

TEMPERATURE MODELING

Version 1.0, October 7, 2007

By Marcin Pohl

So why would we want to model temperature? Why does it need to be modeled at all? Isn't the temperature of the air measured by a sensor called the Intake Air Temperature? This paper concentrates on one aspect of controlling the ever-important Air Fuel Ratio, demonstrating the importance of correctly accounting for the air charge temperature, one of air mass' components.

Problem statement

We've all seen the effects of temperature swings on the AFR: the car runs lean at night, but rich during the day; there's a lean condition every time you take off a red light; there's a perpetual lean or rich condition during seasonal changes.

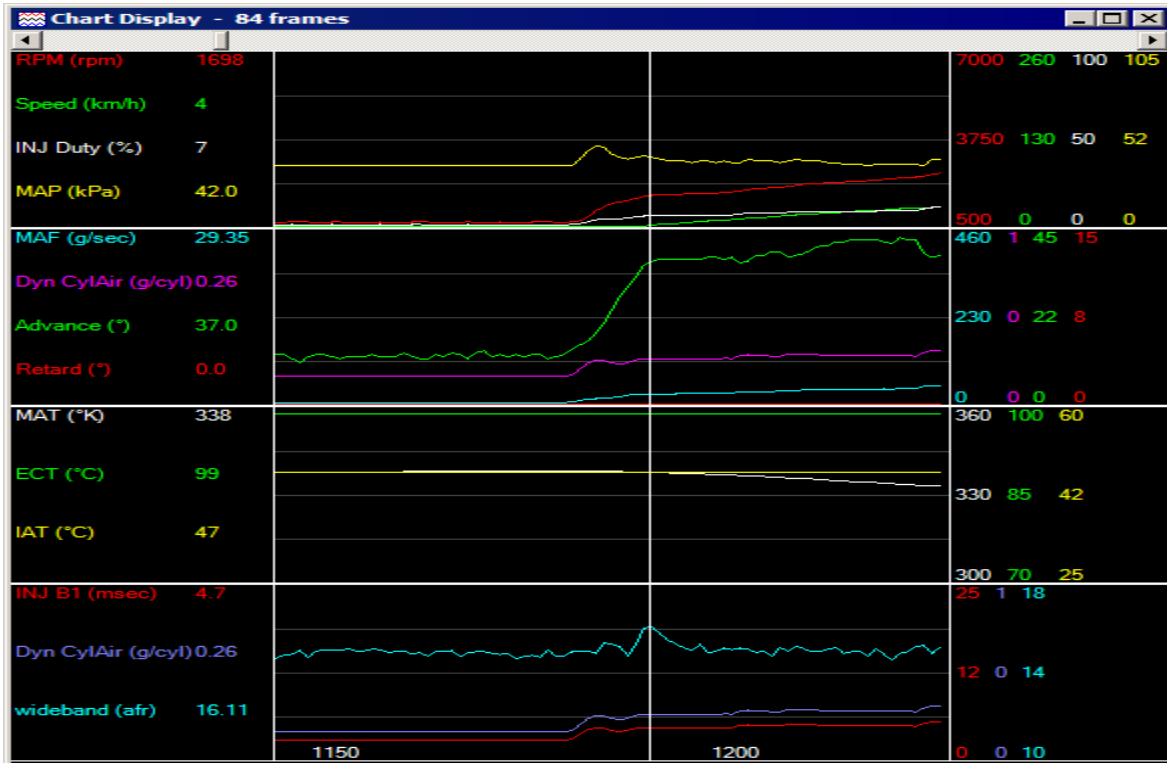


Figure 1--Lean condition on takeoff

All these problems stem from the same issue: the air temperature is either over- or under-reported to the computer. If the temperature reported to the computer is lower than the real

temperature, the computer expects more air, and thus commands more fuel than really needed. The converse is also true: if the sensors report higher temperature than they are in reality, the computer will inject less fuel than necessary, causing a lean condition. Thus, reporting the most precise temperature possible is crucial in obtaining a tune that holds up against all environmental changes.

Current solutions

Most of the times, the imperfections in temperature reporting were just treated as a fact of life. Seasons affect the Speed Density tunes, and require an occasional adjustment. Luckily, some people started observing what causes the various temperature-based problems. Also, my 'How Speed Density works' paper forced me to derive the air mass model, greatly enhancing my understanding as far as what variables affect the air mass and consequently, the fueling.

The formula for cylinder air mass (CAM) involves pressure and temperature ($CAM = GMVE * \frac{MAP}{TEMP}$) so theoretically it should be able to account for most environmental changes. The only notable variable that's skipped in the model is the humidity of air, but since its impact is less than 1%, it should not be too much of a problem. The model itself is sound, so it must be the data we feed into it that's the problem. MAP sensor is pretty fast and usually within a 1 or 2 kPa. Then there's the temperature—we don't know really how quickly the sensor adjusts or how precise it is, as we don't have an external way of comparing it. However, just by looking at the equation we can see that a 30°C error in reported temperature will cause 10% swing in air mass estimation, and thus, the fueling error. This means that to keep the AFR swings <1%, our temperature estimation must be within 3°C of the real temperature. The goal of this temperature model is to be within 3°C of the scanned manifold temperature at all times.

The GTO problems

The GTO owners quickly noticed that their cars are very quick to have timing reduced due to high reported temperatures. Some of them thought that it's just an overly safe tuning from the factory, and disabled the IAT based timing reduction in the tune altogether. Sadly enough, it turns out that at higher temperatures the car will knock and the timing will be lowered regardless of the tuning. So the problem turned out to be mechanical and not tune related. However, during the experiments trying to locate the source of pulled timing, people have discovered that the IAT sensor did not change its reported temperature quickly enough, even when given a healthy jolt of new, cold airflow. Two thoughts followed: the sensor itself is placed too close to other parts retaining heat, and thus experiencing heat soak; the sensor itself does not react as quickly as it should. Both of these can be

solved simultaneously with replacing the sensor with a quicker reacting one, and placing it somewhere less prone to heat soak ([IAT probe replacement how-to](#)). Most people placed the IAT sensor in the air box as it was reporting the fresh air just as it came in. Unfortunately, the temperature of the air in the air box is going to be much colder than the same air once it gets back to the in the manifold, and thus causing unrealistically high cylinder air mass numbers.

The new approach

Historical background

The block, the heads, the exhaust headers are all huge emitters of heat, and the intake sitting on top if all the hot parts is like a turkey in the oven. I guess this would make the air inside of the intake much like the stuffing, cold at first, but ultimately still subject to the surrounding heat, regardless of how many thermal layers are around it. The air in the intake is always moving, and there's no time for the air to heat up. However, the heat transfer is proportional to both time the cold object is exposed to the hot environment, as well as the differential between the two temperatures. With the air charge usually being about 60-90F and the engine bay temperatures being closer to 200F, some of the heat transfer will occur, despite the short of the time of exposure.

All this stuff been thought up by Sir Isaac Newton some 400 years ago, and it's known as the Newton Law of Cooling. He stated that:

$$\frac{dTEMP(time)}{d(time)} = -k * (TEMP_{object} - TEMP_{environment})$$

which means that the rate at which the temperature is changing depends on the difference between the temperature of the object and its surrounding.

If you know some basic differential equations, you can arrive at a different form of the same relationship, which allows us to predict the final temperature after a given time:

$$TEMP(time) = TEMP_{env} + (TEMP_{obj} - TEMP_{env}) * e^{-k*time}$$

k in both of these equations is a constant describing the heat exchange properties between the two objects in the scenario. K can be calculated from any known set of data, and can be treated as a constant for all other calculations between these two objects. This is for example how all the detectives figure out the time of death. To us, this form is important, as we will soon see something of very similar form. These equations should help us understand where all these dependencies come from.

The BIAS table

The BIAS table describes the proportion between ECT and IAT to calculate the new temperature. The expression is:

$$TEMP_{biased} = IAT + (ECT - IAT) * BIAS$$

We just saw a very similar equation in the previous section. We have two temperatures, one lower and one upper boundaries, with a biasing element in between the two. The traditional Newton's Cooling Law equation's has $e^{-k*time}$ as the biasing element, while in the car computer version we have a lookup table with points. However, looking at a shape of the BIAS curve for many cars (they can be quite different actually), most of them have the general properties of a $e^{-k*time}$ function—high around zero, and then dropping to asymptotically approach some value. Could it be that the BIAS component is truly an exponential function? My first concern was that it's not the time we're basing our biasing decision on, but an airflow figure. More airflow will not be as easily heated up by the piping and the intake, thus should be biased toward IAT. So what we don't have in time, we have in sheer volume that will regulate how much heating up is going to occur. My second concern was with the units; how is the result going to be a temperature if I raise a number to an exponent with a unit of grams per second? I quickly had to remind myself that exponentiation is nothing else but multiplying the same thing a given amount of times. This makes the units of the exponent irrelevant, as ultimately they're just a number of multiplications.

While this is all very strange and it's more guts than brains, it seems to work just fine. Also, on some cars, the BIAS table is dependent not only on airflow, but also on speed, so the BIAS element would probably become of $BIAS = e^{-k*Speed*Airflow}$ form. Even with all this knowledge, by itself, the BIAS equation is not enough to calculate the Manifold Temperature just like the PCM would. I tried to use it to estimate the manifold temperatures, and the results were off by a margin significant enough that I was forced to expand this research. Someone on HPT forums noticed that by playing with the BIAS filter values they were able to solve the more temporary temperature-based issues, i.e. the lean on takeoff problem. This was enough of a clue to push me toward the second important piece of the puzzle: BIAS filter table.

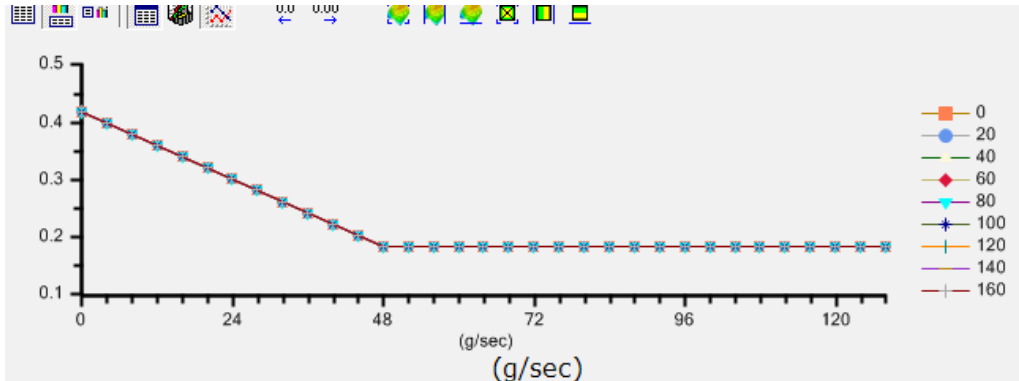


Figure 2--BIAS values on LS4

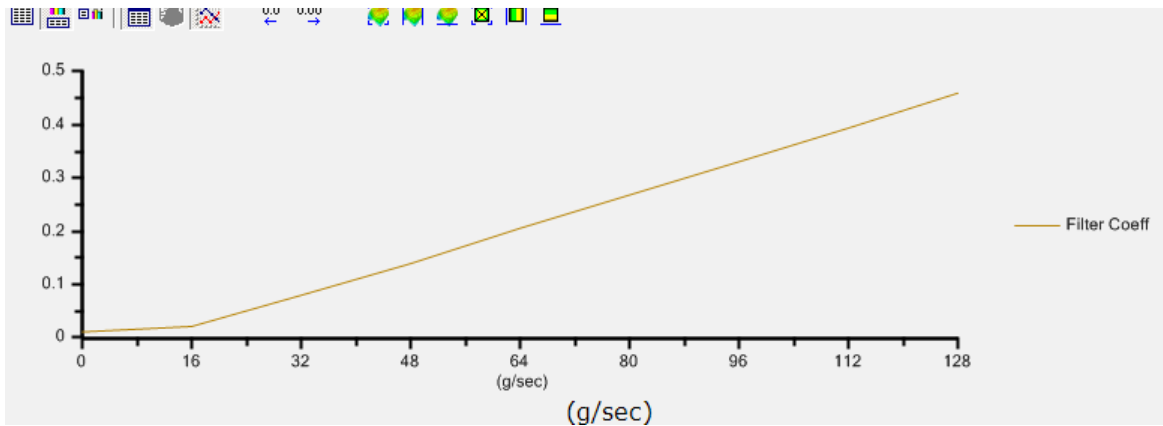


Figure 3--BIAS filter on LS4

The BIAS filter table

Apparently an air charge has its own thermal ‘momentum.’ This means that even if you have a cold object in a very hot environment there’s going to be a delay before the cold object will absorb enough heat. The higher the BIAS filter number is, the quicker the object will change temperature to its environment. Just like in the previous section, the BIAS filter table is also dependent on the amount of airflow. It also makes sense, as more airflow should force quicker changes in object’s temperatures. Effectively, the filter introduces the changes in a delayed, but must smoother fashion. Instead of sharp peaks and valleys, you will have a much smoother looking, but also much more natural and realistic looking function. Here’s the formula for the BIAS filter:

$$TEMP = TEMPold + (TEMPnew - TEMPold) * BIASfilter$$

The BIAS filter is a number in range of 0 to 1 inclusive. When it’s 0, the TEMPnew is going to get the same number as TEMPold. When it’s 1, the TEMPnew is going to completely ignore the TEMPold and jump directly to TEMPnew. If the BIAS filter is 0.5, and a 300K object is sitting in 310K environment,

then after one cycle the temperature is going to be 305K. Then after another cycle, the temp is going to be 307.5K, then 308.75K and so on until it approach the target of 310K.

To better visualize how this filtering works in time, here's two examples, one with fast filter, and one with a slow one:

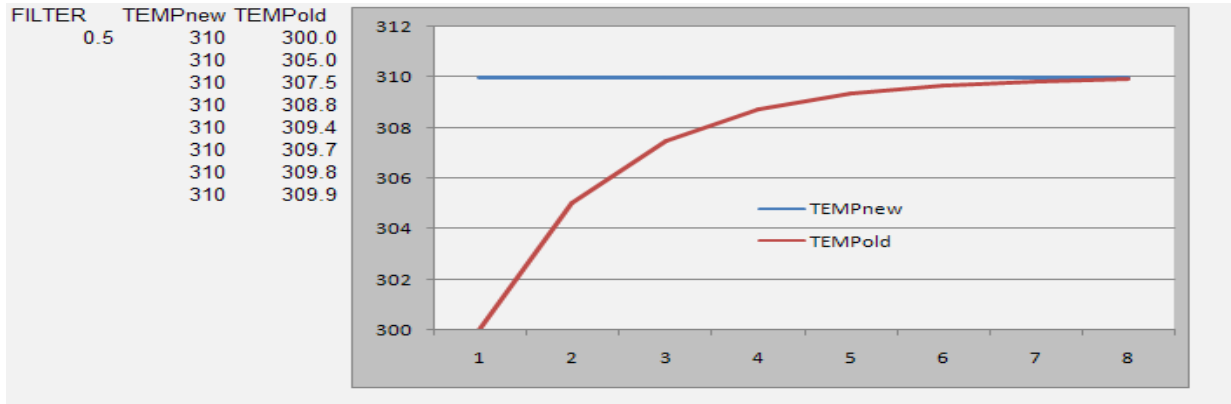


Figure 4--Fast filter

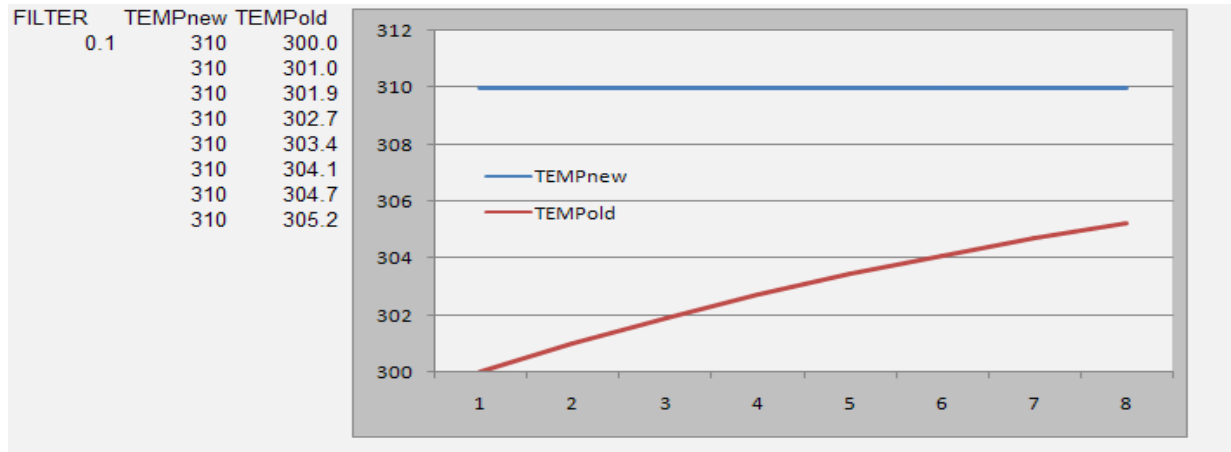


Figure 5--Slow filter

Temperature blending explained

Temperature blending turns out to be simply application of the two engineering math tricks described above. First the TEMPnew gets calculated according to the BIAS table. This value becomes TEMPnew for the BIAS filter portion. The picture below demonstrates how the raw, and the filtered values compare to the scanned manifold temperature.

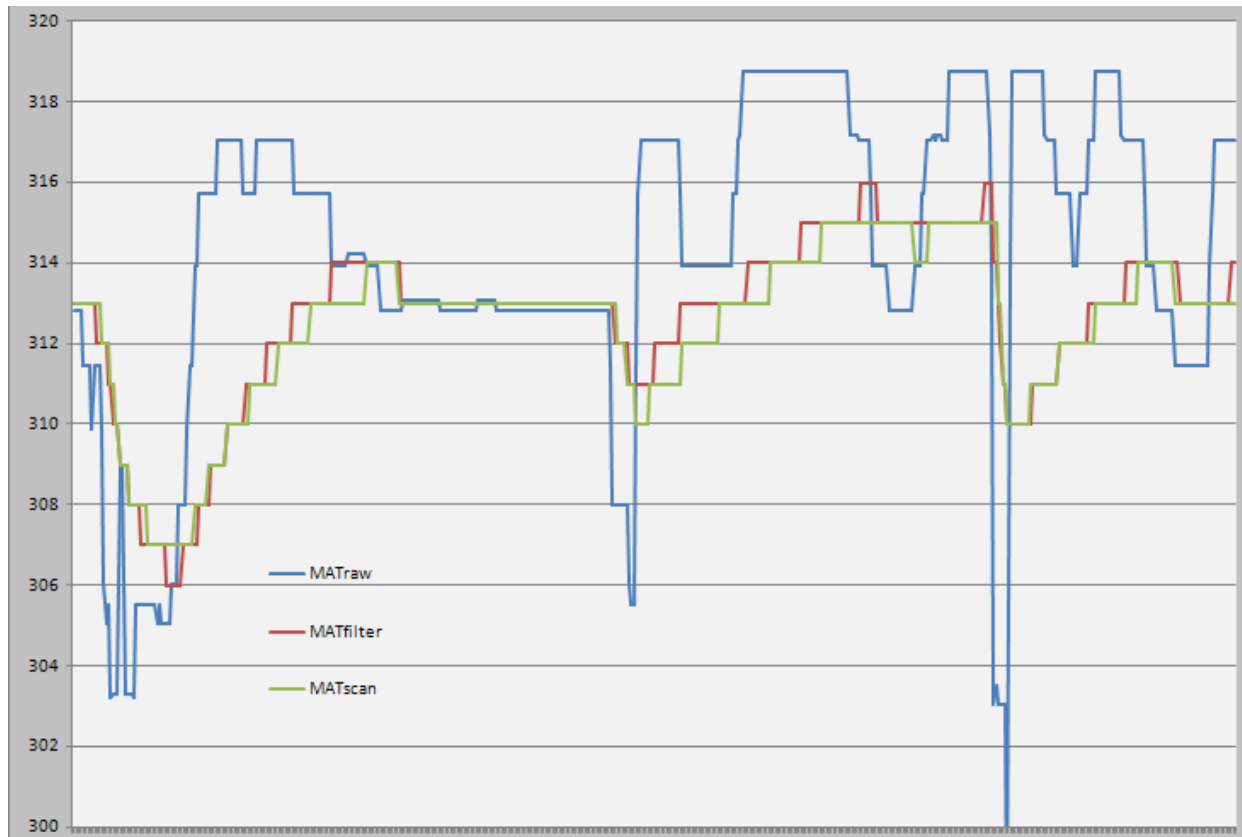


Figure 6--Three temperatures in time

As we can see, the blue data series (MATraw) represents values calculated using only the BIAS table, without any filtering. It's quite jagged and moves about quite a bit, which is quite a different behavior from the green series (MATscan) which is the goal. The red series is the result of both BIAS and Filter calculations, and it's almost on top of the green series, which is the desired effect.

The Simulation

To verify this theory I built an Excel spreadsheet simulating every step of this process. Excel is actually a very good tool for it, as it easily simulates concepts like 'previous temperature' just by pointing to a row above current. Because of the nature of this process, it is impossible to do this directly in the scanners of both EFILive and HPTuners, as they do not deal with the concept of time, or even frame-number-based addressing.

Data

The easiest way to verify a model is to create a simulation, using the stock values as the parameters, and the simulated values should be very close to the scanned values. HPT version 2.2

added the Manifold Temperature MPID, and EFiLive already had this `_DMA` PID, so we're able to peek at the temperature the PCM calculated. However, the resolution of the MAT PID is only 1*K, and makes the MATscanned data series like a stepwise function, which it is not in reality. For our purposes, as long as the MATfilter is within a degree or two of the MATscan, we should claim victory.

To see a clear relationship, I decided to test this on a relatively new and stockish car. A volunteer from Canada with a new LS4 based Pontiac provided me with some excellent data. At this time of the year, in Canada the temperature swings over 50°C within one day sometimes, providing a large spread of data. Also, in the LS4, a transversely mounted V8, the intake is pointing toward the cabin, and it's just few inches above the exhaust headers, being perpetually heat soaked. I figured that if we can make a car like that to account for temperature correctly, then it should be universal enough to deal with many other cases.

The Algorithm and Metrics

The assessment of my model is going to be based on two metrics: one is a sum of squares of differences between the scanned and the calculated values. The second metric is an average error across the full dataset. Another 'metric' is graphical; I want to see the two temperatures track closely as the time progresses. You have seen an example of this already few sections behind. This is a 'reality check' as sometimes numerically superior answers often do not tell us a full story.

The sum of squares is useful as the main value to optimize, as its quadratic nature puts more emphasis on bigger errors. The average error is more informative, as the MATscan value comes only with 1*K resolution, it will not be a precise number, however it will give us a clue to whether we're usually within a degree or two, or we're off by 20.

The main calculations are performed by looking up values of the BIAS filter and the BIAS value for each particular airflow figure. The PCM interpolates between two known numbers if it has no precise lookup value. Excel has no easy way of doing interpolation, thus for the sake of this simulation, I went with a simple 'nearest neighbor' lookup. This is potentially causing some imprecision, but ultimately, the general relationships should remain the same.

Test runs

I ran a few different cases. The first and the most important test was running the simulation on the data generated by a tune with the stock calibration values. They should create a very close fit with the scanned MAT values. If they don't, my model is missing something and needs to be redone.

Another interesting twist on such a test would be to start with a set of randomly picked values for both BIAS values and filters, and optimize them, hoping to see the optimized values converging on values resembling the stock calibration.

Optimizing the stock values could be another potentially eye-opening, to see if it was possible to improve the stock curves.

The cool part about having a real simulator with a metric instantly describing the goodness of a fit is the ability to test the real long shots--BIAS curves I've seen in other cars, as well as common sense adjustments, or even random patterns.

Observations

The first experiment I anticipated the most, as I just wanted to see if my model is even remotely close to what the computer does. Surprisingly enough, both the numerical and the graphical representations looked very good right away. Stock calibration of the LS4 computer, even though it had an unusual shape comparatively to other popular platforms, yielded about 0.83*K of difference on average, between the scanned and the simulated temperature values.

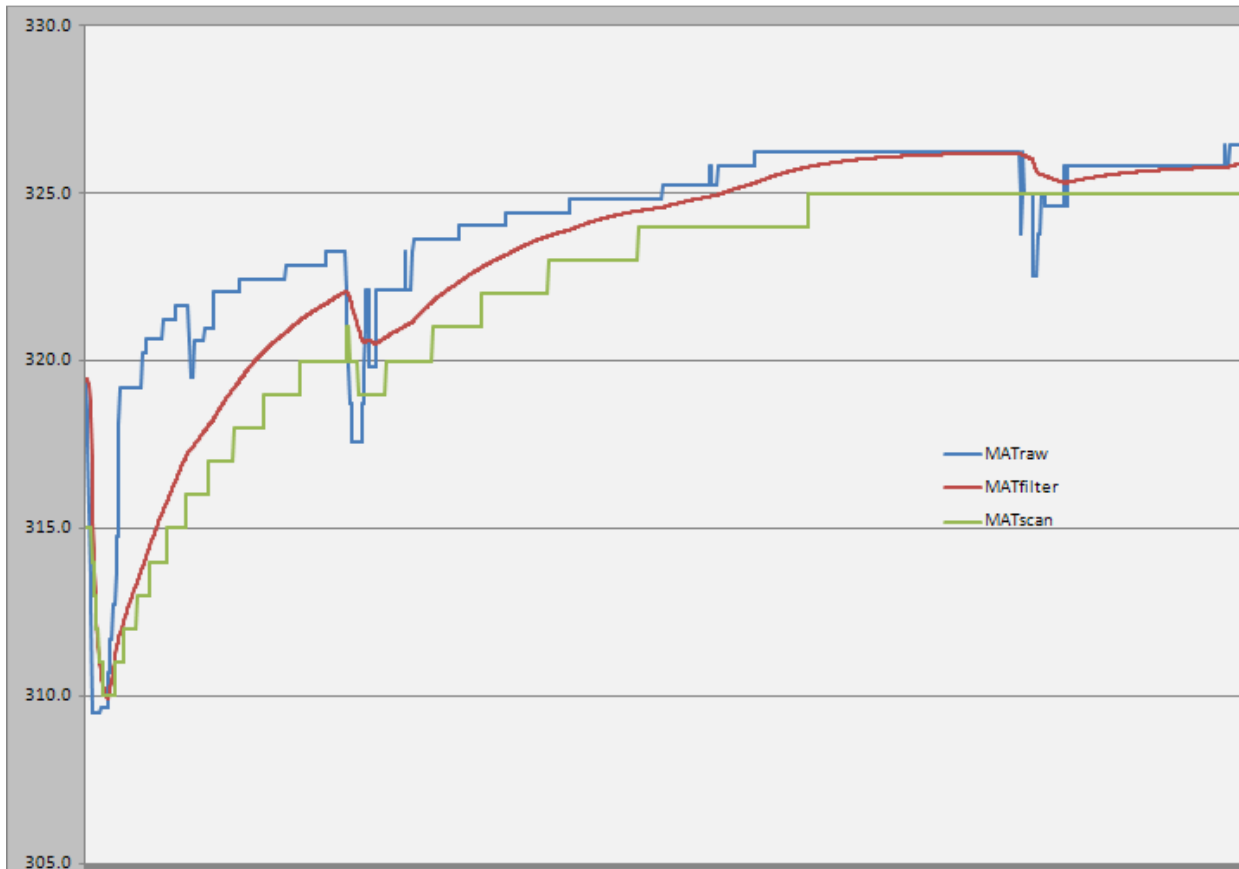


Figure 7--Temperatures resulting from stock calibration

The second experiment was run out of sheer curiosity. Amazingly, Excel's optimizations resulted in an even more dead-on tracking between the expected and measured values. This time the average error was about 0.30*K, which is amazing, considering I'm not even interpolating table values. The fun part was noticing the differences between the stock and the optimized results. The values did change only a little, but the general shape of the curves remained the same! To my surprise the filter curve was bent down a little, as if the computer needed to adjust between old and new temperature readings a bit slower. The BIAS table remained almost identical to the original, with only minor changes.

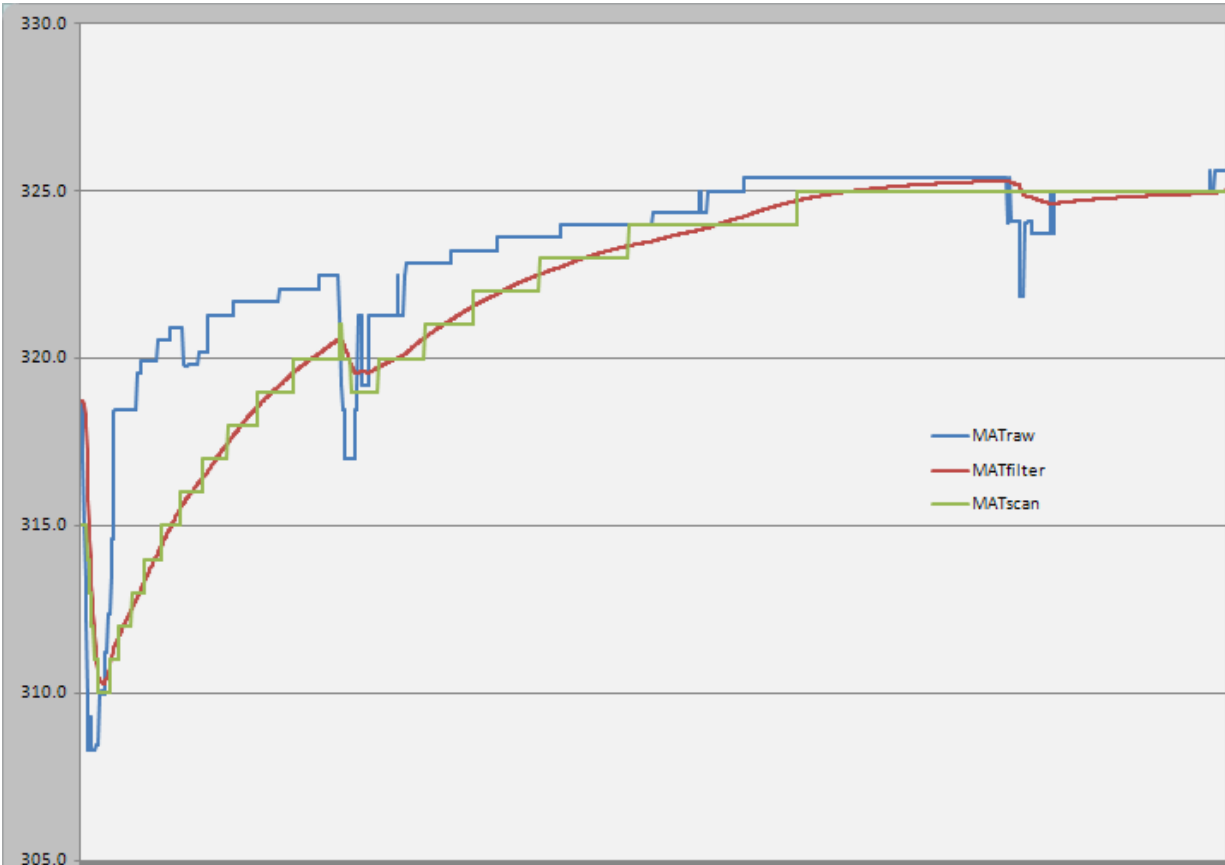


Figure 8--Temperatures from optimized filter and bias values

As we can see on Figure 8, the MATfilter follows MATscan much closer. In Figure 7, MAfilter is consistently higher than MATscan, but after the optimization it's consistently on top of the MATscan series. Here's a comparison of the optimized values to the stock ones, for both BIAS and filter tables:

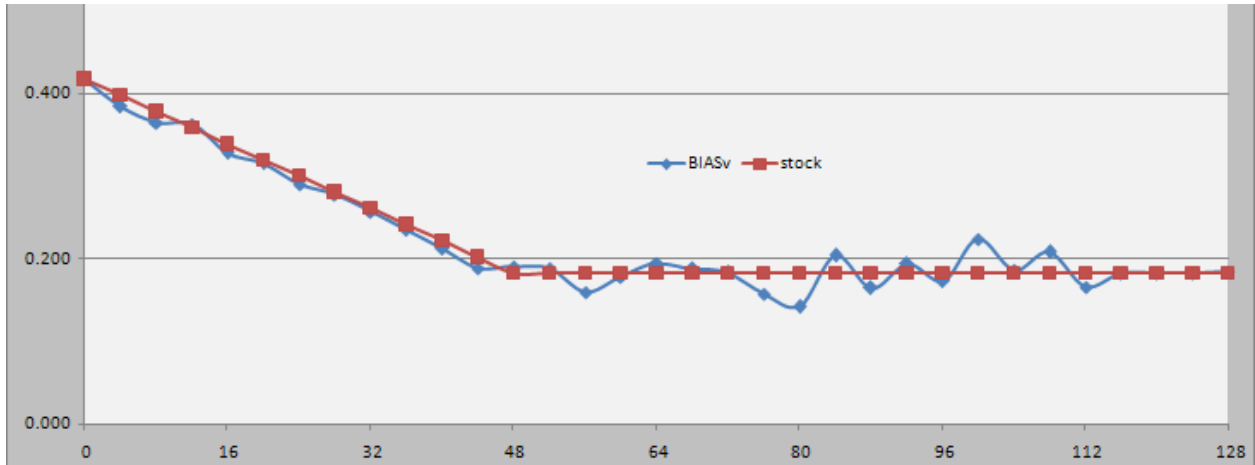


Figure 9--BIAS values, stock vs. optimized

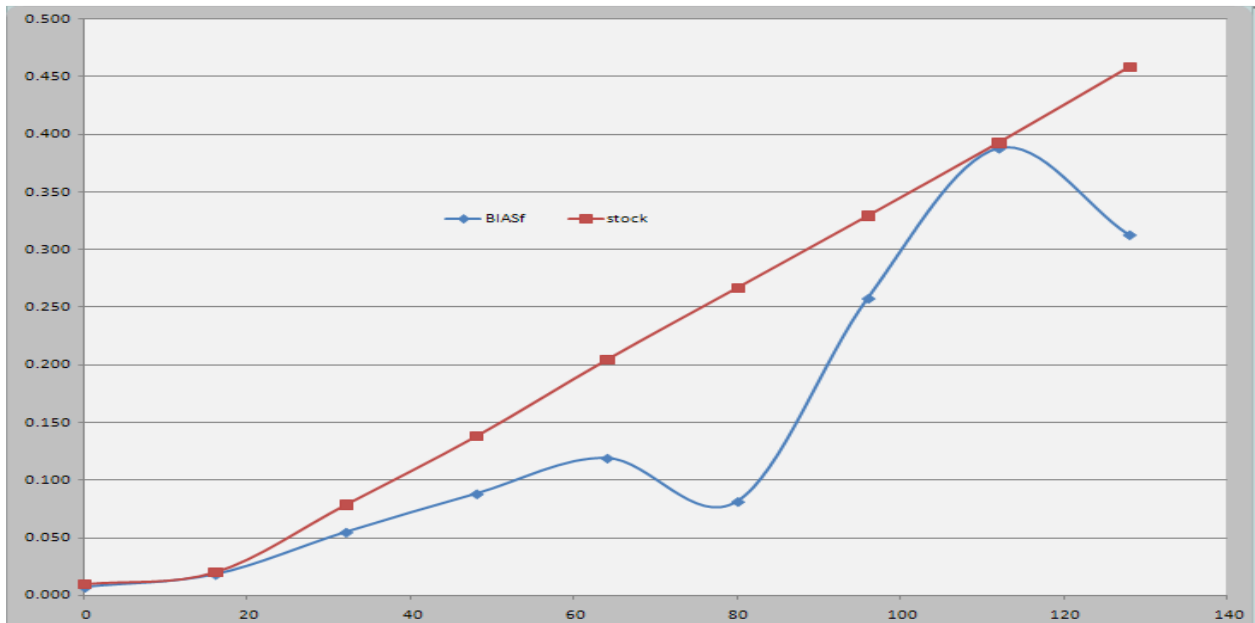
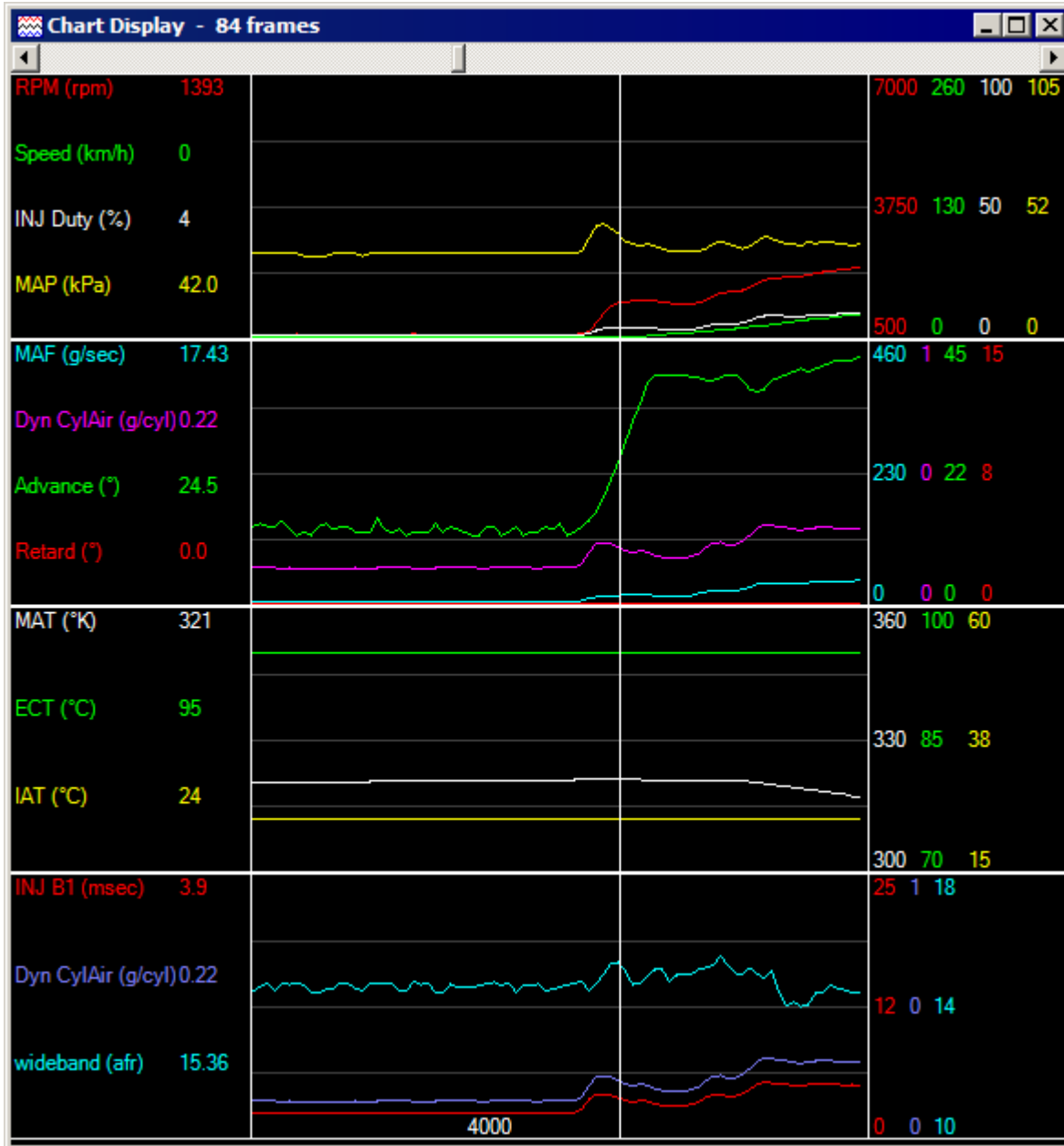


Figure 10--BIAS filter values, stock vs. optimized

These values were inserted in the configuration for the stock LS4 car. The result is a smaller AFR%Error on takeoff:



The third experiment, playing with many other known stock configurations was a long shot, but in the worse case, I figured, we'd we what a bad approximation looks like. I did not have to wait long-- the first set of parameters I threw in (stock 05 GTO numbers) yielded an average error of 6*K, which is seven times worse than the stock numbers and over 20 times worse than the optimized numbers.

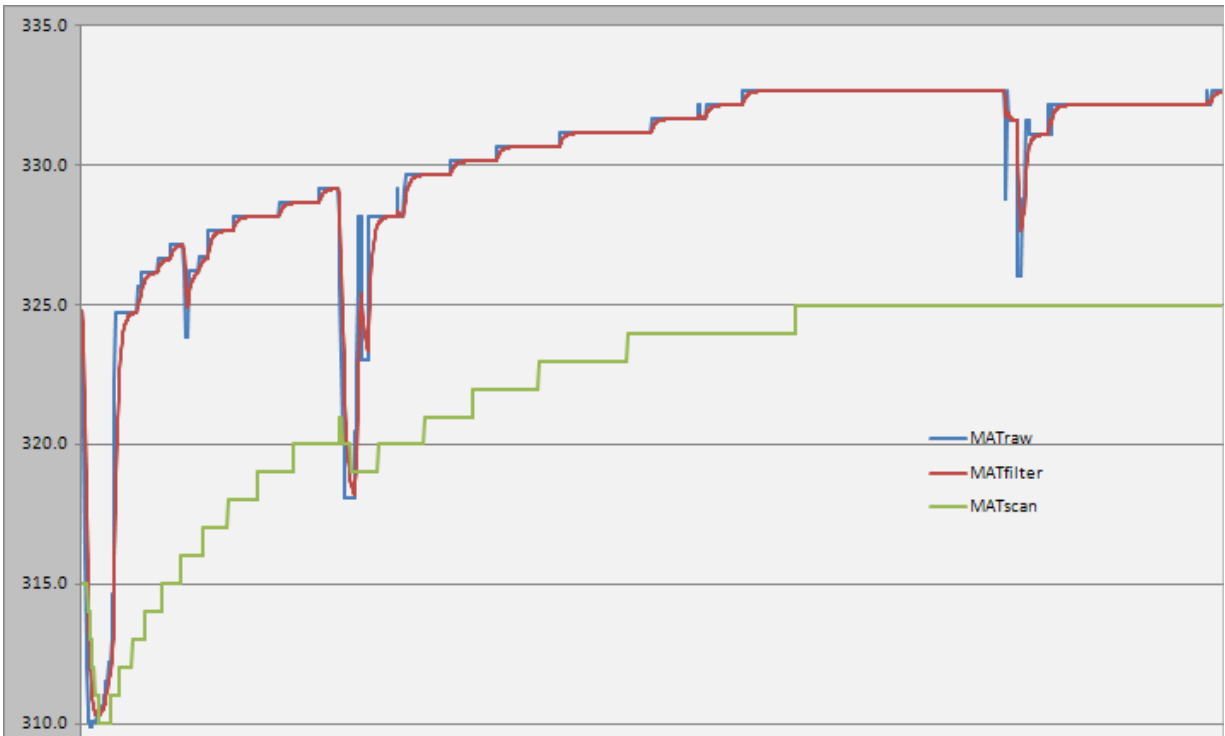


Figure 11--A completely different BIAS values and filters result in a completely wrong estimation

Another experiment I tried was starting with random numbers, and then optimizing them to see what kind of order we can extract from total chaos. Random numbers yielded horrible results, and in few cases I was able to even crash Excel. However, when optimized, the solver yielded a very interesting set of points, that seemed nowhere near as neat as the stock curves, but surprisingly yielded a very good final fit with only a 0.73*K average error.

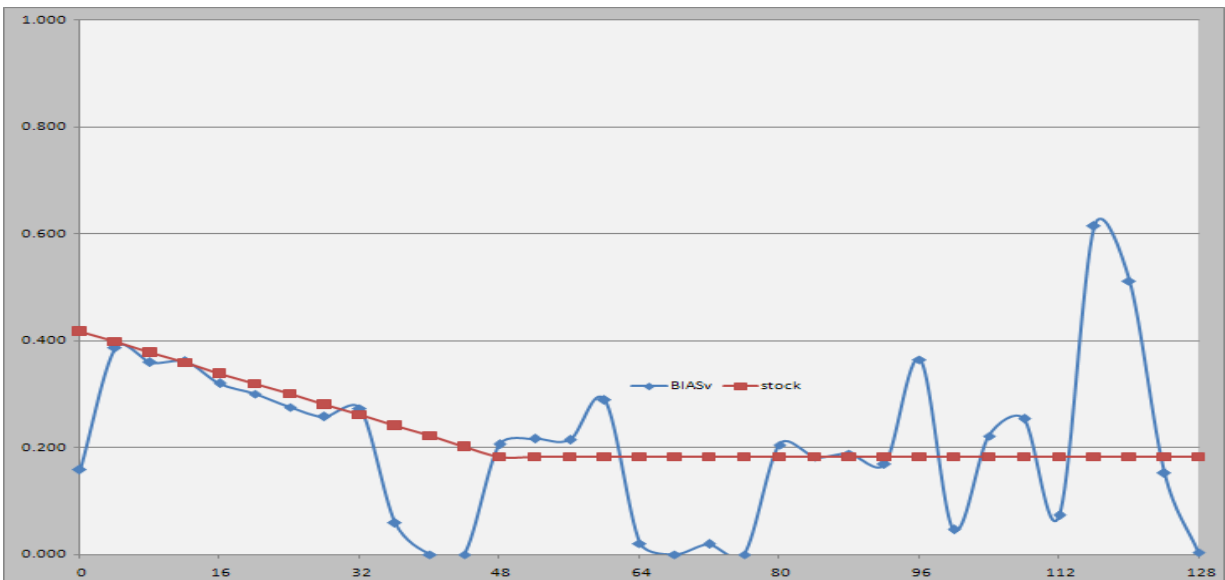


Figure 12--Optimized random BIAS curve

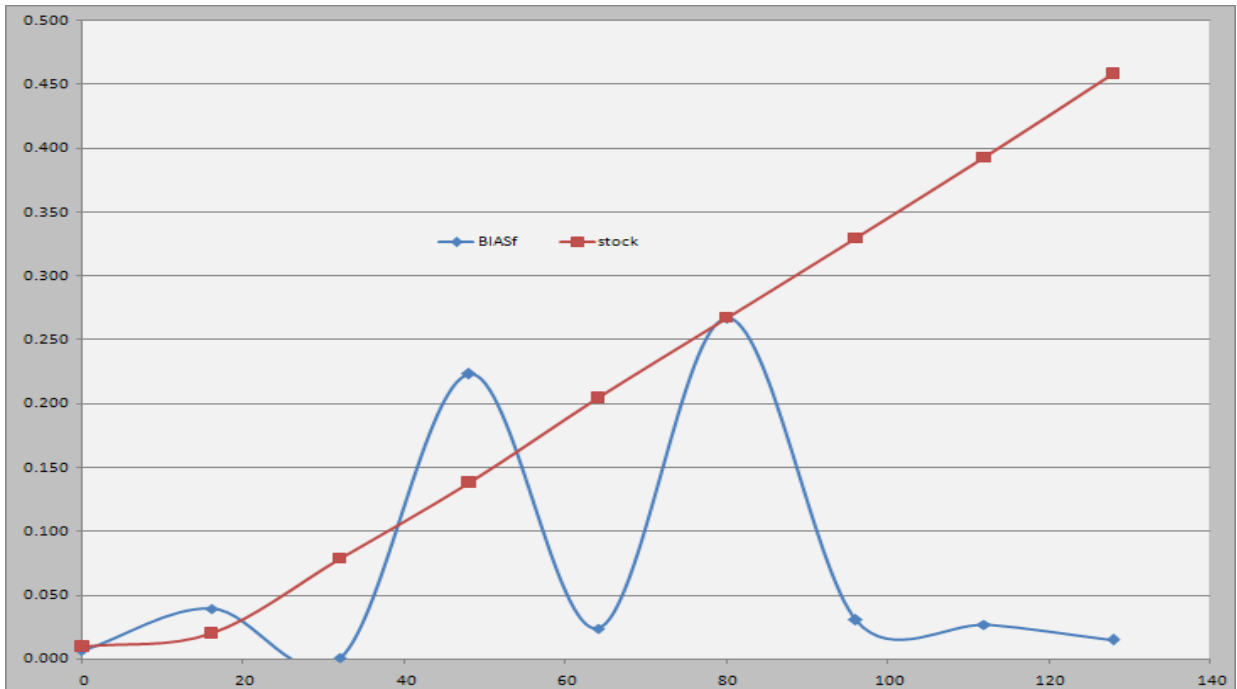


Figure 13--Optimized random BIAS filter curve

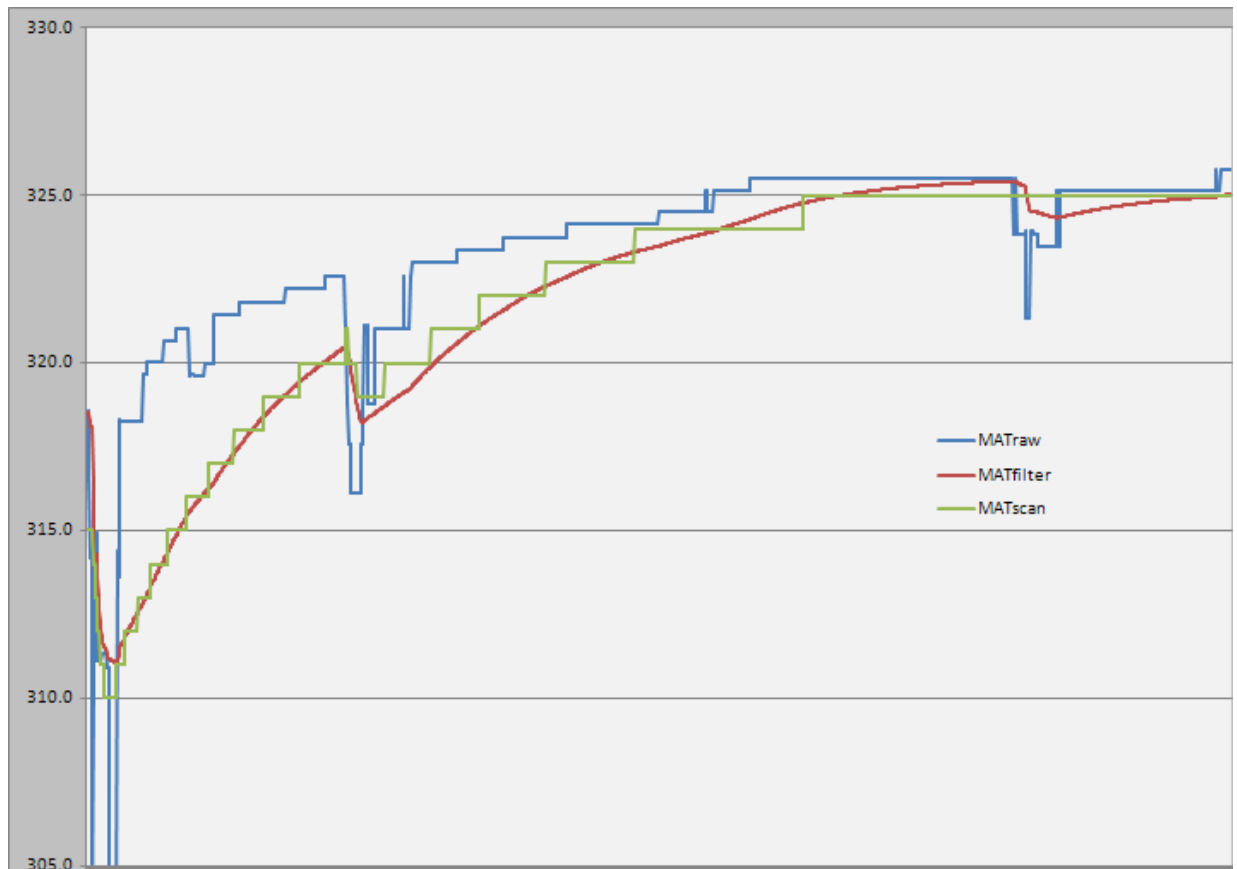


Figure 14--Temperature curves generated by the optimized random BIAS tables

Conclusions, Impact, and Future Works

As we can see, even the small changes to BIAS values or the BIAS filter can result in significant changes in calculated temperatures. We learned that the model I proposed in this paper yields a close (within one degree K) approximation. So what does this model and the simulator really give us? As of right now, by itself, not much. If we use the traditional 'AFR%error' type of method to adjust the BIAS values, we'd end up tuning the whole car with temperature estimation, making up for imprecisions in other places, like the VE/GMVE tables, or the miscalibration of a wideband sensor, or injector's unmatched flow, or the injector calibration data... You get the picture, the list of the potential sources of problems is entire too long, to be compensating for everything with two temperatures tables.

However, when combined with some other research I've been working on the past year, these temperature estimation techniques should allow me to calculate GMVE values fully accounting for temperature swings, and thus result in an all-season Speed Density tune. But that's a whole different story...