```
UNTIL SHIP:ALTITUDE > 30000 {
    SET targetBearing to 90.
    SET targetPitch to LERP(90, 45, CLAMP(SHIP:ALTITUDE, 0, 25000)/25000).
    SET targetRoll to LERP(-90, 0, CLAMP(SHIP:ALTITUDE, 0, 2000)/2000).
    LOCK STEERING TO HEADING(targetBearing, targetPitch, targetRoll).
}
```

Like all orbital rockets, we want to achieve a horizontal velocity of ~2200m/s to stay in orbit around Kerbin. However, we also need to return the boosters to the launch site. Because the boosters have to do boostback burns to arrest their horizontal velocity and move their impact location from far in the ocean to back at the same centre, we want to have as little horizontal velocity as is reasonable when we separate the boosters. That's why we use a lofted trajectory to get to orbit. For reference, as Matt Lowne explains in this video that taught me how to get into orbit in KSP years ago, on a standard ascent profile you aim for 45 degrees off vertical when you're at 10km. We aim for 45 degrees off vertical at 25km. This decreases the fuel required for the boostback burn and hence increases payload performance.

How Staging Works

The upper stage and boosters are dorment until their respective staging events. They wake up when particular mass values are reached. The core for this is shown below. Note that the UNTIL loop is just the opposite of a while loop. It runs the code until the condition is true.

Center Core Script:

```
IF (SHIP:MASS < 35.5 and SHIP:STAGENUM >= initialStageNum - 1) {
        LOCK STEERING TO srfPrograde.
        WAIT 2.5.
        STAGE.
        LOCK STEERING TO HEADING(targetBearing, targetPitch, -90).
        WAIT 0.5.
    }
```

Booster Script:

```
PRINT "WAITING FOR STAGING" at (0, 0).

UNTIL SHIP:MASS < 35.5 {}
WAIT 2.6.

Print "STAGED                 " at (0, 0).
```

```
LOCK THROTTLE TO 0.
```

The messaging system between separate scripts in kOS is slightly complicated, so I decided it was easier for each script to watch for staging events on their own instead of a central script telling all the others to wake up.

I tried to make each script detect staging events on their own, but this failed. The number of stages on a vehicle in kOS can be read with “SHIP:STAGENUM”. However, the number of stages on a vehicle only updates when the player is currently looking at a vehicle. This is a limitation of kOS as it has to work around KSP, which is designed for a single craft to be controlled at a time. This is why I had to use the “SHIP:MASS” value to detect staging events. The center core and boosters all detect the criteria for staging (Criteria: total rocket mass < 35.5t) individually and then move onto the next mission state themselves.

Batshit Crazy Boostback Burn Startup

The atmosphere of Kerbin is not balanced as well at God balanced Earth’s atmosphere. When we stage the boosters, we are still only 28km above the surface of Kerbin. This means aerodynamic affects play a major role in the control of the boosters. Unlike Falcon 9’s comparatively calm reorientation for boostback where it is above the atmosphere, our boosters are still in the atmosphere and have to fight against it. We can’t rely purely on RCS to reorient the boosters and

atmospheric forces are too strong. So, we fire the engines once we are within 90 degree of the boostback burn direction vector. This leads to a batshit crazy looking separation and boostback startup as all the boosters are point in different directions and getting wildly flung around by the atmosphere. Watch the video at the top at the 1:51 mark to see what I mean.

Also, in testing I had the force of the decouple between the center core and the boosters set too high. Because the decoupler is at the bottom of the boosters for aesthetic reasons, this meant the top of the boosters would be pushed into the center core as the bottom was pushed away. Soyuz Style - This image really gets the point across. The solution was the decrease the force the decouplers exerted and to add small solid separation motors. Note that the force vector of the separation motors runs through the center of mass of the boosters to ensure no torque.

"Realistic" Upper Stage Burn

Once the center core propels the stack to an apogee of 77 km (exterimentally found to be the optimal value to have enough fuel to land), the second stage separates and continues onto orbit.

When you're playing KSP yourself, the common way to get the second stage into orbit is to create a maneuver node at the apoapsis and burn prograde until you have your desired periapsis (likely above 70km, the barrier of the atmosphere). However, this usually leads to a coast phase between separation of the first stage and startup of the second stage engine. This is not very realistic, so I wanted to avoid it.

Instead, I start the second stage engine immediately at a throttle of 50%. The engine keeps running until we make orbit. This makes a far more realistic looking ascent profile.

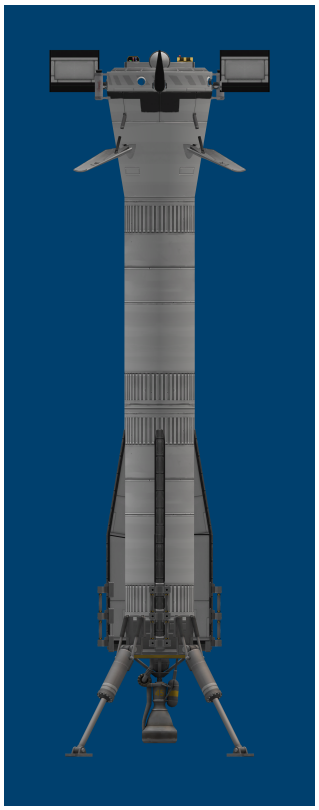
An issue emerges when the engine is ran at a throttle greater than ~50%. Because the second stage starts it's burn so early, it is still increasing in altitude and is far away from it's peak on it's parabolic flight, the apoapsis. This means that as we coast to apoapsis, the thrust from the engine is making that point higher and higher and further away in time. If this flight profile is followed, we keep burning to increase our apoapsis, but never efficiently enough to increase our periapsis so we reach a stable orbit (Oberth effect ftw).

To solve this, I have the second stage engine throttle down to 50% and control it's pitch to ensure the apoapsis doesn't run away from us. When we point down, the apoapsis decreases in height and comes closer to us in time. The opposite is true when we point up. So, the upper stage script is constantly adjusting the pitch to keep the apoapsis at a constant time away from us. It estimates the remaining time in the burn, and tries to make it so that we reach the apoapsis right when the burn ends. Once you understand the formulas, the math for this is very simple to implement.

```
CLEARSCREEN.
Print "BURNING TO ORBIT" at (0, 0).

LOCK STEERING to HEADING(targetBearing, 10, 0).
LOCK THROTTLE to 0.5.

UNTIL SHIP:ALTITUDE > 65000 { PRINT "WAITING FOR 65KM ALTITUDE FOR FINE CONTROL" at (0,

// We aim to burn until 10 seconds after the apoapsis to get into orbit, so calculate th
UNTIL ORBIT:PERIAPSIS > 75000 {
    SET shipdV to 9.81 * isp * ln(SHIP:MASS / SHIP:DRYMASS).
    SET remainingdVToLEO to leoVel - SHIP:VELOCITY:ORBIT:MAG.

    SET finalWetMass to SHIP:DRYMASS * 2.71828^(remainingdVToLEO / (9.81 * isp)).
    SET burnMass to SHIP:MASS - finalWetMass.
    SET burnTime to burnMass / massFlowRate.

    SET targetApTime to 10. // We want to be 10 seconds away from apoapsis forever

    SET timeToAp to ETA:APOAPSIS.

    SET targetPitch to CLAMP((targetApTime - timeToAp)*0.5, -30, 30).

    LOCK STEERING to HEADING(targetBearing, targetPitch, 0).
```

```
    PrintValue("Engine ISP", isp, 2).
    PrintValue("Engine Mass Flow Rate (t)", massFlowRate, 3).
    // The rest of the print statements are not included
}
```

Booster Aerodynamic Control Issues



When I was first testing the landing script, I only needed to test descent and not worry about ascent. This meant I only had to optimize for aerodynamics on descent. If you don't understand how to optimize a booster for descent I'm not gonna be able to explain it without being with you in person - If you'd like this please send me a dm. The basic principles are that you want high drag and for your center of mass to be below your center of drag. However, on ascent you want the opposite. The Falcon 9 solves this with deployable grid fins, but we don't have this in KSP 1 (RIP KSP 2).

Because your rocket burns fuel as it ascends, the center of mass shifts. We can set the rocket to first use fuel from the upper fuel tanks so that the COM shifts downwards. In a scenerio where our COL (Center of Lift) stays constant (Eg. New Glenn without deployable aero surfaces), at the begining of our first stage burn we could have COM below COL to ensure we fly point end up, and at the end when we're flying back, we could have COM above COL to ensure we fly engines first. In my first iteration of the vehicle this was what I aimed for with New Glenn / Superheavy

chines at the bottom of the booster and non-deployable fins at the top, Howeverm the center of mass didn't shift enough to allow for stable ascent and stable descent.

The solution (as SpaceX learned on Falcon 9!) is deployable control surfaces. Instead of having static fins, I added joints from the Breaking Ground DLC to the fins so they could be stowed on ascent and deployed for descent. This solves to COL problem as we can artificially shift it when we descent by deploying fins at the top of the rocket.

Another issue was the drag on descent. When I was testing descent in the previous blog post, I created Falcon 9 Grass Hopper looking landing legs. These had a big base for supporting the legs that provided a tremendous amount of drag on descent. The completed rocket did not have this base and hence had far lower drag when on final descent. This is the difference between a terminal velocity of ~200m/s and ~500m/s. As you might imagine, this is an extreme difference in the fuel required to land. The solution is simple, air breaks. They look slightly ugly and are slightly unrealistic, but they work.

Another possible solution to this problem is to aggressively pitch the booster side to side on descent. This way, you can greater increase average drag force on the booster. Imagine this like doing S-curves while plummeting down to the surface. Although this would be a great mix between and extremely elegent simplification of the problem and a batshit insane looking descent, air breaks were far simpler to implement. Just add them to the craft and set them to an action group, simple.

Binary Search & Lack of Landing Pads

The Kerbal Konstructs mod (I think it's this one) I'm using to add the extra landing pads at the Kerbal Space Centre only adds three landing pads, presumably chosen for the required amount for a Falcon Heavy RTLS mission. I have 5 boosters so why don't we just they to get them to land at the same landing site in a pattern? Very beautiful, except I was slightly off when setting the coordinates for the landing position and they where perfectly aligned. Oh well.

The previous version of the code used the Trajectories mod to find the impact location of the boosters. However, the Trajectories mod is not designed to be used with multiple craft in kOS. This emerges as a phenomenon where the scripts do not get any impact position information if I am not actively focused on the craft. Worse yet, at state changes in the code, the scripts crash. So, a solution for finding the impact location had to be found that didn't use the trajectories mod.

Luckily, I had already created my own function for this that uses binary search to find the coordinates and time the booster would reach a particular altitude. KSP gives you information on your orbit at any given time, so with some bounds (eg. +0 and +10 minutes) and binary search, you can find the time in your orbit when you'll be at a particular altitude (eg. 0 meters). With the time you can convert that to a position using the same orbit information that KSP provides you.

However, the KSP Orbit information is all given relative to the core of the planet. This means that it doesn't account for the rotation of Kerbin. The simple solution to this is finding the circumference of Kerbin (2*600km*pi) and multiplying this by your eta to get an offset. This

approach doesn't take into account aerodynamic forces (I think trajectories does this), but the error is decreases as we get closer to landing so the boosters asymptotically approach the correct impact location (the landing site).

Binary Search Function found in KOS-Scripts/HelperFunctions.ks:

```
// Return the lat/long of the position in the future on the current orbit at a given alt
// Ie. find the geolocation when we're at x meters above the surface in range y seconds
// SET impactGeoPos to GetLatLngAtAltitude(0, SHIP:OBT:ETA:PERIAPSIS, 10).
function GetLatLngAtAltitude {
    local parameter targetAltitude. // Meters
    local parameter timeRange. // Seconds
    local parameter altitudePrecision. // Allowable meters from given altitude to be cor

    // Replace 'SET' with 'Local'
    // Lower bound is present, upper bound is future
    local lowerBound to TIME:seconds.
    local upperBound to TIME:seconds + timeRange.
    local midTime to 0.

    // Binary Search
    for x in range(0, 35) {
        SET midTime to (lowerBound + upperBound) / 2.
        local midAltitude to body:altitudeof(positionat(SHIP, midTime)).

        if midAltitude < targetAltitude {
            SET upperBound to midTime.
        } else {
            SET lowerBound to midTime.
        }

        // If error less than precision
        if ABS(ABS(midAltitude) - targetAltitude) < altitudePrecision { BREAK. }
    }

    local geopos to BODY:GEOPOSITIONOF(positionat(SHIP, midTime)).
    // Longitude rotation of planet during coast to altitude ((360 degrees * seconds unt
    local rotationAdjustment to (360*(midTime-TIME:seconds)/BODY:rotationperiod) * cos(g

    return latlng(geopos:lat, geopos:lng - rotationAdjustment).
}
```

A Slightly More Unified Solution to Landing

In my previous blog post on kOS scripts, I wrote this:
"I imagine the solution is to track the estimated net displacement in landing position during the Suicide Burn. With the estimated time to touchdown, current pitch, and current horizontal velocity you could approximate the net displacement. Add this to the target landing location and it should be a much more accurate landing."

This quote is in reference to suboptimal solution to final propulsive descent that I came up with in my first iteration. An issue arises when your final landing burn is not perfectly vertical. Because there is a horizontal component to your thrust, your impact position shifts closer to you. If you were originally targetting your landing site before the landing burn, this horizontal component means you'll now land far short of it. My solution at the time was to adjust the target landing position by a constant so that we would initially overshoot, but end up landing right on target.

The addition of a constant was a very imprecise way to solve this issue. If we came it in different angles we would be off target (imagine, in the limit, a perfectly vertical or perfectly horizontal trajectory). So, the constant was just the eye-balled optimal value for an expected trajectory.

A better solution is to calculate an estimate of your net horizontal displacement during the landing burn. Then, you can add this value to your target landing location and end up far closer to the target in a wider range of trajectories.

```
function GetSuicideBurnNetDisplacementEstimate {
    local pitchRelativeToDown to vang(ship:facing:forevector, up:forevector). // Eg. up

    // Iterate over every second until impact and linearly estimate the angle relative t
    // With this value, calculate the difference in horizontal velocity, and add to net
    local localSuicideBurnLength to GetSuicideBurnLength().
    local t to localSuicideBurnLength.
    local netDisplacement to 0.
    UNTIL (t < 0) {
        local angle to lerp(0, pitchRelativeToDown, t / localSuicideBurnLength).
        local xVel to SIN(angle) * ((SHIP:AVAILABLETHRUST*EstThrottleInSuicideBurn) / SH
        SET netDisplacement to netDisplacement + xVel.
        SET t to t - 1.
    }

    return netDisplacement.
}
```

Above you can see the code for this estimated displacement function.

It works by getting your current pitch relative to vertical. It assumes you decrease your pitch linearly as you come in for landing. With this assumption, it steps through second by second to calculate your horizontal velocity at each step and takes the sum of these velocities to get your net displacement. This is a slightly shitty implementation and I didn't know the meaning of the word integral when I wrote this, but it works (mostly).

The "mostly works" part is adjusted for by another constant! Use this one easy trick to solve all your problems! Just add another terms! You can see the EstThrottleInSuicideBurn variable in the function above. We need to know our average throttle during the landing burn to get an accurate displacement result, and that's what this value represents. I set it to 1.5 (150%) in the config file for the boosters (KOS-Scripts/Dzhanibekov/DzhanibekovBoosterEast.ks). This is not quite a real value because we of course never expect more than 100% throttle, but I experimentally found it to be the proper value. A common theme in this project is experimentally finding proper parameters mostly because the experiments here are watching rockets land!

The config file for the boosters contains slightly more information than just the estimated throttle during the landing burn. It also contains the pitch multiplier that specifies how much the boosters should pitch to minimize error between intended landing site and impact position, craft height, and other variables. This "config" file isn't quite a config file because kOS doesn't allow this (shitty language). So, it also has the code to handle staging and running the landing script after the boosters have separated from the core.

I've covered the most important parts of this project, If you have questions, DM me.

MERRY CHRISTMAS!!! 🎄🎄🎄

Subscribe

CKalitin
x.com/CKalitin

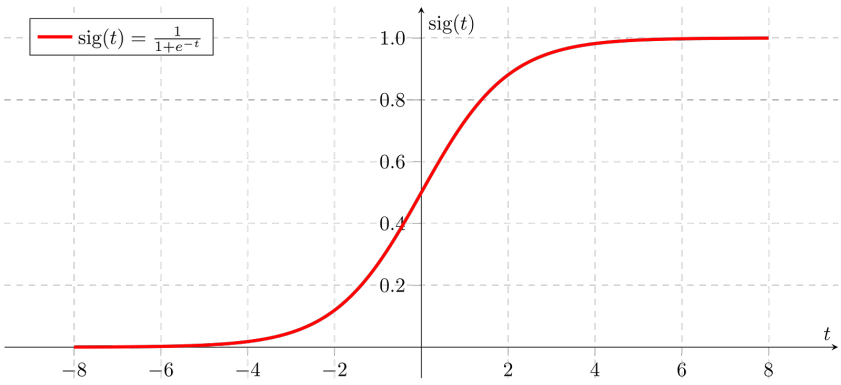
Geohot made a blog too. You should be working on hardware

S-Curves Allow You to Predict the Future

Nov 19, 2024 • Christopher Kalitin

I've decided I'm going to be an Econophysicist.

Historic Proof for S Curves

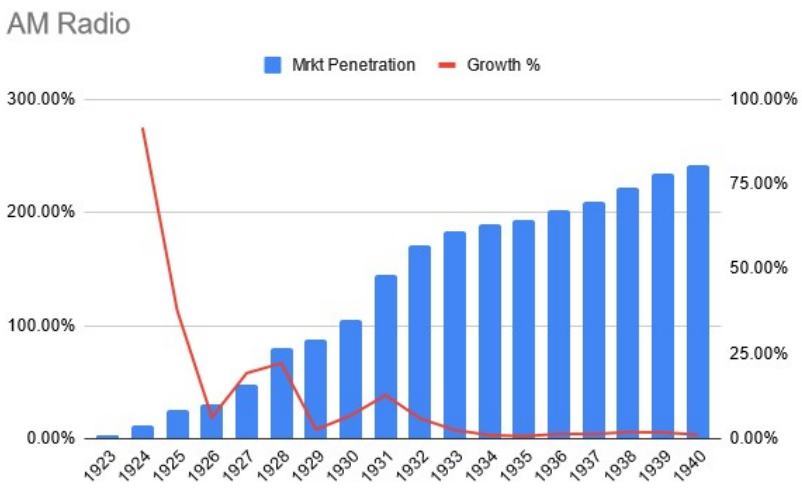


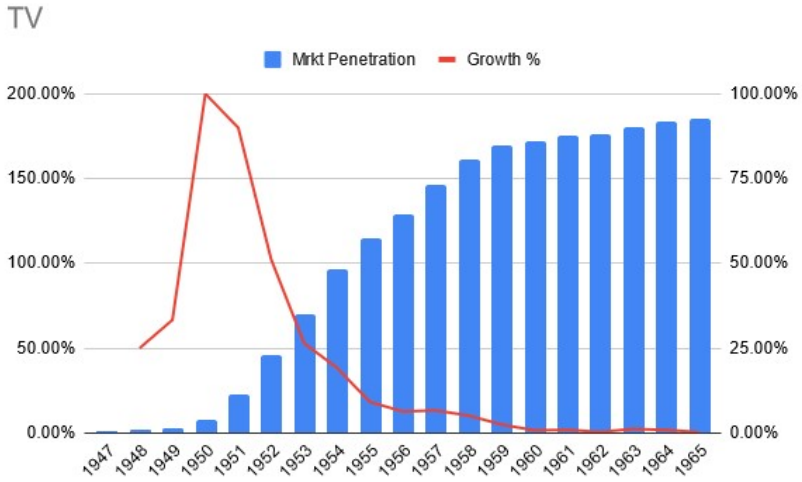
An S curve is a function that describes the shape of the market share vs. time graphs of almost all technologies. These S-curves are characterized by an extremely slow start, where they are arbitrarily close to zero; an exponential growth phase; then followed by a levelling off where they asymptotically approach 100% market share.

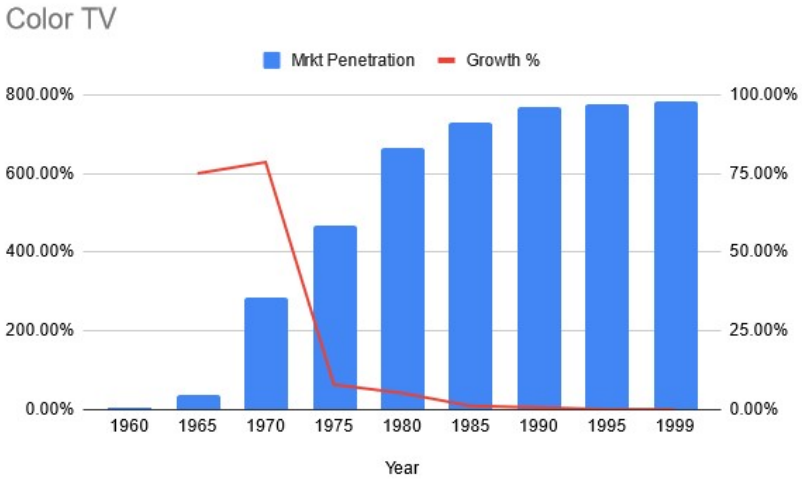
These S curves give you the ability to predict the future with a high level of certainty. Psychohistory brought to reality (Read Foundation). The reason you can use this method to predict the future with such high certainty is that the growth of almost all technologies that we have ever invented have followed the same pattern: the S curve.

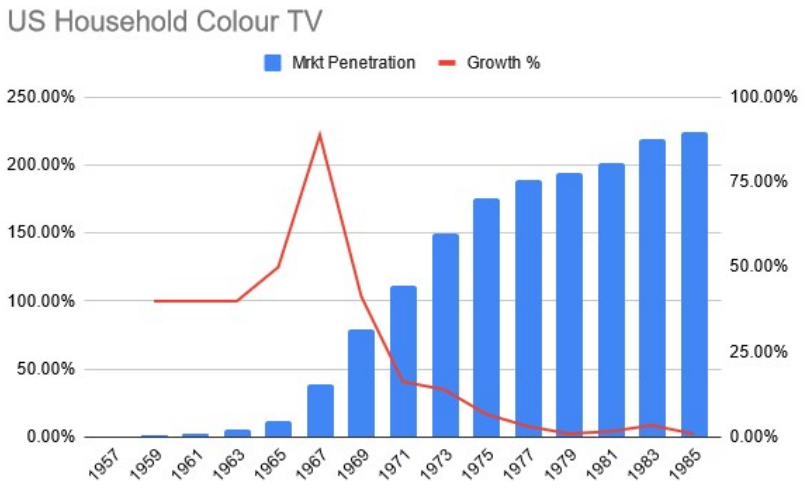
At the beginning of this year I began work on my longest blog post yet, The Transition to EV Robotaxis (FUTURE CHIRS! UPDATE THIS LINK WHEN ITS DONE). In that post I extrapolated historic data on EV adoption by fitting an S curve to the data. I found that 50% of cars sold in 2027 will be EVs and 90% in 2030. To ensure fitting an S curve was an appropriate method to make such predictions, I aggregated the market share data of several technologies over the past century to confirm whether they followed the same growth pattern. All of these charts are shown

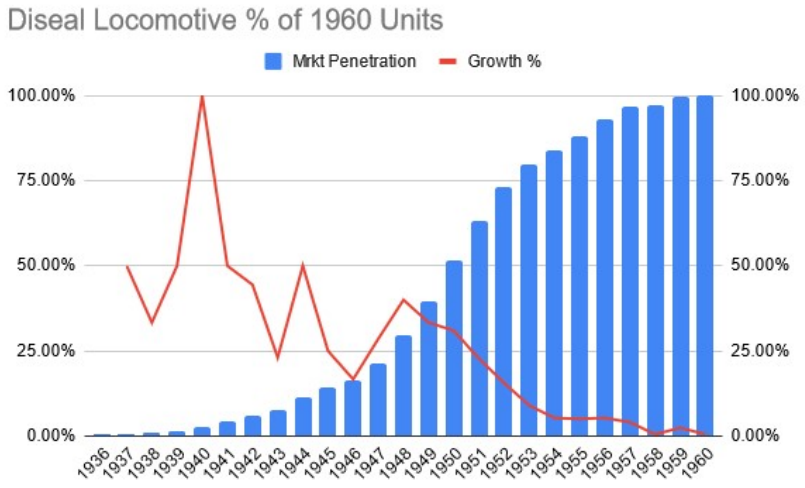
below.

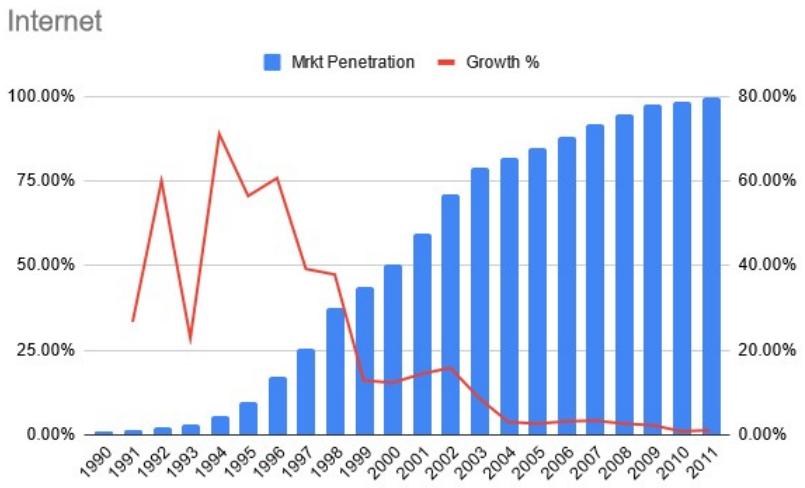

AM Radio
Mrkt Penetration
Growth %
300.00%
200.00%
100.00%
0.00%
100.00%
75.00%
50.00%
25.00%
0.00%
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940

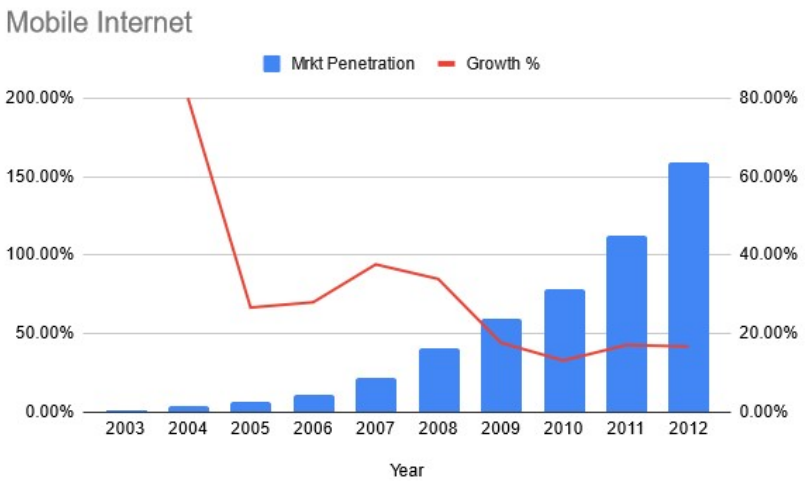

TV
Mrkt Penetration
Growth %
200.00%
150.00%
100.00%
50.00%
0.00%
100.00%
75.00%
50.00%
25.00%
0.00%
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965

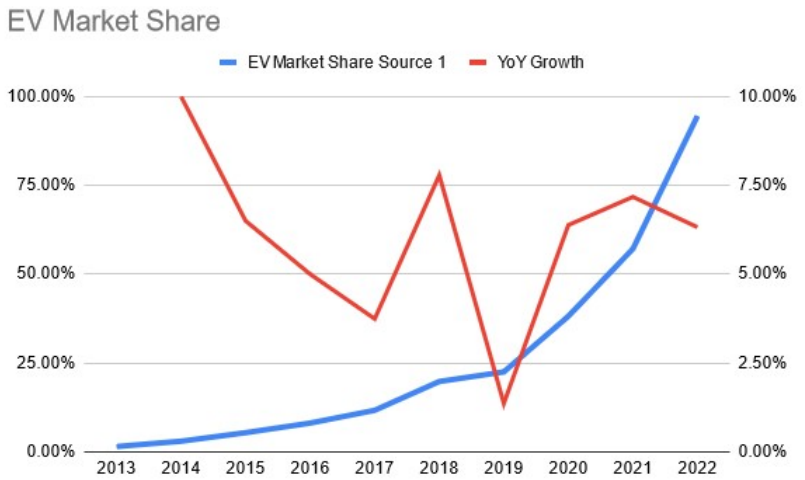

Color TV
Mrkt Penetration
Growth %
800.00%
600.00%
400.00%
200.00%
0.00%
100.00%
75.00%
50.00%
25.00%
0.00%
1960
1965
1970
1975
1980
1985
1990
1995
1999
Year


US Household Colour TV
Mrkt Penetration
Growth %
250.00%
200.00%
150.00%
100.00%
50.00%
0.00%
100.00%
75.00%
50.00%
25.00%
0.00%
1957
1959
1961
1963
1965
1967
1969
1971
1973
1975
1977
1979
1981
1983
1985


Diseal Locomotive % of 1960 Units
Mrkt Penetration
Growth %
100.00%
75.00%
50.00%
25.00%
0.00%
100.00%
75.00%
50.00%
25.00%
0.00%
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960


Internet
Mrkt Penetration
Growth %
100.00%
75.00%
50.00%
25.00%
0.00%
80.00%
60.00%
40.00%
20.00%
0.00%
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011





Source of the charts

As you can see, all of these charts follow the same pattern. It's important to note that I didn't cherry pick any of these, I just searched for growth curves of early technologies and pixel counted to get the data you see above. Whether this introduces bias is up to you, but if you find any growth curves that don't follow the S curve, please immediately send me a message. Such information would be extremely important - my preferred method of contact is carrier pigeon.

Learning Rate Explains it All

Assuming you aren't frantically searching for a carrier pigeon to correct me right now, we can assume that all that historic data proves that all new technologies follow the S curve. If you have the mentality of a school student who only wants to answer the questions on the test, you can be satisfied with the information above and apply it to all new technologies to your heart's content. However, if you're destined for great things, you must understand why S curves happen from first principles.

As Casey Handmer (Highest information density speaker alive) explains in this clip and elsewhere,

the fundamental reason that technologies follow S curves is the exponential growth that occurs when learning rate is allowed to compound.

Learning rate is the percentage decrease in cost that a technology experiences as a result of a cumulative doubling in production. For example, batteries currently have learning rate of about 20%. Batteries double in production about every 2 years and fall in cost by about 20%.

The framework of applying learning rate is most useful for new exponentially growing technologies where doubling time is low. For example, EVs have a doubling time of about 2-3 years. If we extrapolate this out a few years we find that we will undergo a massive paradigm shift where EVs become the most common vehicle type on roads. Electrical transformers can also be described by a learning rate, but because they are such a mature technology the learning rate is only 4% and doubling time is very long.

Like Handmer explains, exponential decrease in cost (described by learning rate) is due to the reinforcing cycle of increased demand which leads to increased production which leads to decreased cost which leads to increased demand. As a product increases in volume, the revenues from the product increase, which allows for further investment into R&D and production, which further decreases cost and increases the desirability of the product.

The most famous application of this framework is Moore's Law, which states that the number of transistors in an integrated circuit doubles about every two years. Moore discovered this with only about 4 points on the graph of transistor count vs. time. Because learning rate appears to be a fundamental law of the universe, he was able to predict the future of integrated circuits far into the future with a high level of certainty.

An even more remarkable fact about Moore's Law is that it applies to computers that existed well before Moore stated his law. When we look at the electromechanical, relay, and vacuum tube based computers that came before integrated circuits we find that the trend of exponential growth was present long before Moore's Law was discovered. Learning rate even works in the backwards direction before we ever thought about it!

The Life Cycle of a Technology

Precursor stage: dreaming, eg. Da Vinci drawing aeroplanes
Invention stage: the birth of a technology
Development stage: often more crucial than invention, many improvements that may be more important than the original invention
Maturity stage: interwoven into the fabric of life
Stage of the false pretenders: upstart threatens to replace the technology, but is missing some element, fails to surpass the original invention, short victory for the original technology
Final stage: new technology renders the original technology into a stage of obsolescence
Actual final stage: The complete end of the original invention, eg. horse and buggy, records

The Life Cycle of a Technology

We can identify seven distinct stages in the life cycle of a technology.

1. During the precursor stage, the prerequisites of a technology exist, and dreamers may contemplate these elements coming together. We do not, however, regard dreaming to be the same as inventing, even if the dreams are written down. Leonardo da Vinci drew convincing pictures of airplanes and automobiles, but he is not considered to have invented either.
2. The next stage, one highly celebrated in our culture, is invention, a very brief stage, similar in some respects to the process of birth after an extended period of labor. Here the inventor blends curiosity, scientific skills, determination, and usually of showmanship to combine methods in a new way and brings a new technology to life.
3. The next stage is development, during which the invention is protected and supported by doting guardians (who may include the original inventor). Often this stage is more crucial than invention and may involve additional creation that can have greater significance than the invention itself. Many tinkerers had constructed finely hand-tuned horseless carriages, but it was Henry Ford's innovation of mass production that enabled the automobile to take root and flourish.
4. The fourth stage is maturity. Although continuing to evolve, the technology now has a life of its own and has become an established part of the community. It may become to intertwined in the fabric of life that it appears to many observers that it will last forever. This creates an interesting drama when the next stage arrives, which I call the stage of the false pretenders.
5. Here an upstart threatens to eclipse the older technology. Its enthusiasts prematurely predict victory. While providing some distinct benefits, the newer technology is found on reflection to be lacking some key element of functionality or quality. When it indeed fails to dislodge the established order, the technology conservatives take this as evidence that the original approach will indeed live forever.
6. This is usually a short-lived victory for the aging technology. Shortly thereafter, another new technology typically does succeed in rendering the original technology to the stage of obsolescence. In this part of the life cycle, the technology lives out its senior years in gradual decline, its original purpose and functionality now subsumed by a more spry competitor.
7. In this stage, which may comprise 5 to 10 percent of a technology's life cycle, it finally yields to antiquity (as did the horse and buggy) the harpsichord, the vinyl record, and the manual typewriter).

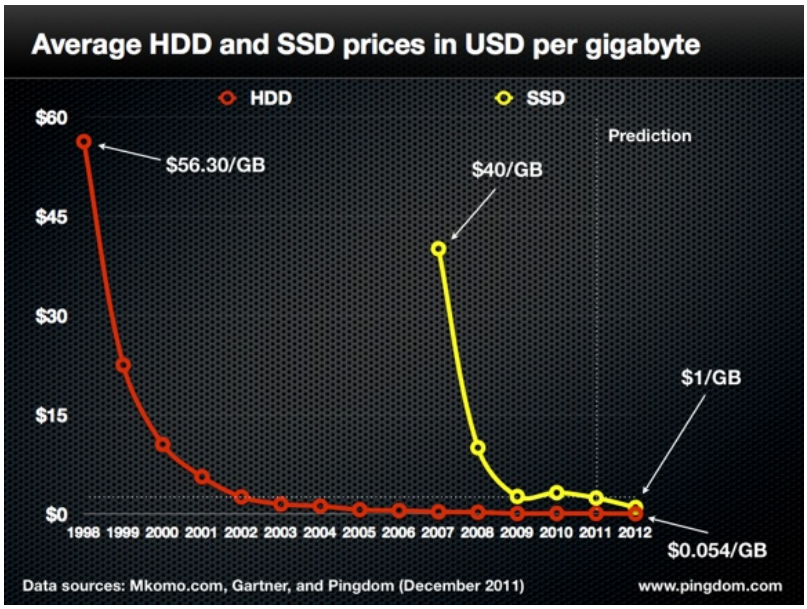
The Singularity Is Near, Chapter 2, page 105 on Internet Archive

Above you can see the seven stages that Ray Kurzweil laid out for the life cycle of a technology. The Singularity is Near was the first place I was exposed to these ideas presented in such a concrete manner. My first blog post was about insights I gained from this book, you may also want to read it.

A technology goes through five majors stages that can be described by its S-curve. These stages are:

1. Early R&D
2. Initial Commercial Appeal
3. Obvious exponential growth
4. Market Saturation
5. Stagnation

1. Early R&D



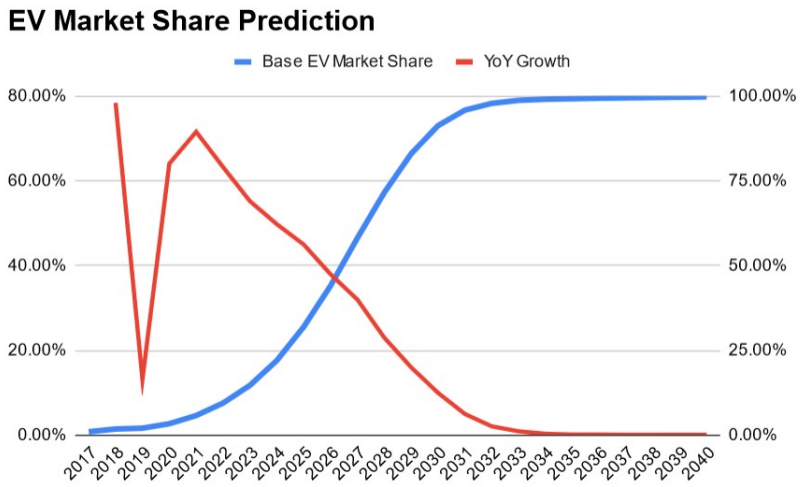
The early growth of a technology is characterized by exponential improvements that do not make much of an impact for its total market share. The technology may be exponentially improving, but it is still not good enough for anyone to use it at scale. For example, Solid State Memory (SSDs) were conceived of in 1978 and were first released as a product in 1991. However, it took until the late 2000s for costs to drop low enough that they were considered a reasonable alternative to hard drives.

2. Initial Commercial Appeal

After this initial R&D phase, there is a smooth transition to a phase in which the technology begins to be adopted. It has exponentially declined in cost enough that it has its own niche. An example here is golf carts, they are a relatively small and niche market compared to all other

vehicles, but it just so happens that the electric powertrain is more optimal for the user experience than an internal combustion engine. This means that electric vehicle technology can be adopted to golf carts, which in the grand scheme of things isn't a very large market, but is just a step along the path to market domination.

3. Obvious exponential growth



Another gradual and smooth transition occurs from the second phase into the third, exponential growth. The technology has declined in cost and improved in capability enough that it is now the best option for a large portion of the market and it begins to take significant marketshare. We are living through this today with EVs. Pure electric vehicles currently have around 25% market share worldwide and the current growth rate is around 50%. Extrapolating this out a few years, we find that EVs will be the most common vehicle type on the road.

Because the exponential growth in this era is driven by economic factors (decreasing cost and increasing capability), the growth continues even through recessions and depressions. For example, look at the charts on AM radio during the Great Depression and EV adoption during Covid.

4. Market Saturation

No product can grow exponentially forever as we live in a finite universe. Once the market for a technology has been saturated, growth rates begin to decline. Now, unlike phase 2 where the technology only had its own niche, the technology makes everything else a niche. Horses were dominant before the internal combustion engine, but now are mainly used for niche recreational purposes, meanwhile the internal combustion engine powers the majority of vehicles.

It's important to note that growth rates linearly decline as a technology increases in maturity. This is again due to the fact that no technology can grow forever. You can see this in any of the charts above or my prediction for EV market share. Also, given a linearly declining growth rate, the magic of derivatives means that we get a nice smooth exponential looking curve. Next time you

hear that the growth rate of EVs is declining, remember that this is a byproduct of market domination.

5. Stagnation
In the final stage of a technology, it asymptotically approaches 100% market share and we await another disruptive technology to emerge that displaces the current paradigm. We currently see this in internal combustion engine powered automobiles. We have had these products for over a century and there are very marginal improvements in their efficiency and cost. Most of the improvement we've seen in vehicles recently has been in comfort and other user-focused features.

The End of Learning Rate and the Death of a Technology

In the final stage of the life cycle of a technology, we see the end of learning rate and the slow death of the technology. As we learn more about how to harness a particular technology, we near the limits of the technology. This means that the once constant variable of learning rate begins to decline and the technology stagnates. Once you're in this stage, applying learning rate is no longer a very useful exercise to understand the future of the technology because the technology has very little future apart from what it already is. This is why learning rate is best applied in stages 2-4.

This stagnation is the time period in which we can hope to see a better technology arise that will supercede the current technology. In some cases, waiting for the emergence of this new technology can be a very long process that has extremely major consequences for the future of humanity. We are unable to predict when this new technology will emerge if it is still early in stage one - unless you have a perfect mental model of all similar technologies that are currently being researched and are able to predict their success.

Waiting on these new technologies can have major implications because of the impacts of the current technology. For example, Climate Change is caused by the previous/current paradigm of Hydrocarbon-based fuels. This technology was a huge win for humanity hundreds of years ago and allowed us to build the modern world and feed billions of people living better lives than they ever have. However, this technology has now begun to stagnate and we are seeing the negative impacts of it (citation needed). The solution to Climate Change is the next energy technology which will elevate humanity into a new era of unprecedented prosperity. Solar panels are batteries are declining in cost exponentially and will replace the vast majority of previous energy technologies in the coming decades. Modelling this out is left as an exercise to the reader.

The inherent hope of this perspective is that as long as humanity continues to develop technology, all of our problems will be solved and we will continually prosper to greater and greater levels. However, this isn't automatic! We must continue to work extremely hard to