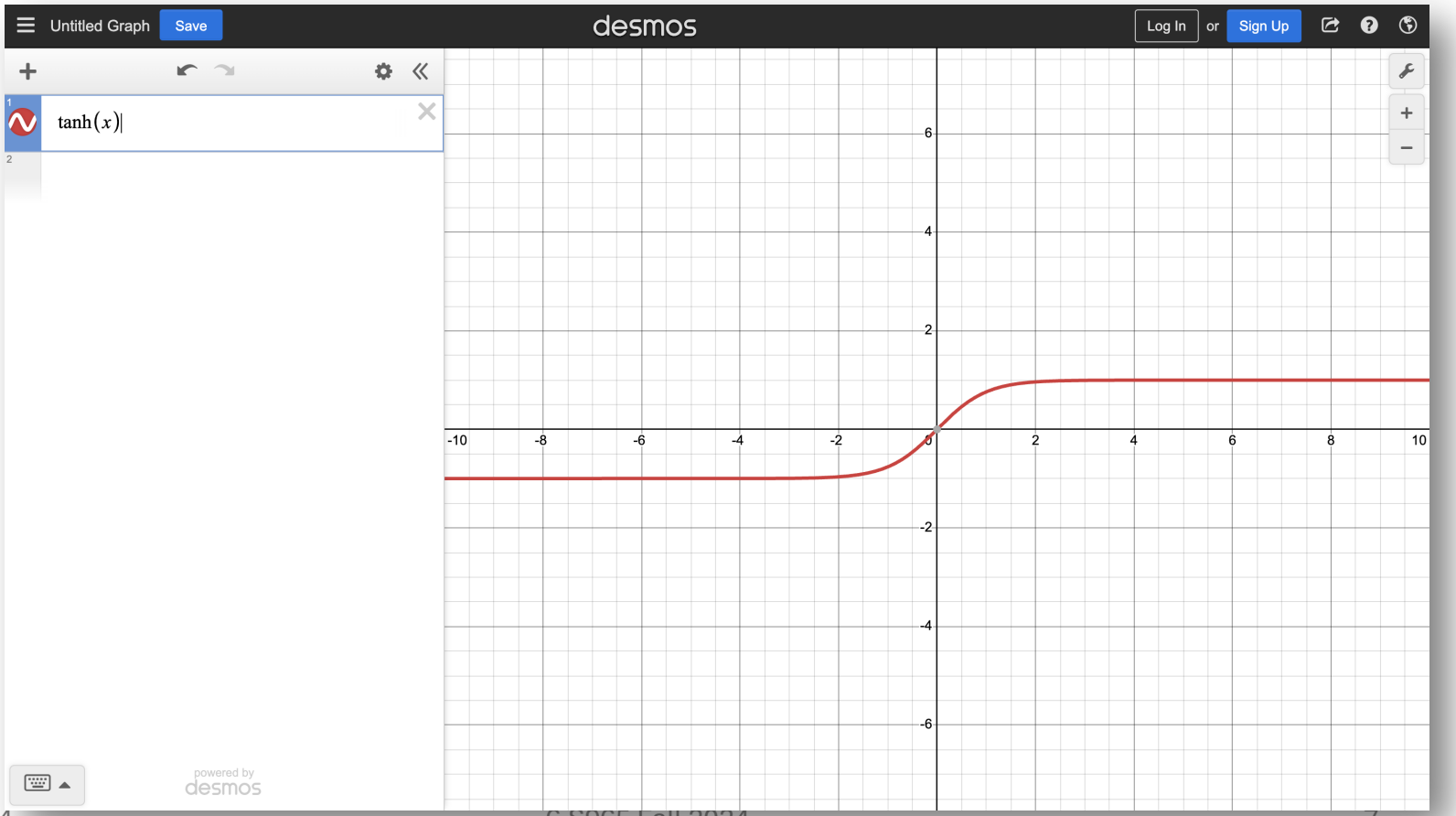


$\pi/2$

$-\pi/2$

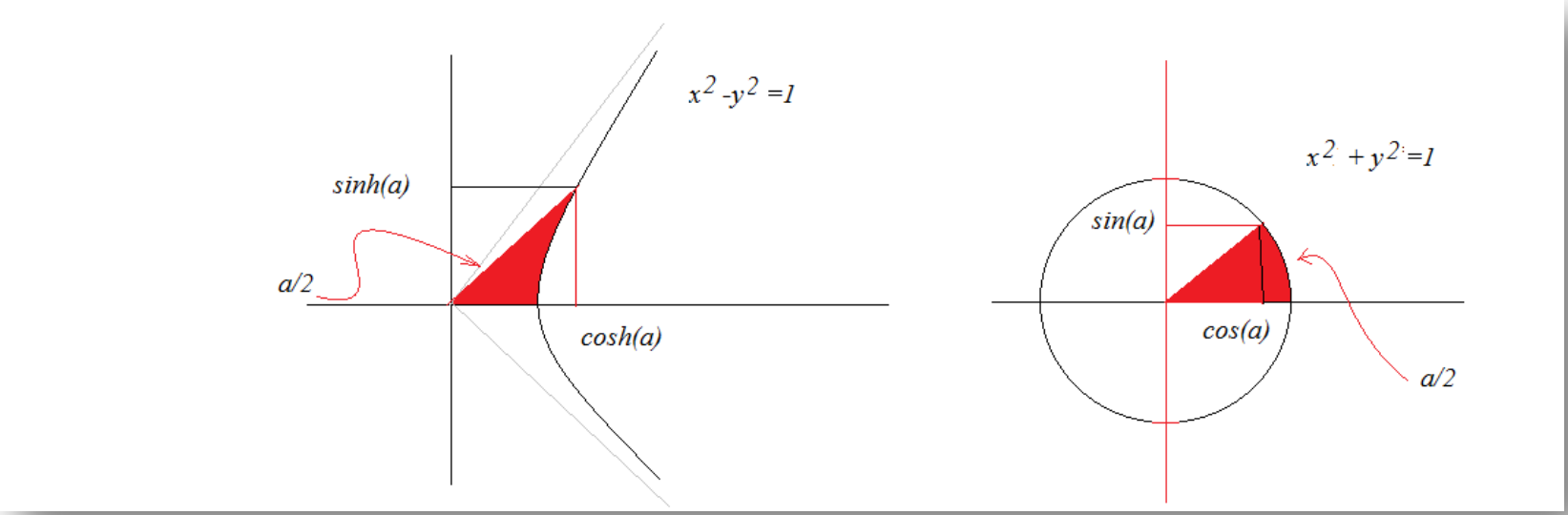
Observe $\tanh(x)$

- Just a different function…still get our values for lookup from it…but kinda weird…



Hyperbolic Functions

- Whereas regular trig functions are following around the unit circle… $x^2 + y^2 = 1$



- Hyperbolic trig functions are following the hyperbola: $x^2 - y^2 = 1$

Mode	Rotation	Vectoring
	$d_i = \text{sgn}(z_i), \quad z \to 0$	$d_i = -\text{sgn}(y_i), \quad y \to 0$
Circular $\mu = 1$ $\alpha_i = \tan^{-1}2^{-i}$	$x \to K(x\cos z - y\sin z)$, $y \to K(y\cos z + x\sin z)$, $z \to 0$	$x \to K\sqrt{x^2+y^2}$, $y \to 0$, $z \to z+\tan^{-1}(y/x)$
Linear $\mu = 0$ $\alpha_i = 2^{-i}$	$x \to x$, $y \to y+xz$, $z \to 0$	$x \to x$, $y \to 0$, $z \to z+y/x$
Hyperbolic $\mu = -1$ $\alpha_i = \tanh^{-1}2^{-i}$	$x \to K'(x\cosh z - y\sinh z)$, $y \to K'(y\cosh z + x\sinh z)$, $z \to 0$	$x \to K\sqrt{x^2-y^2}$, $y \to 0$, $z \to z+\tanh^{-1}(y/x)$

- In hyperbolic mode, iterations 4, 13, 40, 121, ..., j, $3j+1$,... must be repeated. The constant K' given below accounts for this.
- $K = 1.646760258121...$
- $1/K = 0.607252935009...$
- $K' = 0.8281593609602...$
- $1/K' = 1.207497067763...$

$$x_{i+1} = x_i - \mu d_i y_i 2^{-i}$$

$$y_{i+1} = y_i + d_i x_i 2^{-i}$$

$$z_{i+1} = z_i + d_i x_i 2^{-i}$$

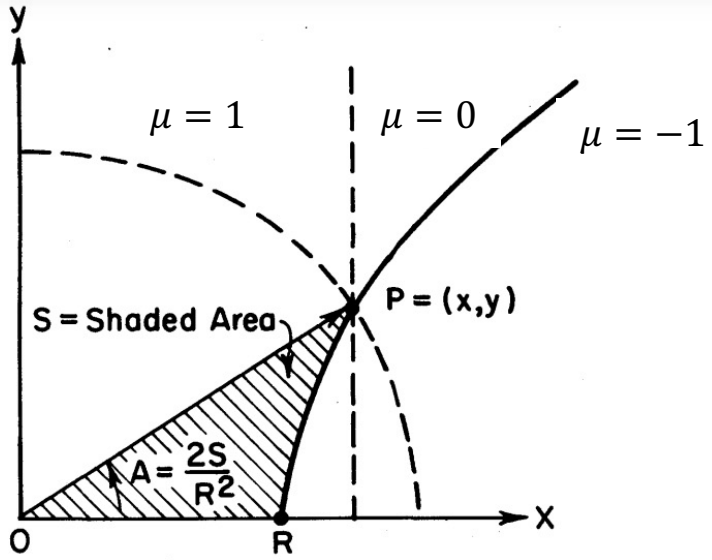
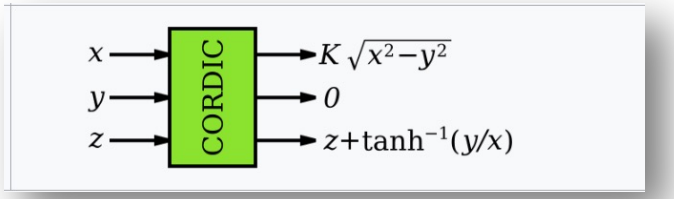


Figure 1—Angle A and Radius R of the vector $P = (x, y)$

J.S. Walther, "A Unified Algorithm for Elementary Functions," Conference Proceedings, Spring Joint Computer Conference, May 1971, pp. 379-385

How to Actually Get $\sqrt{a}$?



- You then need to do:
  - $x = a + 0.25$
  - $y = a - 0.25$
  - So that…$\sqrt{(a+0.25)^2 - (a-0.25)^2} = \sqrt{a}$

- AMD/Xilinx has a pretty decent writeup of how to do it in a low-level digital form.

From the Xilinx/AMD Docs…

That is, given input x, it computes the output sqrt (x). The CORDIC processor is implemented using building blocks from the Xilinx blockset.

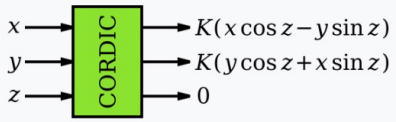
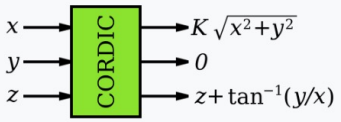
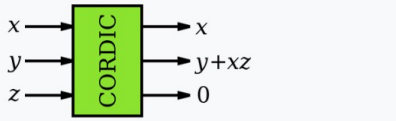
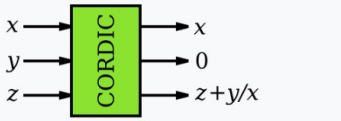
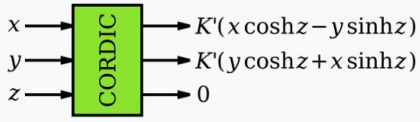
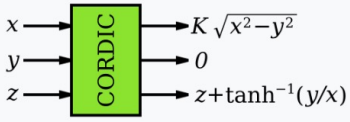
- The square root is calculated indirectly by the CORDIC algorithm by applying the identity listed as follows. sqrt (w) = sqrt { $(w + 0.25)^2 - (w - 0.25)^2$ }
- The CORDIC square root algorithm is implemented in the following 4 steps:
  1. Co-ordinate Rotation: The CORDIC algorithm converges only for positive values of x. If x < zero, the input data is converted to a non-negative number. If x = 0, a zero detect flag is passed to the co-ordinate correction stage. The square root circuit has been designed to converge for all values of x, except for the most negative value.
  2. Normalization: The CORDIC algorithm converges only for x between 0.25 (inclusive) and 1. During normalization, the input x is shifted to the left till it has a 1 in the most significant non-signed bit. If the left shift results in an odd number of shift values, a right shift is performed resulting in an even number of left shifts. The shift value is divided by 2 and passed on to the co-ordinate correction stage. The square root is derived using the identity sqrt (w) = sqrt $\{(w + 0.25)^2 - (w - 0.25)^2\}$. Based on this identity the input x gets mapped to, X = x + 0.25 and Y = x - 0.25.
  3. Hyperbolic Rotations: For sqrt $(X^2 - Y^2)$ calculation, the resulting vector is rotated through progressively smaller angles, such that Ygoes to zero.
  4. Co-ordinate Correction: If the input was negative and a left shift was applied to x, this step assigns the appropriate sign to the output and multiplies it with 2^{-shift}. If the input was zero, the zero detect flag is used to set the output to 0.

CORDIC Square Root Convergence

- If you sum up all the possible angle they converge to about 99.88 degrees
- Dictates the range over which cordic functions can “converge” through multiple iterations
- Same thing with hyperbolic…but their possible angle total approaches: 64.74 degrees (significantly lower since can’t do for i=0)

i	$\alpha_i = \tan^{-1}(2^{-i})$	
	Degrees	Radians
0	45.00	0.7854
1	26.57	0.4636
2	14.04	0.2450
3	7.13	0.1244
4	3.58	0.0624
5	1.79	0.0312
6	0.90	0.0160
7	0.45	0.0080
8	0.22	0.0040
9	0.11	0.0020

Another Interesting Thing…

Mode	Rotation	Vectoring
	$d_i = \operatorname{sgn}(z_i), \quad z \to 0$	$d_i = -\operatorname{sgn}(y_i), \quad y \to 0$
Circular $\mu = 1$ $\alpha_i = \tan^{-1}2^{-i}$		
Linear $\mu = 0$ $\alpha_i = 2^{-i}$		
Hyperbolic $\mu = -1$ $\alpha_i = \tanh^{-1}2^{-i}$		

- In hyperbolic mode, iterations 4, 13, 40, 121, ..., j, $3j+1$,... must be repeated. The constant K' given below accounts for this.
- $K = 1.646760258121...$
- $1/K = 0.607252935009...$
- $K' = 0.8281593609602...$
- $1/K' = 1.207497067763...$

Source...

- This 1971 paper is what everyone points to as justification for the repeated sequence for convergence

The magnitude of each element of the sequence may be predetermined, but the direction of rotation must be determined at each step such that

$$|A_{i+1}| = ||A_i| - \alpha_i| \tag{22}$$

The sum of the remaining rotations must at each step be sufficient to bring the angle to at least within α_{n-1} of zero, even in the extreme case where $A_i = 0$, $|A_{i+1}| = \alpha_i$. Thus,

$$\alpha_i - \sum_{j=i+1}^{n-1} \alpha_j < \alpha_{n-1} \tag{23}$$

The domain of convergence is limited by the sum of the rotations.

$$|A_0| - \sum_{j=0}^{n-1} \alpha_j < \alpha_{n-1} \tag{24}$$

$$\text{-}_1 + \sum_{j=0}^{n-1} \alpha_j \tag{25}$$

es to within α_{n-1} of zero
he following theorem.

$$\text{-}_1 + \sum_{j=i}^{n-1} \alpha_j \tag{26}$$

TABLE II—Shift Sequences for a binary code

radix ρ	coordinate system m	shift sequence $F_{mi}; i \geq 0$	domain of convergence max $\lvert A_0 \rvert$	radius factor K
2	1	0, 1, 2, 3, 4, i, . . .	∼1.74	∼1.65
2	0	1, 2, 3, 4, 5, i+1, . . .	1.0	1.0
2	−1	1, 2, 3, 4, 4, 5, ... *	∼1.13	∼0.80

* for $m = -1$ the following integers are repeated:
{4, 13, 40, 121, . . ., k, $3k$+1, . . .}

Table II shows some F sequences, convergence domains, and radius factors for a binary code.

The hyperbolic mode ($m = -1$) is somewhat complicated by the fact that for $\alpha_i = \tanh^{-1}(2^{-i})$ the convergence criterion (23) is not satisfied. However, it can be shown that

$$\alpha_i - \left(\sum_{j=i+1}^{n-1} \alpha_j \right) - \alpha_{3i+1} < \alpha_{n-1} \tag{33}$$

and that therefore if the integers {4, 13, 40, 121, ..., k, $3k$+1, ...} in the F_i sequence are repeated then (23) becomes true.

J.S. Walther, "A Unified Algorithm for Elementary Functions," Conference Proceedings, Spring Joint Computer Conference, May 1971, pp. 379-385

Square root convergence in Practice

- Normalization: **The CORDIC algorithm converges only for x between 0.25 (inclusive) and 1.** During normalization, the input x is shifted to the left till it has a 1 in the most significant non-signed bit. If the left shift results in an odd number of shift values, a right shift is performed resulting in an even number of left shifts. The shift value is divided by 2 and passed on to the co-ordinate correction stage. The square root is derived using the identity sqrt (w) = sqrt {$(w + 0.25)^2 - (w - 0.25)^2$}. Based on this identity the input x gets mapped to, X = x + 0.25 and Y = x - 0.25.

Overcoming Algorithm Input Range Limitations

Many square root algorithms normalize the input value, v, to within the range of [0.5, 2). This pre-processing as well as large input value ranges.

https://www.mathworks.com/help/fixedpoint/ug/compute-square-root-using-cordic.html

So these are coming from the assertion

- Keeping in mind: $\sqrt{(a+0.25)^2-(a-0.25)^2}$ which is forcing the starting value of the x and y values…

Wrote some code to test it

```
import sys

a = float(sys.argv[1])

x = a+.25
y = a-.25

for i in range(1,20):
  if y >0:
    xn = x - 1/(2**i)*y
    yn = y - 1/(2**i)*x
  else:
    xn = x + 1/(2**i)*y
    yn = y + 1/(2**i)*x
    print(f"x:{xn}, y:{yn}")
    x = xn
    y = yn
print(x/0.828)
```

So I ran it…

```
python3 cordic_test.py 4
x:2.375, y:1.625
x:1.96875, y:1.03125
x:1.83984375, y:0.78515625
x:1.790771484375, y:0.670166015625
x:1.7698287963867188, y:0.6142044067382812
x:1.760231852531433, y:0.5865508317947388
x:1.7556494241580367, y:0.572799020446837
x:1.7534119279844163, y:0.5659410148837196
x:1.7523065744397215, y:0.562516382211875
x:1.7517572420352177, y:0.5608051453227738
x:1.751483411397853, y:0.5599497951069363
x:1.751346704904907, y:0.5595221868522006
x:1.7512784038567073, y:0.559308399412637
x:1.7512442663811572, y:0.5592015098616203
x:1.7512272009053924, y:0.559148066127905
x:1.7512186689829967, y:0.5591213445214459
x:1.7512144032256685, y:0.5591079837833097
x:1.7512122703979716, y:0.5591013034305142
x:1.7512112039968648, y:0.5590979632581845
2.1149893768078076
```

So I ran it...

```
python3 cordic_test.py 0.81
x:0.78, y:0.030000000000000027
x:0.7725, y:-0.16499999999999998
x:0.751875, y:-0.06843749999999998
x:0.74759765625, y:-0.021445312499999987
x:0.746927490234375, y:0.001917114257812512
x:0.7468975353240966, y:-0.009753627777099597
x:0.746821335107088, y:-0.003918490782380092
x:0.7468060285024694, y:-0.0010012199421180297
x:0.7468040729947699, y:0.0004573855823008558
x:0.7468036263291622, y:-0.0002719152702330992
x:0.7468034935580341, y:9.273493793543702e-05
x:0.7468034709176684, y:-8.95901337340049e-05
x:0.7468034599813728, y:1.5723993369995485e-06
x:0.7468034598854011, y:-4.400886653100416e-05
x:0.7468034585423571, y:-2.121823359993113e-05
x:0.7468034582185925, y:-9.822917154887839e-06
x:0.7468034581436496, y:-4.125258934836321e-06
x:0.7468034581279129, y:-1.2764298250964468e-06
x:0.7468034581254783, y:1.47984729743475e-07
0.9019365436298048
```

Conclusions

- Seems to converge for input values of 0 to 2
- Beyond that it doesn't converge, and this is because for hyperbolics, $\mu = -1$ so:

$$x_{i+1} = x_i + d_i y_i 2^{-i}$$

$$y_{i+1} = y_i + d_i x_i 2^{-i}$$

- Whereas in original:

$$x_{i+1} = x_i - d_i y_i 2^{-i}$$

$$y_{i+1} = y_i + d_i x_i 2^{-i}$$

I/Q Format

Motivation

- One of the reasons I wanted to look at CORDIC stuff was it would let us think more about doing trig functions and other operations
- Forms an important part of a lot of how FPGAs are used, particularly in signal processing applications.
- A lot of signals come in and you need to do very quick math to extract/refine the information from them.

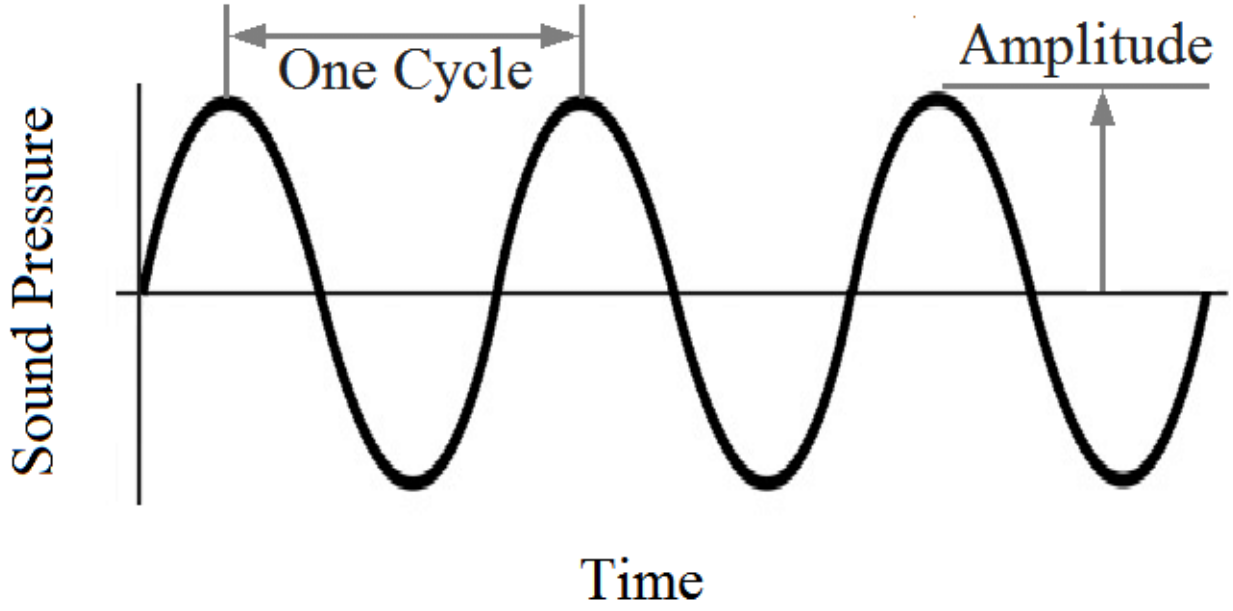
So most wireless data is transferred on electromagnetic sine waves

- The frequencies of these waves are chosen such that they can propagate effectively through free space

- Different frequencies travel in different ways.

- We throw this electromagnetic energy around and use it to convey information

Sine wave looks like this



$$v(t) = A\sin(2\pi ft + \phi)$$

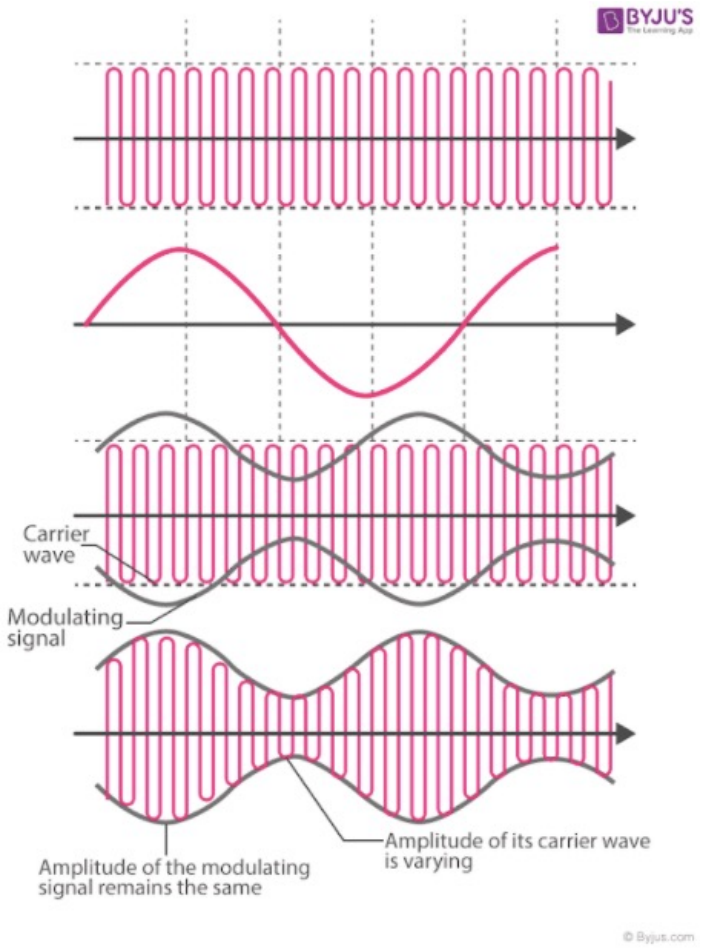
If you needed to convey information on this wave what could you do?

$$v(t) = A\cos(2\pi f t + \phi)$$

- You could:
  - Vary the amplitude (Amplitude modulation)

Amplitude Modulation

- Keep frequency the same and then modulate then modulate the amplitude of your carrier wave…
- Usually something as simple as a low-pass filter and some non-linearity can get the info out



https://byjus.com/jee/amplitude-modulation/

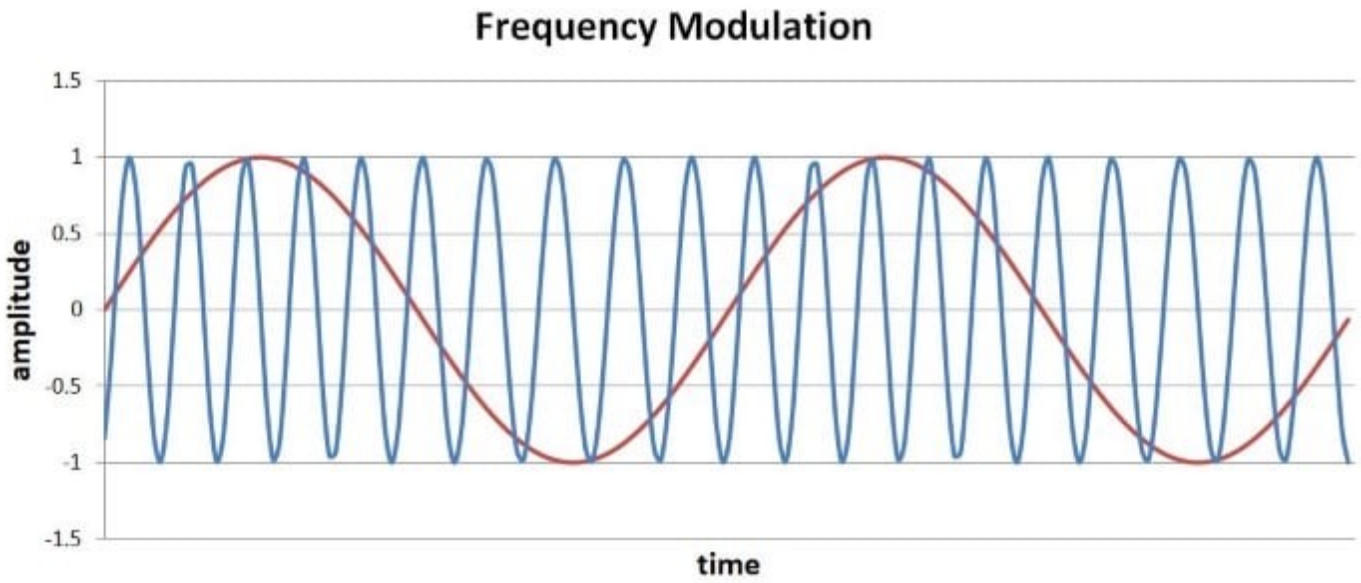
If you needed to convey information on this wave what could you do?

$$v(t) = A\cos(2\pi f t + \phi)$$

- You could:
  - Vary the frequency (Frequency modulation)

Frequency Modulation

- You keep a constant amplitude and then vary your frequency around a constant center frequency
- Circuits can extract those deviations to get the info



https://www.allaboutcircuits.com/textbook/radio-frequency-analysis-design/radio-frequency-modulation/frequency-modulation-theory-time-domain-frequency-domain/

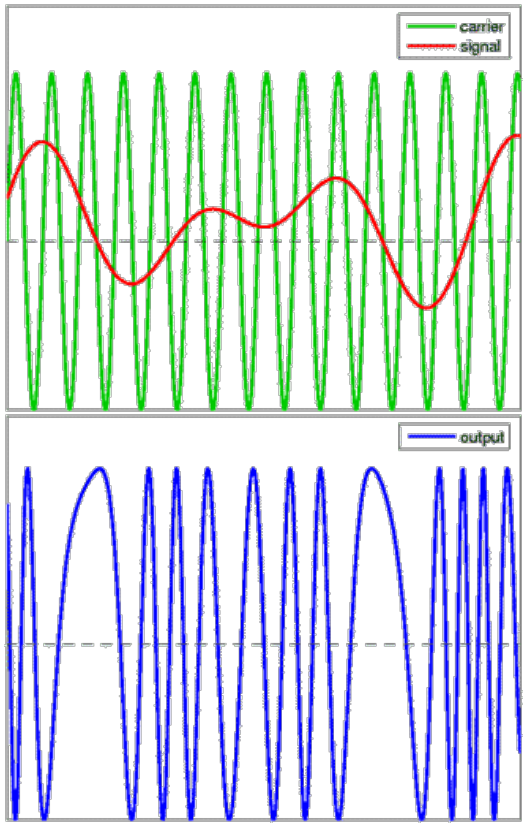
If you needed to convey information on this wave what could you do?

$$v(t) = A\cos(2\pi f t + \phi)$$

- You could:
  - Vary the phase (Phase modulation)

Phase Modulation

- Varying the phase over time to convey your signal can be done
- But in a purely analog setting it is quite rare



https://commons.wikimedia.org/wiki/File:Phase_Modulation.png

Reason for that…

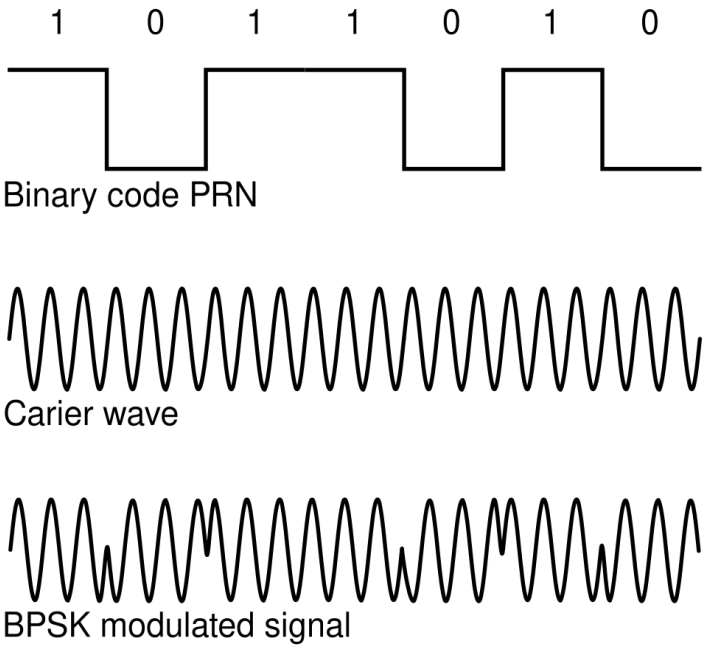
- Varying the phase of a signal over time starts to get tangled with the frequency of the signal

$$v(t) = A\cos(2\pi f t + \phi(t))$$

- Frequency really is just time-varying phase after all…

Phase Modulation

- You do see phase modulation in lots of digital settings however. Distinct changes in the phase of a signal are easier to detect and less ambiguous than continuous “analog” phase modulation



https://commons.wikimedia.org/wiki/File:Phase_modulation_BPSK_GPS.svg

What do modern systems use?

- Really depends

- Not a ton of regular only-AM anymore
- FM still common
- PM still common
- And as we'll see some other combinations.

Returning to our generic Sine Wave…

- Let's say all three portions of the signal can be functions of time to convey information…

$$v(t) = A(t)\cos(2\pi f t + \phi(t))$$

Trig Identities

en.wikipedia.org/wiki/List_of_trigonometric_identities

Contents hide

$$\operatorname{sgn}(\cos\theta) = \operatorname{sgn}(\sec\theta) = \begin{cases} +1 & \text{if } -\frac{1}{2}\pi < \theta < \frac{1}{2}\pi \\ -1 & \text{if } -\pi < \theta < -\frac{1}{2}\pi \text{ or } \frac{1}{2}\pi < \theta < \pi \\ 0 & \text{if } \theta \in \left\{-\frac{1}{2}\pi, \frac{1}{2}\pi\right\} \end{cases}$$

$$\operatorname{sgn}(\tan\theta) = \operatorname{sgn}(\cot\theta) = \begin{cases} +1 & \text{if } -\pi < \theta < -\frac{1}{2}\pi \text{ or } 0 < \theta < \frac{1}{2}\pi \\ -1 & \text{if } -\frac{1}{2}\pi < \theta < 0 \text{ or } \frac{1}{2}\pi < \theta < \pi \\ 0 & \text{if } \theta \in \left\{-\frac{1}{2}\pi, 0, \frac{1}{2}\pi, \pi\right\} \end{cases}$$

The trigonometric functions are periodic with common period 2π, so for values of θ outside the interval $(-\pi, \pi]$, they take repeating values (see § Shifts and periodicity above).

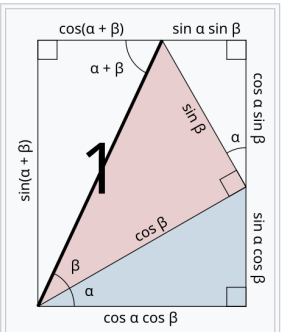
Angle sum and difference identities [edit]

See also: Proofs of trigonometric identities § Angle sum identities, and Small-angle approximation § Angle sum and difference

These are also known as the *angle addition and subtraction theorems* (or *formulae*).

$$\begin{aligned} \sin(\alpha+\beta) &= \sin\alpha\cos\beta + \cos\alpha\sin\beta \\ \sin(\alpha-\beta) &= \sin\alpha\cos\beta - \cos\alpha\sin\beta \\ \cos(\alpha+\beta) &= \cos\alpha\cos\beta - \sin\alpha\sin\beta \\ \cos(\alpha-\beta) &= \cos\alpha\cos\beta + \sin\alpha\sin\beta \end{aligned}$$

The angle difference identities for $\sin(\alpha-\beta)$ and $\cos(\alpha-\beta)$ can be derived from the angle sum versions by substituting $-\beta$ for β and using the facts that $\sin(-\beta) = -\sin(\beta)$ and $\cos(-\beta) = \cos(\beta)$. They can also be derived by using a slightly modified version of the figure for the angle sum identities, both of which are shown here.



Any sine wave that is running…

$$v(t) = A(t)\cos(2\pi f t + \phi(t))$$

- Can be said to actually be made of two other sinusoidal waves:

$$\ldots = \cos(2\pi f t) \cdot A(t)\cos(\phi(t)) - \sin(2\pi f t) \cdot A(t)\sin(\phi(t))$$

Any sine wave that is running…

$$A(t)\cos\left(2\pi f t + \phi(t)\right) =$$

$$= \underbrace{\cos(2\pi f t)\cdot A(t)\cos(\phi(t))}_{\textit{In-phase component}} - \underbrace{\sin(2\pi f t)\cdot A(t)\sin(\phi(t))}_{\textit{Quadrature component}}$$

I/Q signals

$$= \cos(2\pi f t) \cdot A(t)\cos(\phi(t)) - \sin(2\pi f t) \cdot A(t)\sin(\phi(t))$$

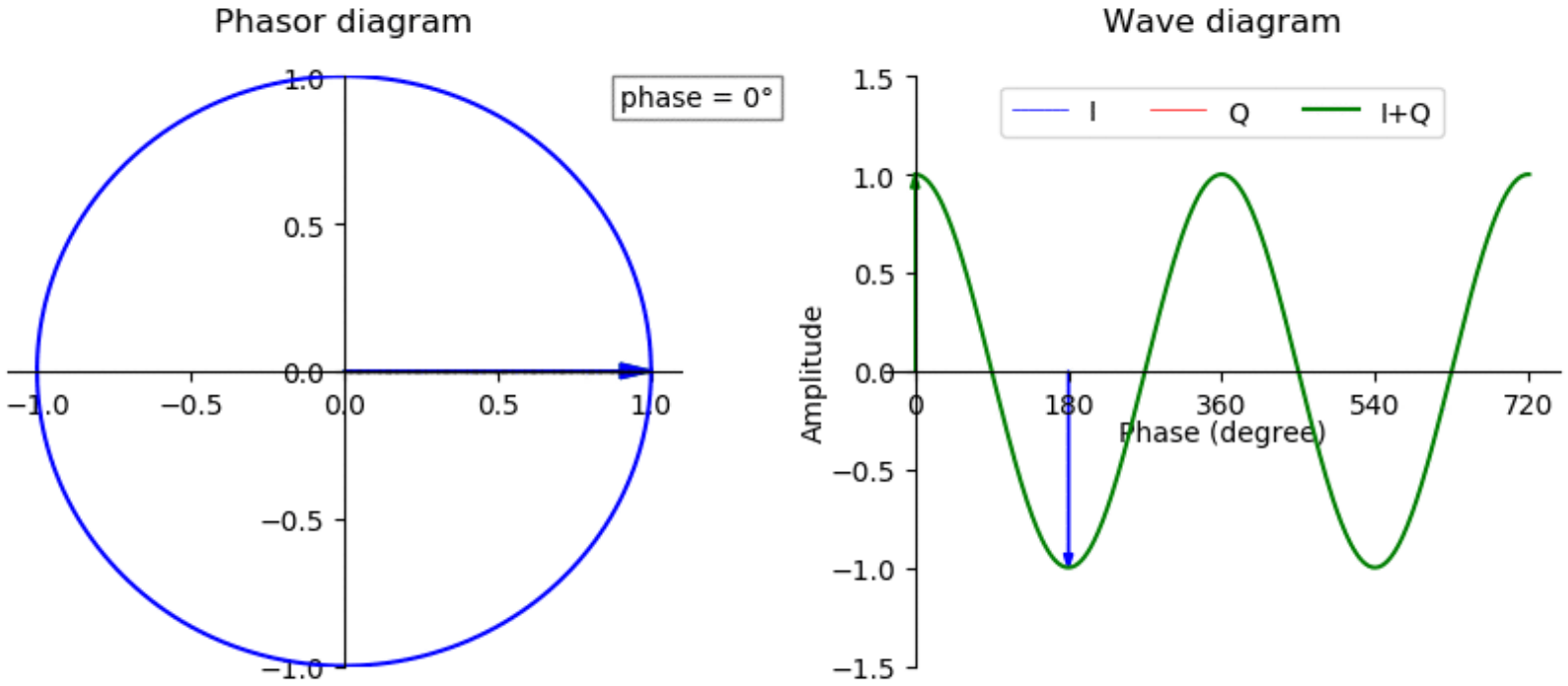
- You have two signals of fixed frequency and modulatable amplitudes.
- We don't need to think of those two amplitudes as related to the original signal, instead just think of them as modulatable signals on their own:

$$= \cos(2\pi f t) \cdot I(t) - \sin(2\pi f t) \cdot Q(t)$$

- This $I(t)$ and $Q(t)$ signal contain your information and can be useful in both creating and analyzing modulated signals.

Plot these I/Q values out on the complex plane…help us visualize signals and information…

quadrature component (y axis)

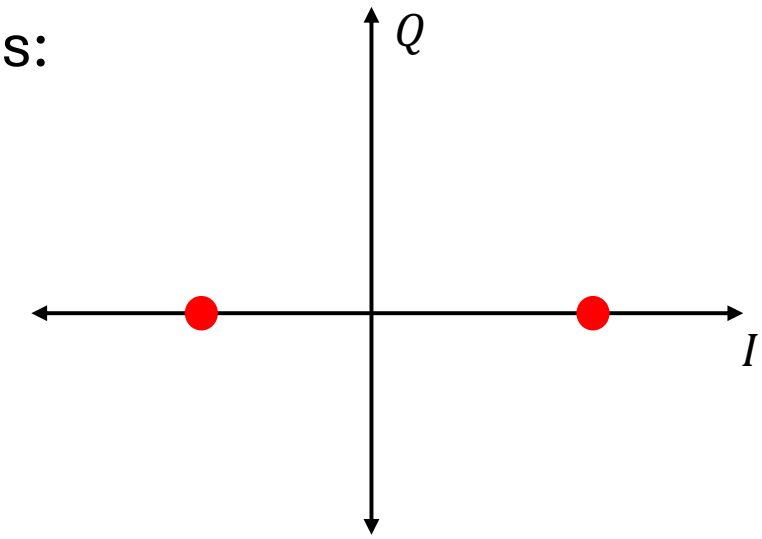


In-phase component (x axis)

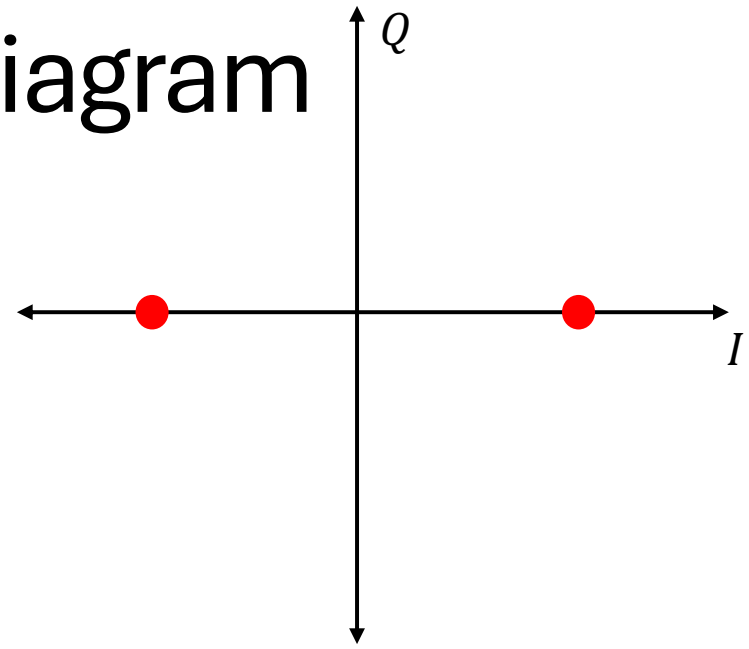
$$v(t) = \cos(2\pi f t) \cdot I(t) - \sin(2\pi f t) \cdot Q(t)$$

Varying I/Q

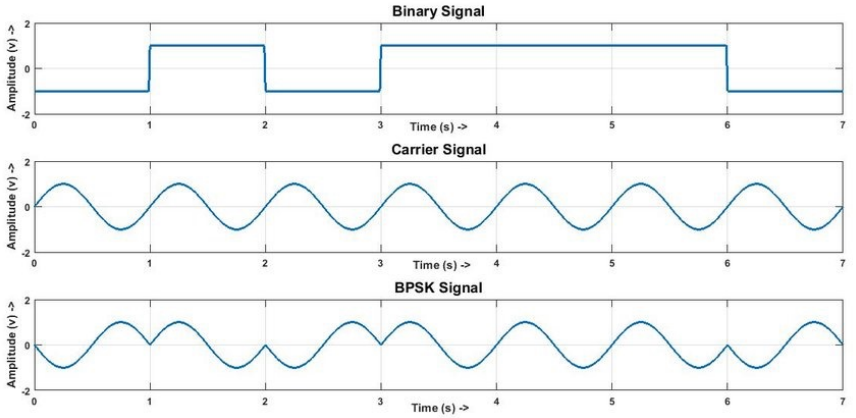
- Let's say we want to send binary 1's and 0's.
- One way to do this would be by doing the following:
  - $Q(t)$ is always set to 0
  - $I(t)$ is +1 for binary 1 and -1 for binary 0
  - We'd get an I-Q plot like this:



Constellation Plot/Diagram



- Plotting out the possible I/Q combinations in the 2D space is known as a constellation diagram
- Each constellation diagram is a way to depict all the forms of a signal that could be expected
- Time series plot of this signal shown here-→

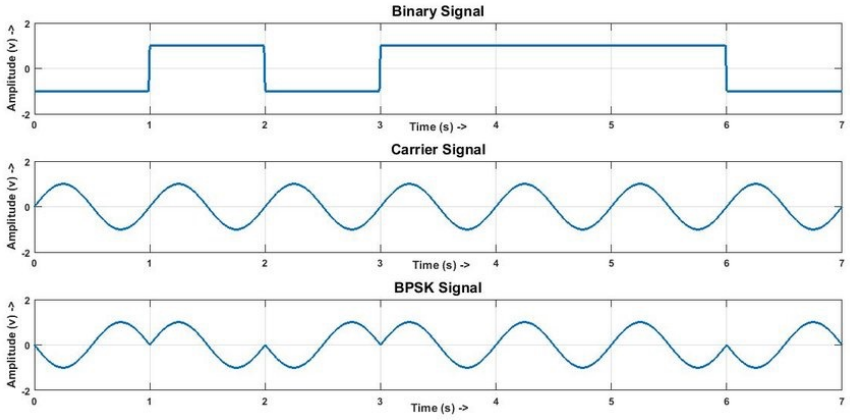


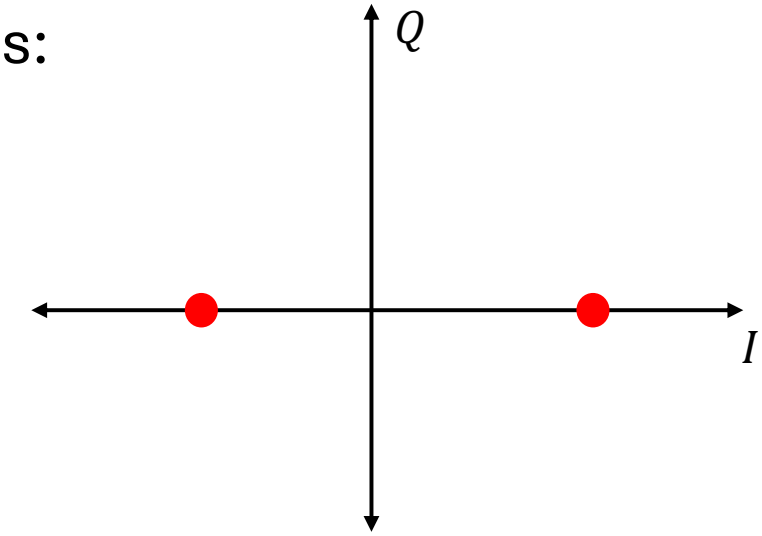
https://www.researchgate.net/figure/Implementation-of-Binary-Phase-Shift-Keying_fig10_348364945

$$v(t) = \cos(2\pi f t) \cdot I(t) - \sin(2\pi f t) \cdot Q(t)$$

Binary Phase Shift Keying

- Let's say we want to send binary 1's and 0's.
- One way to do this would be by doing the following:
  - $Q(t)$ is always set to 0
  - $I(t)$ is +1 for binary 1 and -1 for binary 0
  - We'd get an I-Q plot like this:





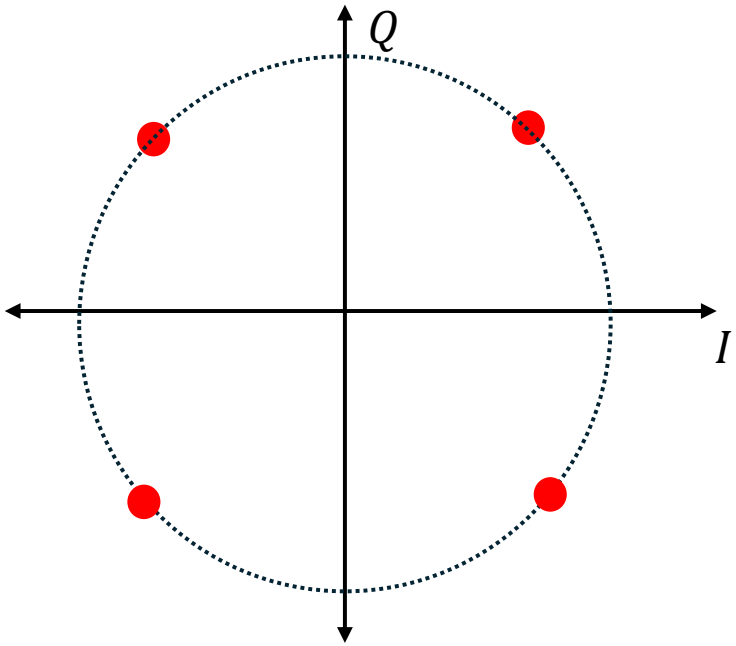
$$v(t) = \cos(2\pi f t) \cdot I(t) - \sin(2\pi f t) \cdot Q(t)$$

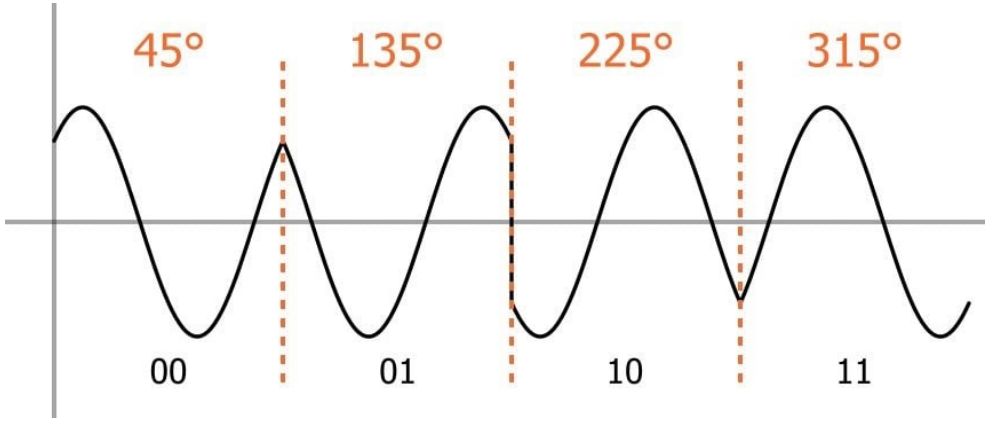
Can we do more?

- Right now, we have only two states, so can only send one bit per ”beat” of data.
- Could we do more? What if we use both dimensions of our I/Q signal?
- Have:
  - I be +0.707 or -0.707
  - Q be +0.707 or -0.707
- What would that constellation plot look like?

Quad-Phase Shift Keying

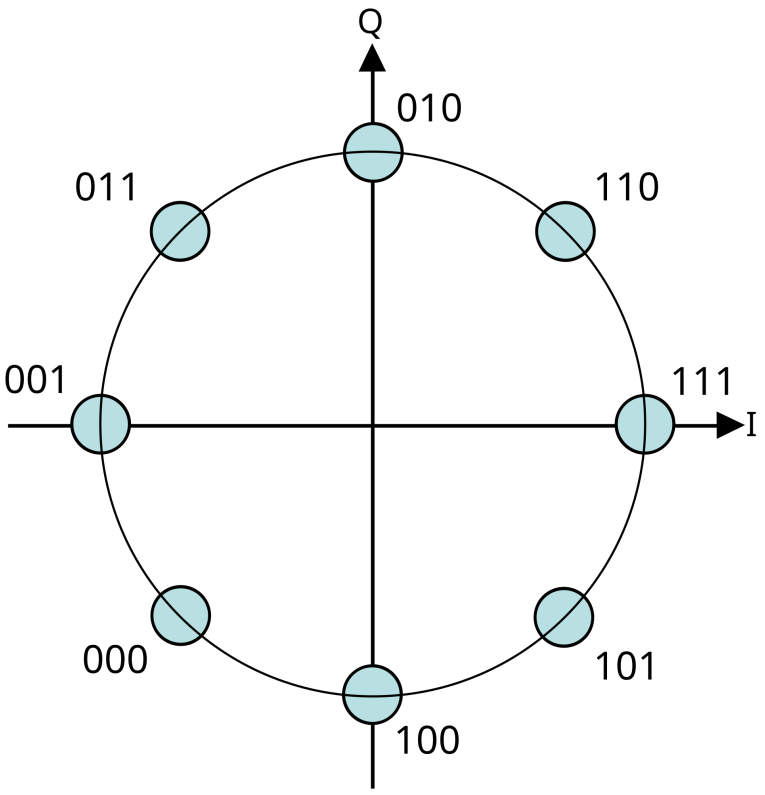
- Constant amplitude, but four distinct phases, each 90 degrees separated from eachother. Each measurement gives two bits of info





Keep going

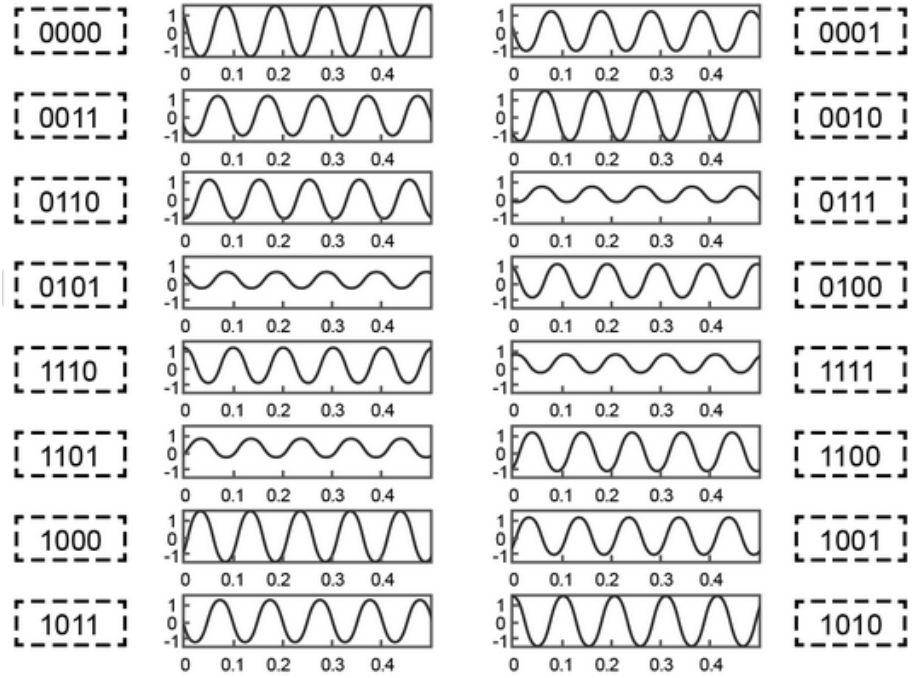
- Instead of assign I/Q separately, start to have pairs of I/Q values that position your signals all over the constellation plot as desired.
- 8 points separated by 45 degrees gives 8psk
- Each measurement gives 3 bits.



https://en.wikipedia.org/wiki/Constellation_diagram

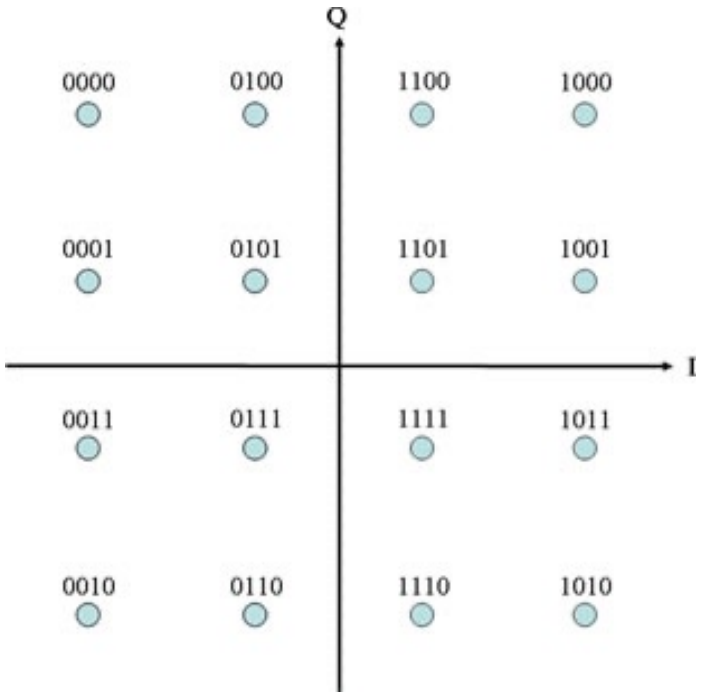
Can also not just stick to unit circle (vary amplitude of signal!

- Called Quadrature amplitude modulation.
- 16 QAM has 16 different possibilities:
- In time they look like:



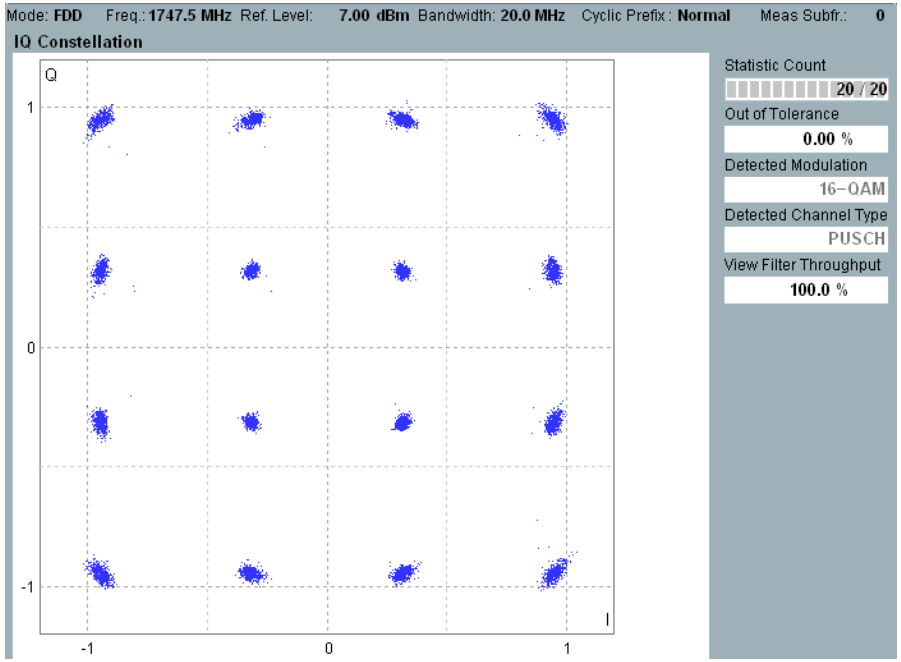
Can also not just stick to unit circle (vary amplitude of signal!

- Called Quadrature amplitude modulation.
- 16 QAM has 16 different possibilities:
- In constellation plot:



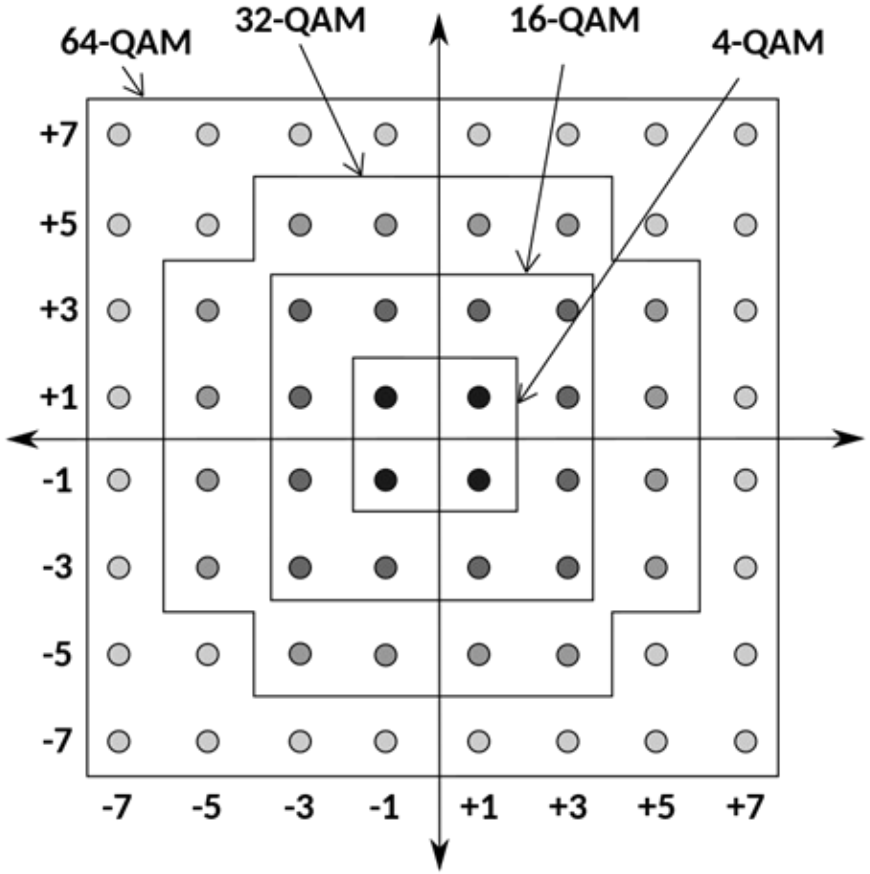
And this can keep going.

- This methodology of both modulation but also of analyzing signals can allow for relatively easy and productive analysis

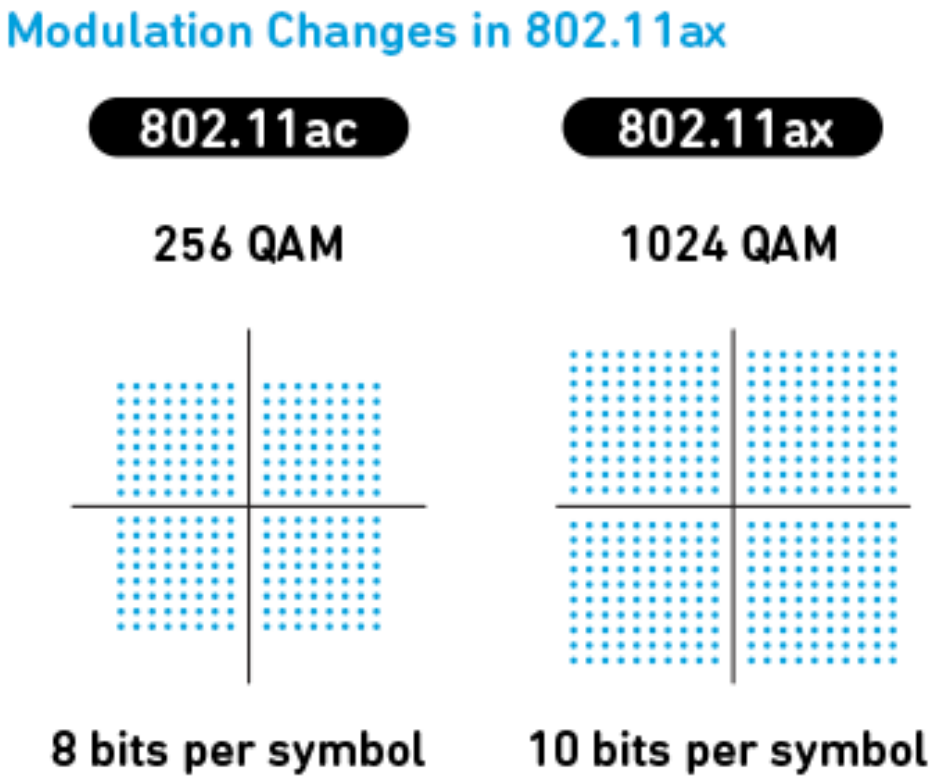


Wifi Uses QAM at higher and higher densities

- Each measurement becomes a symbol
-



More advance wifi uses 256 and 1024 QAM



Use trig functions to analyze these

- Find their magnitude and find their phase in I/Q data…that will tell you the value.