Automated Tuning Of The Vehicle Control Module In A Car

Submitted To

Kyungtae Han

Prepared By

Sam Harwell
Will Thomas

EE464 Senior Design Project
Electrical and Computer Engineering Department
University of Texas at Austin

Spring 2005

CONTENTS

LIST OF TABLES

LIST OF FIGURES

EXECUTIVE SUMMARY

In modern engines, there are many aspects which can be modified to get an increased performance. One of these systems is the vehicle control module (VCM). This microcomputer specifies several of the aspects of a car's drivability, including the spark plug advance and the fuel table. Often when people make modifications to their cars, they forget about one of the key changes that they need to make to their car, the tables inside of this VCM. Our project aims to automate this process, thus reducing the cost of this crucial step in a cars tuning. With cheaper cost, hopefully people will be able to tune their cars more often, thus increasing the drivability, safety and lifespan of the car.

Of course, this automated tuning process would be pointless if it failed to meet several design criteria. The project aimed to increase the gas mileage, power, and drivability of a car which we automatically tuned. Another key to this process is to minimize human interaction with the actual decision making. If the code still required many human decisions, it would fail to improve the level of tuning efficiency to be worthwhile. Also, the whole process has to be performed offline. Optimally no extra hardware would need to be purchased to use the software on a car other than some simple interface equipment.

We began our approach by splitting the project into three distinct parts: the fuel map, the modifiers and the spark table. The fuel map was tuned using a mean squared error solution. This was done using a series of matrix equations. Our modifiers were done in much the same way, but iterations were required due to the dependence of these variables on each other. The spark map problem proved to be much more difficult. One problem we ran into was an inability to get the materials we needed from a shop. After countless tries to retrieve the sensor, we had to give up. At that point in our project, time no longer remained in which to design, code, and test the algorithm we had planned.

Before we could actually do any of the calculations of our modified maps, however, we had to deal with some preprocessing of the logs. This involved log alignment and some filtering of the logs to help smooth and remove outlying data points. Both of these proved to be difficult to perform in the final code. Alignment included not only the start and stop points, but also variations in the timing at irregular intervals in the log.

Although our project failed to produce an increase in gas mileage, it did provide an increase in drivability. The power test could not be performed due to time constraints and monetary cost. While performing the project, we followed legal guidelines regarding safety and environmental issues. With the increased drivability in the car, we also believe that that there is an increase in safety due to the project. With an easier response of the vehicle to the driver, the driver is more likely to be able to react to an obstacle and avoid incident.

Overall we were pleased with our results in this otherwise untapped field of signal processing. We were able to make an automated system of sorts to perform the tuning of a fuel map in a car engine. With further modification to the ideal maps, an increase in fuel efficiency might have been reached. We further suggest that an offline mode be researched for the tuning of a spark table, and believe that acoustic analysis is the way to succeed in this. If this spark tuning was combined with our present fuel system, the resulting savings could mean a more conscious society about the electronic tuning of their cars.

1.0 INTRODUCTION

This report documents our project on the automated tuning of a cars vehicle control module (VCM). In this project we attempted to optimize the computer inside of a car's engine. This computer has several different tables which control various aspects of how the car runs including the fuel injector timing and when the spark plugs fire. This has been conducted by Sam Harwell and Charles Thomas, senior electrical engineering students at the University of Texas at Austin, under the guidance of Kyungtae Han. In this project, we expand the field of signal processing to encompass a process normally reserved to human interaction. Through the use of computers in the field, we were able to achieve results that greatly enhanced the drivability of a car.

2.0 DESIGN PROBLEM STATEMENT

In this project, we expanded on an existing practice of tuning cars. When a modern car comes from the factory, it is tuned optimally for performance and gas mileage under a set of guidelines. Many people, however, make modifications to their car engines which alter how a car responds under various conditions. Even if these physical modifications are not made, a car can stray from the optimal computer tuning over time. Currently, if a person wants these tables tuned they must take it to an expert. This expert would then manual edit the tables based on the visual processing of various graphs and feedbacks presented. This can often be a long process, done mainly by trial and error. This can also be costly, with one reputable tuning shop charging $975 a day plus expenses for a tune [1].

2.1 GOALS

In order to make this process more affordable, our project aimed to automate it. To do this we needed to first set several goals for our project. First, we knew that we wanted the project to require as little user input as possible. Optimally, this would mean the interaction of our program directly with the VCM. It also means that a direct calculation of at least the basic values is necessary. In addition to this, we did not want the project to require the user to buy additional hardware. Therefore, the project would require all of

the calculations to be done in an offline mode, with the only changes being made to the tables inside of the car. In no way could the project alter the method that the cars engine was run, only the tuning of its current tables.

We also knew that we would need to meet several more practical criteria for our final solution. In order for the tune to be of any use, it would have to increase the drivability of the car. Also, we started the project with the goal of increasing the fuel economy of the engine inside the project car. While increasing this gas mileage, we also set out to increase the power output in the performance segments of the cars drive, such as wide open throttle.

All of this had to be accomplished within a limited budget and the time constraints of one semester of course work. This meant that we needed to try to keep our project under $100 (we knew early on that we would be unable to include gas cost in this budget). This meant a limit on the new hardware purchased as well as a time crunch for some more involved signal processing techniques.

2.2 BASIC APPROACH

To do this we divided our project into several basic parts. The first part of our project involved the tuning of the fuel map in the VCM. This table tells the car how long the fuel injectors should be open under a given manifold air pressure (MAP) and revolutions per minute (RPM). To tune this table we compared air fuel ratio (AFR) in parts of the map with a theoretically optimal table for each point. We could then minimize the mean square error (MSE) of the AFR by performing certain calculations on the fuel table.

Another part of our project was to optimize the modifier tables in the engine. These values are used to compensate for various conditions in the engine. Several of these modifiers focus on the temperature in various places in the car engine, and we chose to focus on these due to time constants and their apparent importance on the drivability of the car. To solve for these we used a similar method used for the fuel map. Taking logs of the data from an actual run, we then operated on these values to try and minimize the overall MSE of the AFR in the engine under the temperature conditions.

The final part of the project initial decided would be beneficial was the tuning of the spark table. This table tells when the spark plugs fire in relation to the engine timing. To do this we had original decided to do some acoustical analysis of the spark plug firing to determine how to adjust the timing of the spark. We later had to drop this portion of our project due to time and budget constraints.

3.0 DESIGN PROBLEM SOLUTION

For our product, we decided that a coded MATLAB solution would be the method we would use for implementation. MATLAB contains many of the matrix manipulations that we would need in order to do much of the data filtering in the project. It also was able to handle the file sizes we needed to process in the data relatively quickly without going into a lower level language (C++, assembly). For the project purposes given the time constraints, the abilities and tradeoffs seemed ideal.

To begin with we broke the design into 3 basic sections. These sections were the fuel map, the modifier tables, and the spark table. For each of these sections we evaluated several alternatives and choose one that best suited our needs.

3.1 FUEL TABLE

The largest goal of our project was to find a way to optimize the fuel table in the car. For this method we evaluated several possible alternatives in the implementation.

3.1.1 Trial and Error

The first implementation method we looked at was a simple trial and error method for tuning. This is the method used in the after market tuning of cars today. In this, a professional looks at the AFR of the car under certain MAP and RPM regions of the map. The tuner then can change the values in the fuel table based on basic knowledge and past experience with similar cars. The tuner can then look at the new performance maps, and continue updating them until he is satisfied with the results of the tune. The problem with this form of tuning lies in the need for human feedback in the system. Even if a method could be formed without the human interaction, it still results in a method that is

not offline in anyway, and therefore fails two of our basic constraints, automation and offline ability.

3.1.2 Iterative Solution

Another method that we took a brief look into was an iterative solution. Similar to the trial and error method, this would be more like a plus minus move in small increments. Through repetition, and reduction of the step size, a solution could be asymptotically approached. This solution also ran into the problem however of needing to be run online, with many modifications of the car's computer.

3.1.3 Mathematical Solution

The third approach we looked into was a mathematical optimization of the fuel table. By looking at current output values as well the current tables, and matching them with a theoretically optimized table, we were able to minimize the MSE between the two values. This allowed us to optimize the tables in an offline mode, which was unavailable in the other methods we examined. Because of this ability, we chose to use this method over the other ones mentioned. Although the coding for this solution was much more intensive than the others would have been, we believed that it would meet all other requirements of the project guidelines. We later found that this solution would require us to exceed our initial cost guidelines, but on further examination, this proved that it would have been unavoidable to get the desired results.

3.2 MODIFIERS

For the modifiers section of the project we also came up with several possible solutions. These solutions also each had their positive and negative aspects.

3.2.1 Simulated Environment

One option we explored was the use of a controlled environment to run the engine in to set the modifiers. This would involve running the engine in an environmentally controlled chamber. With this we could individually tune each modifier without having to worry about other varying conditions in the car. Even if a physical environment could not be created, it might have also been possible to make a complete simulation in

software given enough time and computational power. With either of these options for simulating the environment it would be extremely costly to perform. The number of variables needed for a computerized environmental simulation is astronomical, and would in itself be more time consuming than could be approached in the three months available. The monetary cost of using a controlled environment can easily be thousands of dollars an hour assuming that the environment is already available. In addition, this would require removing the engine from the car which would be too timely to perform.

3.2.2 Gas Laws

Another option we looked into was the solution to the modifiers using ideal gas laws. By calculating how the car engine and fuel mixture should react to various temperature and pressure combinations we could provided a modifier table to compensate for this. This option initially appeared to be much more viable than it turned out to be. When we actually used this implementation the result was that our car began to run too lean in some areas and too rich in other areas of the temperature range. We later found through our MSE solution that although the solution to this would theoretically be ideal, the actual solution to optimize the way the car ran was much different. Although this solution appeared technically sound, there were more variables than the apparent ones, making this solution unusable.

3.2.3 MSE Solution

Again, the final solution we came up with involved minimizing the mean square error of the AFR as it changed within a modifier map. This was done slightly differently than the fuel map solution, since we had to iterate the solution several times to get good results. This was because the variables were not fully independent, meaning that multiple iterations were needed to optimize all of the modifiers we tried to update. Overall, although this did not match up with our theoretical solution, it minimized our MSE and gave us the optimal performance of the vehicle.

3.3 SPARK MAP

This proved to be the most difficult part of our project. We ended up having to abandon this portion of our project after researching possible solutions and spending time on the

other parts of our project. Due to a variation of time constraints and cost, we did not feel that it would be prudent to attempt this portion at the time of our project.

3.3.1 Neural Network

This was one of the options we looked into for the spark map of the project. Since the position of the spark firing has no direct formula, we believed that the neural network would be an ideal solution to the problem. Upon further consideration, however, we decided that it would be very time consuming to obtain the data for the correct values to train the neural network with. In addition to this, we would still have to use one of the other methods to obtain the data for the neural network to work with. This made it clear that a neural network was not the method that we should use for the spark map in the project.

3.3.2 Knock Adjustment

The most commonly used method for adjusting the spark map is through knock adjustment. With this method, the firing position of the spark plus is slowly advanced until knock is heard. Upon hearing this know, the spark plug firing is backed off by 2 degrees, and left as good. This however, is a very conservative tuning, but is unlikely to result in engine damage. Also, if you back off the engine by less than 2 degrees you run the risk of leaving inaudible knock in the engine. The major problem with this method is that it requires human interaction. This does not comply with the automated requirement of our project.

3.3.3 In Cylinder Pressure Sensor

This method is the one commonly used by automotive manufacturers to tune their spark tables. The in cylinder pressure sensor can tell the user the optimal point of spark plug firing. The major problem with this method is that it is very high cost. This requires building a special engine with new holes for these sensors. This is obviously not obtainable under the budget restraints of the project.

3.3.4 Acoustic Analysis

This was originally the most promising method for spark plug advance. With this method, an analysis of the in cylinder acoustics would have to be performed. This requires a specialized microphone called a knock sensor. By analyzing the readouts of the microphone, we believed that we might be able to determine the optimal firing point through use of second and third order feedbacks in the sound system. The major problem we faced in this approach was obtaining the knock sensor. After contacting Holly, the VCM manufacturer, we were told that we needed to go through a specific supplier. We contacted this supplier several times over the course of about a month and a half, and each time we were given the response that the product was being ordered for us. After this time we determined that even if we received the sensor in time, we would be unable to complete this portion of the project on time. Also, the most extensive articles found on the subject suggested that a feedback system would be the best way to do these operations is using a feedback system [2].

4.0 DESIGN IMPLEMENTATION

In our original design, we solved for the each cell in the fuel and modifiers individually, and used a merge algorithm to build a single map from the results. Figure 1 shows the surface of a single cell in the fuel map. The cell was “solved” by finding the optimal fuel values at the corners to minimize the mean-square error between data points in the cell and the interpolation surface. Our MATLAB code to solve the cells and merge the results is included in Appendix A.

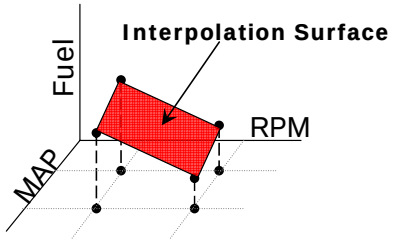

Figure 1. Inverse Bilinear Interpolation Of A Single Cell

Over the course of our project, we made several changes to our original design. While many of these changes were made to include improved algorithms, some reflected trouble spots we encountered and the methods we used to work around them.

4.1 MAJOR CHALLENGES

Major challenges are problems we encountered over the course of our project that were unexpected. The combination of several of these challenges greatly limited our progress rate on the project, but over time we ended up finding usable solutions to nearly all of them.

4.1.1 Inaccurate Oxygen Sensor

One challenge we faced in this project was simply the collection of useful data from the O2 sensor. The built in sensor was a narrow band sensor which had very limited resolution. About the only information that the narrowband sensors can provide is whether the oxygen ratio is precisely at the stoichiometric level. Any slight variations from this and there is a large change in voltage as seen below in Figure 2 below.

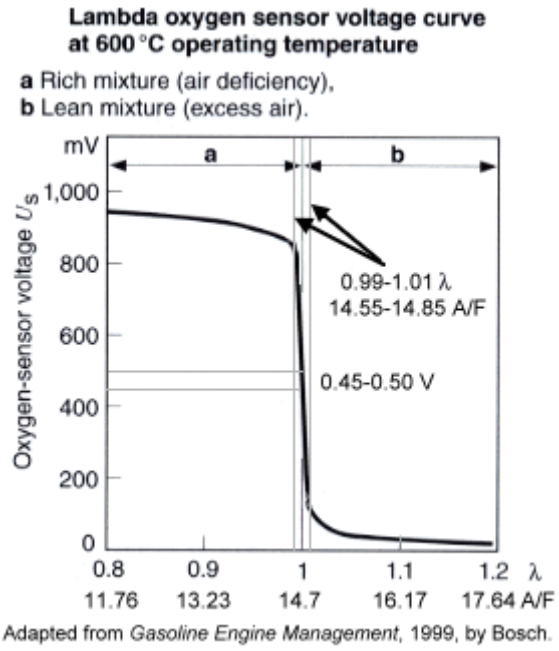

Figure 2. Narrowband Oxygen Sensor Voltage Response

Since the idealized fuel map values are not at stoichiometric in all regions of the idealized map, this sensor was inadequate for tuning our fuel map. In order to solve these response problems, we had to order a new sensor with wide band capabilities. The response of a wideband sensor to changes in the fuel levels is much more linear (Figure 3), making it possible to observe other fuel ratios. Although this new sensor cost $600, we felt it was a necessary modification to proceed with the project and could not be avoided. With this modification we were able to get the resolution we needed to correctly determine the effects of the fuel map on the AFR.

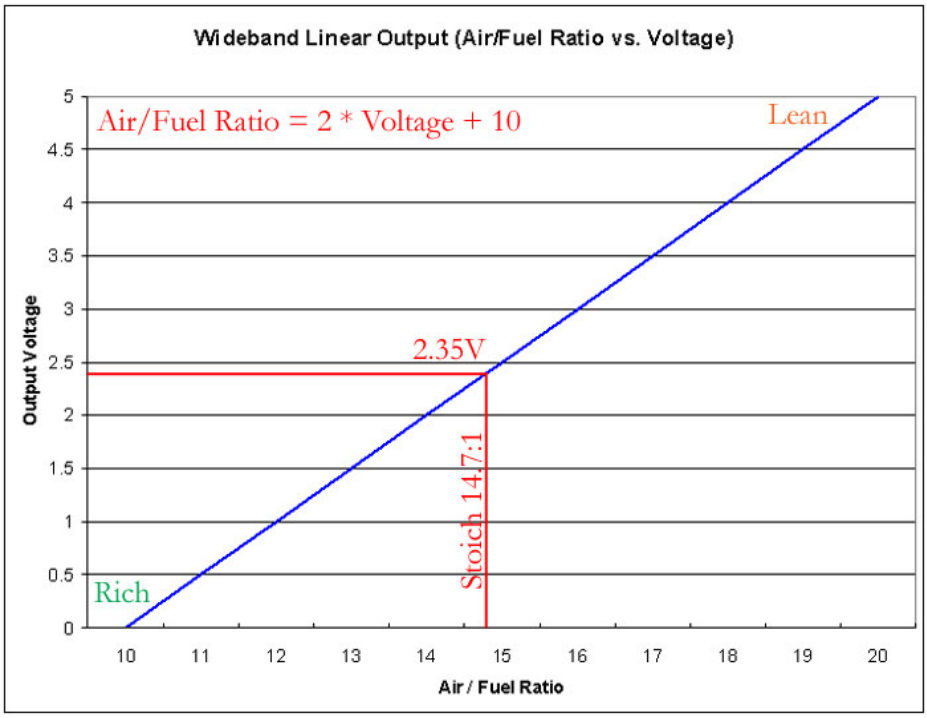

Figure 3. Wideband Oxygen Sensor Response

4.1.2 Misaligned Logs

The largest problem we encountered over the course of this project was misalignment in our data logs. We found that the real-time logging system logs samples at varying sampling rates. Over the course of a long data-log, we found that timing differences of 20 seconds or more were present throughout the logs. In a system that relies on accurate point-by-point measurements, an uncorrected offset of even a few samples renders the

data unusable. Figure 4 shows the offsets in samples present in our 1 hour and 45 minutes test and validation log. Without correction, the test would come to a stop with this plot.

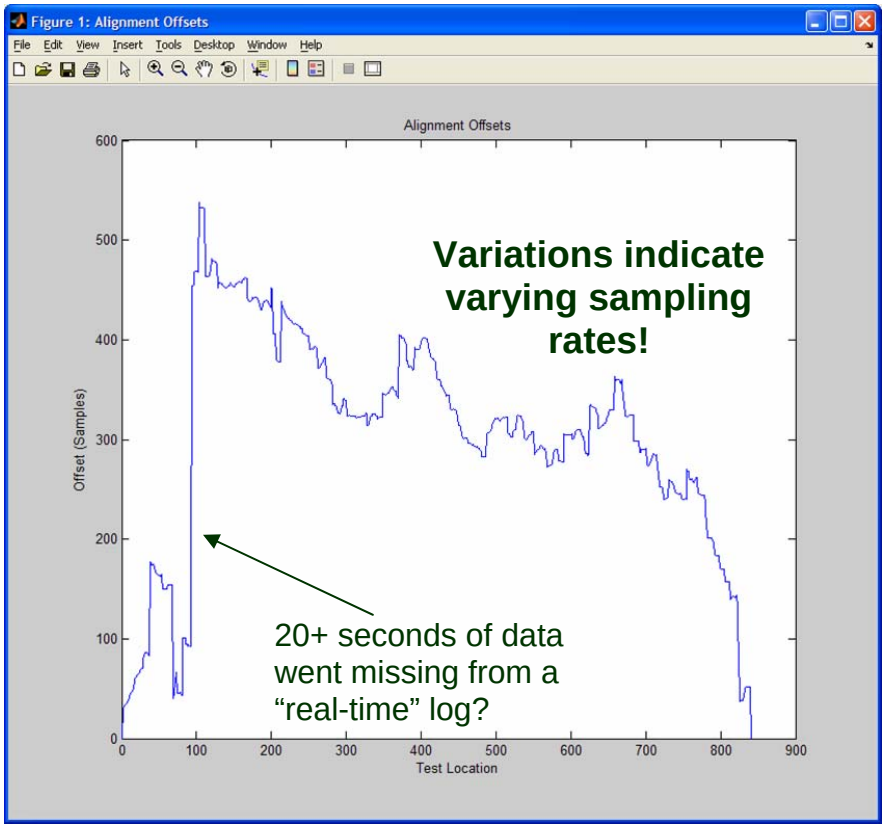

Figure 4. Data-Log Alignment Offsets

To solve this problem, we developed a correlation algorithm to detect sample offsets of small data windows throughout the log. To estimate offsets at locations that were not tested with a window, we used a linear interpolation based on the offsets of the two closest test windows. To start with, we used a simple pure correlation algorithm. We found that it occasionally failed to detect a proper offset when a large spike was present in one of the windows, or when the signal was too small to accurately detect an offset due to noise.

We updated the correlation algorithm by dividing the correlation result by the sum-of-squares energy present in the difference between the two windows. By heavily weighting the correlation against absolute differences in the windows, this algorithm prevented

extraneous spikes from affecting the detected offset. Figure 5 shows how this algorithm can properly detect an offset when the original pure correlation routine would fail. Note the extremely sharp falloff rate when the window shifts away from the proper offset while using the new algorithm. Our MATLAB code to implement the data alignment can be found in Appendix B.

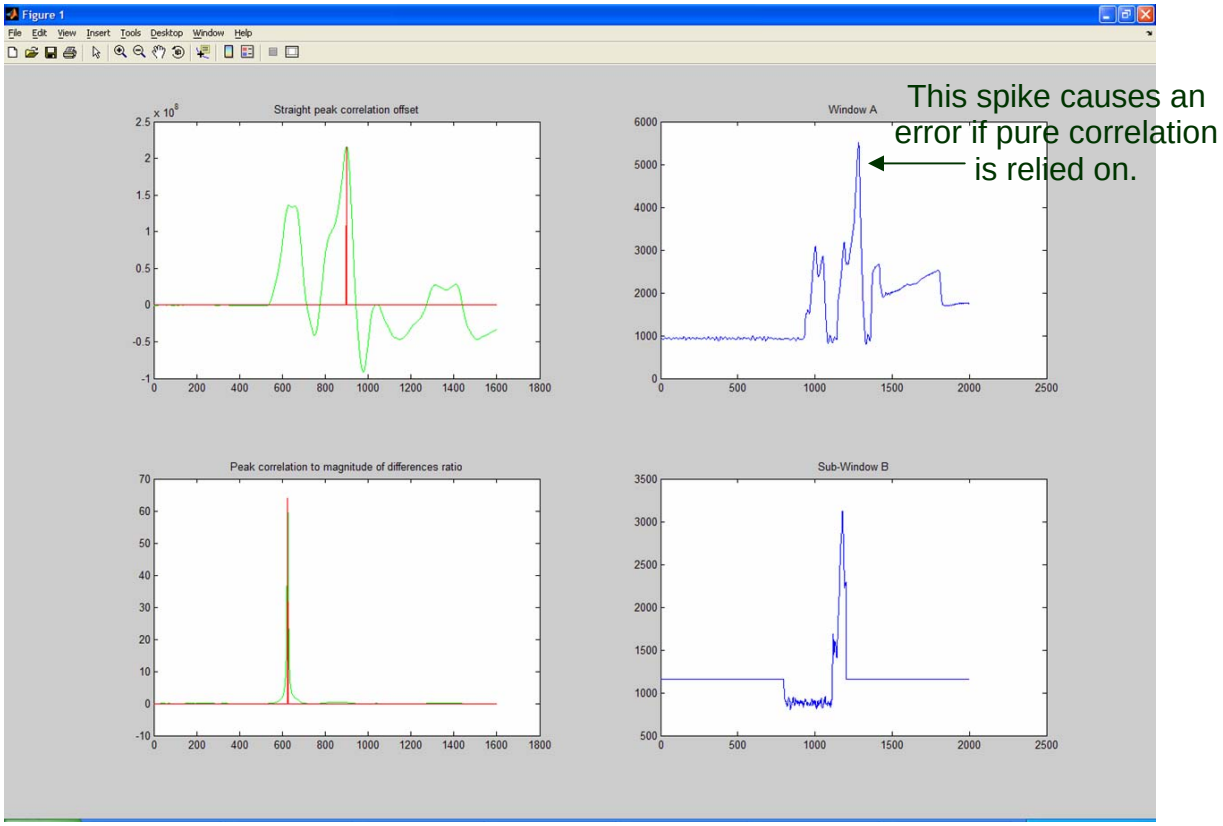

Figure 5. Comparison Of Correlation Algorithms

4.1.3 Sparse Data In Fuel Cells

Over the course of a logging session, we found that most of our data points fell within a relatively small portion of the fuel map. In cells that contained very few points, minimizing the mean-square error resulted in obvious, serious errors in the calculated fuel map. During one log session of over 25,000 data points on 6 March 2004, only 25 points fell across six adjacent wide-open-throttle cells. The calculated fuel map contained one cell with a negative value, and two cells had values over triple the hardware limit of

the injectors. Our estimate of the range of acceptable values in these cells fell between 80 and 100% of the maximum allowed limit.

An important consideration regarding wide-open-throttle cells and other sections containing sparse data is the fact that sparse cells can be extremely important. When a driver presses his foot to the floor, for those few moments he expects the car to operate at its peak power level without hesitation. Inaccuracies in these cells are therefore completely unacceptable in a production-ready tuning solution.

Initially, we avoided problems in these cells by preventing adjustment within cells that contained fewer than a specified threshold number of data points. We raised the threshold via trial-and-error until the results stopped showing sharp peaks or valleys in the output. Later, we improved the performance in sparse cells greatly using the bias method described in section 4.2.2.

4.1.4 Short Laptop Battery Life While Driving

One of the drawbacks of using a high power laptop computer is a limitation of only one hour on battery power. For most of the project, we overcame this limitation by logging data in 30 minute sections and plugging the laptop into the wall before beginning the analysis process. During our later test and evaluation stage, however, we needed much longer data logs to verify the long-term consistency of our results. After contacting the laptop manufacturer, we learned that while they offered an automotive power adapter for our computer, availability and shipping delays would prevent it from arriving before the end of our project.

We went to Fry's, a local electronics warehouse, to find another power adapter. Eventually, we chose a DC to AC inverter, which allowed us to plug our laptop using our existing laptop power supply. The inverter required a direct connecting to the car battery instead of a simple cigarette lighter plug, but our test vehicle's battery had been relocated to inside the car so this was not a limitation. The new inverter allowed us to obtain 1 hour 45 minute logs during the test and validation stage, where we met the limit set by the real-time logging software.

4.2 DESIGN MODIFICATIONS AND IMPROVEMENTS

The basic portion of our design was not static by any means. Throughout the project we had to make changes to many of the sub processes to get good results. Through effective alteration of these processes we were able to get much clearer results from our system.

4.2.1 Matrix Form Least-Mean-Square Solution

Toward the end of the semester, we discussed the general problem of solving linear least-mean-square (LMS) error problems in one of Professor Heath's lectures. He showed that the LMS solution $\hat{x}$ to the matrix equation $\mathbf{A}x=b$ is $\hat{x} = (\mathbf{A}^{\mathbf{H}}\mathbf{A})^{-1}\mathbf{A} \times b$. We defined $\hat{x}$ as a vector containing every cell in the fuel map in row-major ordering. We created the matrix $\mathbf{A}$ by solving for the effect each element of $\hat{x}$ had on a data point after a bilinear interpolation operation, and b is a vector of each target AFR value.

Using this algorithm, we are able to solve for the entire fuel map at once. The resulting MATLAB code is also much shorter and easier to read, leading to a simpler and more reliable debugging process. This algorithm also allows cell biasing as discussed in section 4.2.2 next. We called our implementation of this algorithm our inverse cell interpolation routine, and have included a copy of our MATLAB code in Appendix A.

Unfortunately, the new matrix LMS algorithm is preventing our existing MATLAB code from operating at real-time speeds. This is not currently a problem, however, since we are only using it to perform offline optimization after the entire log is saved. Even though we don't have C++ code written to implement our optimization routines, we experimented with a modified matrix multiplication algorithm to show that an optimized C++ implementation would offer much greater than real-time performance when necessary. The optimized algorithm works by only multiplying new cells as data is added to the matrix $\mathbf{A}$ instead of performing the entire multiplication operation at every step.

4.2.2 Cell Biasing

The results of the matrix LMS solution can be biased by inserting rows into the matrix $\mathbf{A}$ that say that a point is exactly correct in the original map. These rows force a cell to have

several times as many data points as there are biasing rows before the measured data can overcome the effect of the biasing rows.

Biasing a cell prevents errors due to sparse data points by forcing large mean-square-error (MSE) values as a cell corner moves away from its existing value. Unlike our original thresholding algorithm, we are now able to use sparse data points to perform small updates to cells instead of throwing out the data points. Over time, this algorithm allows gradual optimization of sparse cells without worrying about cell failures in the process. The actual biasing code is including in the inverse cell interpolation code in Appendix A, and the controlling parameters are defined in our main optimization control code included in Appendix B.

4.2.3 Adjusting The Target Air-Fuel Ratio

Another change we made to our original design dealt with the AFR table that we were using. As we ran the table on the car each time, we found that although the car matched the desired value, the car was not performing in the desired way. In some places, more fuel was needed to keep the car stable at both low and high ends. Using information on the toyotaperformance.com site [4], we were able to update our fuel map to perform more to the specifications we wanted. Several of the steps we went through with table updates can be seen in our optimization parameter files included in Appendix D. With our final table, we felt that we were getting the power output in the places we needed it and maintain the fuel economy in the cruise portion of the maps.

4.2.4 Data Preprocessing Filters

We also changed some of methods of data filtering. To begin with, we were using a simple low pass filter in order to smooth the data over time. This got rid of some of the noise in our data logs, making the processing more reliable and giving us some better results. After our success with this form of filtering, we then tried two other forms of filters for the raw log. One type of filter we tried was a simple mean filter over time. This helps to remove noise by decreasing the peaks and smoothing the function. It also results in a rounding of corners in the data over time. The last type of filter we decided to try was a simple morphological filter, the median filter. Like all morphological filters, the median

filter can be hard to model its response. In some cases there is a possibility of actually making the data worse from the use of the filter. In our experience, though, the noise has been centered enough to give good results using this type of non-linear filter. All three of these types of filters can be used on the data through the adjustment of the main control code in Appendix B. Through experimentation we have found that the median filter seems to yield the best results. Its ability to remove outliers in the data points is valuable as in all data processing, and this is the method we currently use.

4.3 ESTIMATING EXPECTED PERFORMANCE

As with any system, some verification of results had to be done in the software before implementing it in hardware. For this we used basic knowledge of how the fuel table affects output along with principles of minimizing MSE.

4.3.1 Estimating Gas Mileage Improvements

When modeling our system, we had to have a way of understanding what sort of response we would get from our new AFR. Using the information on the theoretical AFR, we could see what to expect in our fuel efficiency and power output based in Table 1 below. Another source we used for the same information can be seen in Figure 6.

Table 1. Key AFR numbers [4]

AFR	Comment
12.2	Safe Best Power
14.6	Stochimetric
15.5	Lean Cruise
16.5	Best Fuel Economy

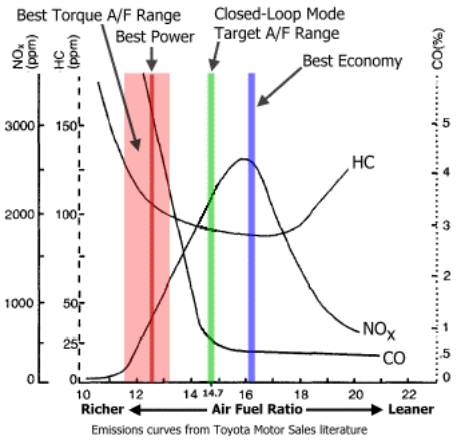

Figure 6. Engine Response vs. Air-Fuel Ratio

With the use of these two resources, we increased the AFR in areas of the graph where we wanted to get more gas mileage out of the car. These regions of our table were those cells which we were mainly driving in for cruise. Similarly, in places that we needed more power, such as acceleration regions of the table, we used and AFR closer to 12.2. Manipulation of these tables with experience and time allowed us to fine tune our car's performance and based on these and driving behavior observed in the logs we saw what we believed would be an increase in gas mileage while maintaining power. Although we did not get the gas mileage we had hoped for, this is most likely due to us needing a lower AFR in the idle position than we had planned, meaning that we burned more fuel than in the original model.

4.3.2 Error In The Measured Air-Fuel Ratio

The other method we used for modeling our performance was our MSE from the target air fuel ratio. By monitoring how far our new theoretical fuel map would perform from the target AFR we wanted, we could get an idea of the increase in performance we were getting from our car. The lower the MSE, the closer our code was finding an ideal solution. A relatively simple calculation could give us an idea of our performance, and can be seen in Appendix D. This was useful for a quick check to see if our modified code was actually outperforming our previous design.

5.0 TESTING AND EVALUATION

While our project had several concrete goals, only one of them was easily evaluated in an objective manner. While fuel consumption was relatively easy to test, the power output and drivability of the car proved more difficult.

5.1 GAS MILEAGE

It was fairly simple to track the gas mileage of the engine through the course of our project. As shown in Figure 7 below, you can see the gas mileage we got out of the car at each fuel stop.

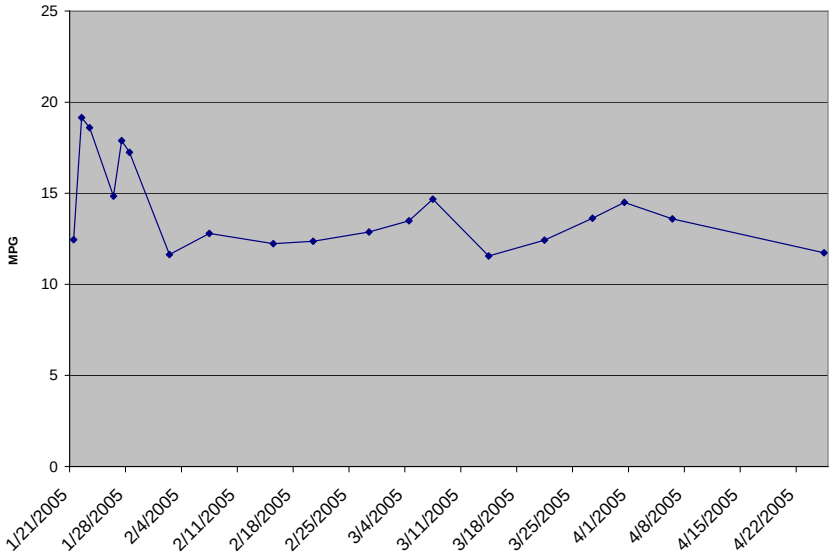

Figure 7. Gas Mileage Over The Course Of The Project

The beginning of the data represents a weekend where the car was driven back and forth to Houston several times. During this highway driving portion the gas mileage was naturally much higher. Other than during this time, the trend shows little change in the fuel consumption through our project.

There are a number of reasons that this could happen. One is simply the driving conditions. While working on our project it was necessary to drive the car aggressively at many times in order to try and get data in other cells of the fuel map. This of course could

result in an increase in fuel usage. Another factor is the idle position. While doing our project we realized that more fuel was needed at idle. Since the vehicle spends much of the time city driving in idle, this could also increase the fuel consumption. Another possibility is the weather conditions. When it is warmer it is harder for the engine to stay cool, and thus it must run the fan more aggressively. Although this effect is slight, it could also add to the amount of fuel the car is using.

5.2 DRIVABILITY

Another part of the goals of our project was the drivability of the car. This is a very subjective manner, and we decided to evaluate the drivability in the following areas:

- Idle stability – does the idle remain stable or does the car feel like its going to die
- Smell of exhaust – since it is an older car the exhaust can smell, does it, and to what level
- Smoothness of ride – how smooth does the car drive
- Engine response – how well does the engine accelerate and handle changes in its demands

In each of the above areas, the car performed better. Prior to the project, it often felt like the car was going to die in idle. The modifications made by our project fixed this idle running level so it was more stable. Also, the project greatly reduced the smell of the exhaust of the car in all areas. It also improved the general smoothness of the ride. Finally, the engine response was improved. Before the project, the engine would sometimes die upon turning on the AC at idle. This problem was solved and resulted in a more reliable drive. Overall, based on our subjective measure, the drivability increased.

5.3 POWER

This part of our project proved very hard to test due to time and cost demands. Putting a car on a dynamometer is the best method for testing the power. These, however, require you to find a shop willing to let you use theirs. Many of these shops would only let you use their equipment if you either paid them for the time or used one of their tuning

services. Due to time constraints, we were unable to get this time donated, and felt that the best option was to leave off this test. In addition, for much of the project the car was Sam's only means of transportation. Leaving the car in the shop would have made it very difficult for him to get around to various appointments.

6.0 TIME AND COST CONSIDERATIONS

Our project was unable to be completed in its entirety under the original time and cost constraints. Originally we projected that our project would be using only equipment that we had available to us. After some work on the fuel map, we determined that we needed a wideband O¬2 sensor to properly monitor performance and optimize the table. This wideband sensor was very pricy at $600, but we felt that this increase in cost could not be avoided. Also, due to this unexpected along with the time crunch and unresponsive nature of the shop we were working with, we felt that we must drop the spark map portion of our project. With some of the complications we met in log alignment with the O2 sensor, we felt that it would be unwise to undertake the tuning of the spark map.

7.0 SAFETY AND ETHICAL ASPECTS OF DESIGN

In addition to the time and cost considerations, there were several other aspects that limited our design. New vehicles are not allowed to use a lean-on-cruise condition [3]. In testing our code, we were able to test more scenarios by using an older emission exempt vehicle. We did, however, reduce the smell of the engine which usually translates to the vehicular emissions. If we are to port our project to another car we would be sure to avoid the lean-on-cruise conditions unless they had been approved prior during factory testing by the EPA. Also, we made sure to follow several safety guidelines during the testing of our vehicles. In accordance with Texas State Law §545.413, safety belts were worn at all times while testing the vehicular modifications. Because we did not feel that we had a safe effective way to test the power of the vehicle, we refrained from performing this test without the proper equipment. The product itself also helps increase the public interest. By improving the drivability of the car, the driver is able to focus on the road, making the road a safer place to drive.

8.0 CONCLUSIONS AND RECOMMENDATIONS

In this project we have learned many things. Through the use of matrix arithmetic, we found that we were able to achieve an MSE which was much lower than we had expected. Despite this result, we were unable to realize the improved fuel economy that we expected. Due to budget and time constraints, we were forced to leave out an entire sector of our proposed project. We found that although we did not accomplish our goal of gas mileage increase, we were very pleased with the overall drivability of the car. In this manner, the code we implemented exceeded our expectations, making the car much more usable in a variety of driving conditions. We also learned that sometimes the results can be correct even though they do not follow your expectations. Such is the case in our modifier results, which defied our initial physics models, only to result in an increase in the performance of our code. In order to determine how well we met the power requirements for our desired result, we would need to perform more extensive testing using equipment that we did not have access to in the course of the project.

We also have several recommendations for the continuing study in the area of our project. The most important aspect we did not cover was our spark map adjustment. Although we did not have the budget or the time to undertake this portion of our project, the TI whitepaper we read on the subject presented an excellent starting point [2]. This would be a good place to start on this subject and could easily be viewed as a project on its own. We suggest that this portion of our original idea be given further consideration and research in modifying what was done in that paper into an offline model could be a valuable tuning aid for at least a starting tune on a car. Further improvement in gas mileage and power might also be achieved by varying the target AFR in our project. If these modifications could be made, I feel that the automated tuning we have begun could be a real benefit to society, reducing cost of turning an often unthought-of portion of the car’s engine.

REFERENCES

[1] "CSS Tuning Fuel Injection," http://www.ccstuning.com/customtuning.htm (current 4 May 2005).

[2] T.G. Horner, "Engine knock detection using spectral analysis techniques with a TMS320 DSP," http://focus.ti.com/lit/an/spra039/spra039.pdf (current 4 May 2005).

[3] "Control of Air Pollution From New Motor Vehicles and New Motor Vehicle Engines: State Commitments to National Low Emission Vehicle Program," http://www.epa.gov/fedrgstr/EPA-AIR/1997/August/Day-22/a21138.htm (current 4 May 2005).

[4] "Fuel Metering," http://www.toyotaperformance.com/fuel_metering.htm (current 4 May 2005).

[5] J. Lucius, "A'PEXi S-AFCII Adjustment in the Mitsubishi 3000GT VR4 & Dodge Stealth TT," http://www.stealth316.com/2-safcii-adjust.htm (current 4 May 2005).

[6] "M-400/M-500 Wideband Oxygen Sensor Controller Product Information," http://www.plxdevices.com/M-Series_logging_productinfo.htm (current 4 May 2005).

APPENDIX A. INVERSE CELL INTERPOLATION CODE

invInterp1Lin.m

```
 1. function [nmap, trust] = invInterp1Lin( Xi, Yi, Xt, P )
 2. % [nmap, trust] = invInterp1Lin( Xi, Yi, Xt, P )
 3. %
 4. % Created: 11 March 2005
 5. % Updated: 11 March 2005
 6. %
 7. % This file & its contents Copyright © 2005 Sam Harwell unless otherwise stated
 8.
 9. %%%%%
10. % NOTES
11. %
12. % Xt must be a monotonic 1-D vector
13. % every Xi must be within [min(Xt) max(Xt)]
14. % every Yi must be within [min(Yt) max(Yt)]
15.
16. if strcmp(P.method,'LMS')==1
17.     %L = zeros(length(Xi),length(Xt));
18.     %R = zeros(length(Xi),length(Xt));
19.     D = zeros(length(Xi),length(Xt));
20.     for x=1:length(Xi)
21.         % SCH 21 Apr 2005: index values
22.         nlx = max( Xt(Xt<=Xi(x)) );
23.         nhx = min( Xt(Xt>=Xi(x)) );
24.         if length(nlx)==1 && length(nhx)==1
25.             % SCH 21 Apr 2005: PERCENTAGES
26.             mrange = nhx-nlx;
27.             % SCH 21 Apr 2005: left
28.             if nhx==nlx
29.                 pl = 1;
30.             else
31.                 pl = (nhx-Xi(x))./mrange;
32.             end
33.             if pl>0
34.                 A = find( Xt(:)==nlx );
35.                 if length(A)>0
36.                     D(x,A) = D(x,A) + pl;
37.                 end
38.             end
39.             % SCH 21 Apr 2005: right
40.             if nhx==nlx
41.                 pr = 1;
42.             else
43.                 pr = (Xi(x)-nlx)./mrange;
44.             end
45.             if pr>0
46.                 A = find( Xt(:)==nhx );
47.                 if length(A)>0
48.                     D(x,A) = D(x,A) + pr;
49.                 end
50.             end
51.         else
52.             Yi(x)=0;
53.         end
54.
55.         %L(x,1:(length(Xt)-1)) = ((Xt(2:length(Xt))-Xi(x))./(Xt(2:length(Xt))-
    Xt(1:(length(Xt)-1))))';
56.         %R(x,2:length(Xt)) = 1-L(x,1:(length(Xt)-1));
57.     end
58.     %mask = L>0 & L<=1;
59.     %L(mask==0) = 0;
60.     %mask = (R>0 & R<=1);
61.     %R(mask==0) = 0;
62.     %D = L + R;
63.     T = sum(D,2) .* Yi(:);
64.     % SCH 26 Apr 2005: tack on the bias if requested
65.     if P.biased==1
66.         dhead = P.priorweight .* eye(length(Xt));
```

```
        thead = P.priorweight .* P.OldYi(:);
        D = [dhead;D];
        T = [thead;T];
    end
    %D = sparse(D);
    %m = lscov(D,T);
    G = sparse(D);
    %[m,flag] = lsqr(D,T,[],[],[],[],P.OldYi(:));
    %m = (D' * D)\(D' * T);
    E = G' * G;
    F = G' * T;
    m = E\F;
    nmap = m;
    trust.method = 'LMS';
    return;
end

if strcmp(P.method,'griddata')==1
    nmap = griddatan( Xi, Yi, Xt );
    trust.method = 'griddata';
    return;
end

% cells lie between points, so subtract 1 from each dimension
ncell = zeros( length(Xt)-1, 1 );
% these are the solved cells. 4 corners for each cell.
icell = zeros( length(Xt)-1, 2 );

for x=1:(length(Xt)-1)
    % this cell goes from Xi(x) to Xi(x+1), Yi(y) to Yi(y+1)
    xmin = min(Xt(x),Xt(x+1));
    xmax = max(Xt(x),Xt(x+1));
    xdir = Xt(x)<Xt(x+1);
    cellmask = (Xi>=xmin) & (Xi<=xmax);
    %goodcells = find(Xi>=xmin & Xi<=xmax);
    if sum(cellmask)>0
    %if length(goodcells)>0
        % extract all the data inside this cell
        %xptsincell = Xi(goodcells);
        %yptsincell = Yi(goodcells);
        xptsincell = Xi(cellmask~=0,:);
        yptsincell = Yi(cellmask~=0,:);
        % note how many points are in this cell
        ncell(x) = length(xptsincell(:));
        % if there are any data point in the cell, solve the cell
        if ncell(x)>5
            h = invInterpSegment( xptsincell, yptsincell, [Xt(x) Xt(x+1)] );
            if h(1)~=-1
                icell(x,1) = h(1);
                icell(x,2) = h(2);
            else
                icell(x,1) = 0;
                icell(x,2) = 0;
            end
%             h = invInterpSegment( xptsincell, yptsincell, [xmin xmax] );
%             if xdir==1
%                 icell(x,1) = h(1);
%                 icell(x,2) = h(2);
%             else % xrev
%                 icell(x,1) = h(2);
%                 icell(x,2) = h(1);
%             end
        else
            icell(x,1) = 0;
            icell(x,2) = 0;
        end
    else
        icell(x,1) = 0;
        icell(x,2) = 0;
    end
end
```

```
138.
139.  % icell(:,1)
140.  % icell(:,2)
141.  % figure;
142.  % scatter( Xt(1:length(Xt-1)), icell(:,1), 'r' );
143.  % hold on;
144.  % scatter( Xt(2:length(Xt)), icell(:,2), 'b' );
145.  % pause;
146.
147.  % new merge code
148.  trust.method = 'Threshold';
149.  nmap = zeros( length(Xt), 1 );
150.  trust.cells = zeros( length(Xt), 1 );
151.  trust.ncell = ncell;
152.  switch P.merge
153.      case {'weightedcells'}
154.          % weighted segment merge by Sam Harwell, 11 March 2005
155.          for x=1:length(Xt)
156.              S = 0;
157.              t = 0;
158.              % SCH 11 Mar 2005: left
159.              if x-1>0 && ncell(x-1)>P.threshold && icell(x,2)>0
160.                  xmin = min(Xt(x-1),Xt(x));
161.                  xmax = max(Xt(x-1),Xt(x));
162.                  cellmask = (Xi>=xmin) & (Xi<=xmax);
163.                  xptsincell = Xi(cellmask~=0,:);
164.                  xp = (xptsincell-Xt(x-1))/(Xt(x)-Xt(x-1));
165.                  Aleft = exp(-4*xp.^2);
166.                  S = S + sum(Aleft);
167.                  t = t + sum(Aleft)*icell(x-1,2);
168.  %                 S = S + ncell(x-1);
169.  %                 t = t + ncell(x-1)*icell(x-1,2);
170.              end
171.              % SCH 11 Mar 2005: right
172.              if x<length(Xt) && ncell(x)>P.threshold && icell(x,1)>0
173.                  xmin = min(Xt(x),Xt(x+1));
174.                  xmax = max(Xt(x),Xt(x+1));
175.                  cellmask = (Xi>=xmin) & (Xi<=xmax);
176.                  xptsincell = Xi(cellmask~=0,:);
177.                  xp = (Xt(x+1)-xptsincell)/(Xt(x+1)-Xt(x));
178.                  Aright = exp(-4*xp.^2);
179.                  S = S + sum(Aright);
180.                  t = t + sum(Aright)*icell(x,1);
181.  %                 S = S + ncell(x);
182.  %                 t = t + ncell(x)*icell(x,1);
183.              end
184.              % SCH 11 Mar 2005: merge
185.              if S>P.threshold
186.                  nmap(x) = t/S;
187.                  trust.cells(x) = S - P.threshold;
188.              end
189.          end
190.      otherwise
191.  end
192.
193.
194.  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
195.  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
196.  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
197.  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
198.  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
199.  function [h] = invInterpSegment( Xi, Yi, c )
200.
201.  % SCH 11 Mar 2005: this should only need to know the g axis endpoint values, the
202.  % g axis values of the data points, and the target modifier value at those
203.  % points.
204.
205.  % warning off MATLAB:polyfit:RepeatedPointsOrRescale
206.  % p = polyfit( Xi, Yi, 1 );
207.  % warning on MATLAB:polyfit:RepeatedPointsOrRescale
208.  % h(1) = p(1)*c(1)+p(2);
```

```
% h(2) = p(1)*c(2)+p(2);
%
% return;

Sy = sum(Yi(:));
Sxx = sum(Xi(:) .* Xi(:));
Sx = sum(Xi(:));
Sxy = sum(Xi(:) .* Yi(:));
S = length(Xi(:));

if (S*Sxx-Sx^2)==0 | (Sxx)==0
    h = [-1;-1];
    return;
end

b = (Sy*Sxx - Sx*Sxy) / (S*Sxx-Sx^2);
m = (Sxy-b*Sx)/(Sxx);

h(1) = m*c(1)+b;
h(2) = m*c(2)+b;

return;

% c holds the coordinates [g1 g2]
g1 = c(1);
g2 = c(2);
Zii1 = zeros(size(zi));
Zii2 = zeros(size(zi));
S1 = 0;
S2 = 0;
S3 = 0;
S4 = 0;
S5 = 0;
S6 = 0;

Zii1 = zi .* zi .* (g2-ag) ./ (g2-g1);
Zii2 = zi .* zi .* (ag-g1) ./ (g2-g1);
S1 = sum( Zii1 .* (g2-ag) ./ (g2-g1) );
S2 = sum( Zii1 .* (ag-g1) ./ (g2-g1) );
S3 = sum( Zii2 .* (g2-ag) ./ (g2-g1) );
S4 = sum( Zii2 .* (ag-g1) ./ (g2-g1) );
S5 = sum( zi .* at .* (g2-ag) ./ (g2-g1) );
S6 = sum( zi .* at .* (ag-g1) ./ (g2-g1) );

K = S6 - S5;

h1 = (S5*S4 - K*S2 - S5*S2) / (S1*S4 - S2*S3);
h2 = (h1*S1 - h1*S3 + K) / (S4 - S2);

h = zeros(2, 1);
h(1) = h1;
h(2) = h2;

return;
```

invInterp2Lin.m

```
 1. function [nmap, trust] = invInterp2Lin( Xi, Yi, Zi, Xt, Yt, P )
 2. % [nmap, trust] = invInterp2Lin( Xi, Yi, Zi, Xt, Yt, P )
 3. %
 4. % Created: 8 March 2005
 5. % Updated: 15 March 2005
 6. %
 7. % This file & its contents Copyright © 2005 Sam Harwell unless otherwise stated
 8.
 9. % The current algorithm computes each cell individually and then merges the
10. % corners.
11.
12. %%%%%
13. % NOTES
14. %
15. % Xt and Yt must be monotonic 1-D vectors
16. % every Xi must be within [min(Xt) max(Xt)]
17. % every Yi must be within [min(Yt) max(Yt)]
18.
19. if strcmp( P.method, 'LMS' )==1
20.     D = zeros(length(Xi), length(Xt(:)));
21.     for x=1:length(Xi)
22.         % SCH 21 Apr 2005: index values
23.         nlx = max( Xt(Xt<=Xi(x)) );
24.         nhx = min( Xt(Xt>=Xi(x)) );
25.         nly = max( Yt(Yt<=Yi(x)) );
26.         nhy = min( Yt(Yt>=Yi(x)) );
27.         if (length(nlx)==1 && length(nhx)==1) && (length(nly)==1 && length(nhy)==1)
28.             % SCH 21 Apr 2005: PERCENTAGES
29.             if nhy==nly
30.                 mrange = (nhx-nlx);
31.             elseif nhx==nlx
32.                 mrange = (nhy-nly);
33.             else
34.                 mrange = (nhx-nlx)*(nhy-nly);
35.             end
36.             % SCH 21 Apr 2005: top left
37.             if nhy==nly
38.                 ptl = 2.*(nhx-Xi(x))./mrange;
39.             elseif nhx==nlx
40.                 ptl = 2.*(Yi(x)-nly)./mrange;
41.             else
42.                 ptl = (nhx-Xi(x)).*(Yi(x)-nly)./mrange;
43.             end
44.             if ptl>0
45.                 %A = find( Xt(:)==nlx );
46.                 %B = find( Yt(:)==nhy );
47.                 %c = intersect(A,B);
48.                 %if length(c)==1
49.                 %    D(x,c(1)) = D(x,c(1)) + ptl;
50.                 %end
51.                 A = find( Xt(1,:)==nlx );
52.                 B = find( Yt(:,1)==nhy );
53.                 c = 16*(A(1)-1)+B(1);
54.                 if length(A)>0 && length(B)>0
55.                     D(x,c) = D(x,c) + ptl;
56.                 end
57.             end
58.             % SCH 21 Apr 2005: top right
59.             if nhy==nly
60.                 ptr = 2.*(Xi(x)-nlx)./mrange;
61.             elseif nhx==nlx
62.                 ptr = 2.*(Yi(x)-nly)./mrange;
63.             else
64.                 ptr = (Xi(x)-nlx).*(Yi(x)-nly)./mrange;
65.             end
66.             if ptr>0
67.                 %A = find( Xt(:)==nhx );
```

```
                %B = find( Yt(:)==nhy );
                %c = intersect(A,B);
                %if length(c)==1
                %    D(x,c(1)) = D(x,c(1)) + ptr;
                %end
                A = find( Xt(1,:)==nhx );
                B = find( Yt(:,1)==nhy );
                c = 16*(A(1)-1)+B(1);
                if length(A)>0 && length(B)>0
                    D(x,c) = D(x,c) + ptr;
                end
            end
            % SCH 21 Apr 2005: bottom left
            if nhy==nly
                pbl = 2.*(nhx-Xi(x))./mrange;
            elseif nhx==nlx
                pbl = 2.*(nhy-Yi(x))./mrange;
            else
                pbl = (nhx-Xi(x)).*(nhy-Yi(x))./mrange;
            end
            if pbl>0
                %A = find( Xt(:)==nlx );
                %B = find( Yt(:)==nly );
                %c = intersect(A,B);
                %if length(c)==1
                %    D(x,c(1)) = D(x,c(1)) + pbl;
                %end
                A = find( Xt(1,:)==nlx );
                B = find( Yt(:,1)==nly );
                c = 16*(A(1)-1)+B(1);
                if length(A)>0 && length(B)>0
                    D(x,c) = D(x,c) + pbl;
                end
            end
            % SCH 21 Apr 2005: bottom right
            if nhy==nly
                pbr = 2.*(Xi(x)-nlx)./mrange;
            elseif nhx==nlx
                pbr = 2.*(nhy-Yi(x))./mrange;
            else
                pbr = (Xi(x)-nlx).*(nhy-Yi(x))./mrange;
            end
            if pbr>0
                %A = find( Xt(:)==nhx );
                %B = find( Yt(:)==nly );
                %c = intersect(A,B);
                %if length(c)==1
                %    D(x,c(1)) = D(x,c(1)) + pbr;
                %end
                A = find( Xt(1,:)==nhx );
                B = find( Yt(:,1)==nly );
                c = 16*(A(1)-1)+B(1);
                if length(A)>0 && length(B)>0
                    D(x,c) = D(x,c) + pbr;
                end
            end
            tper = sum(D(x,:));
            % SCH 21 Apr 2005: tolerance added for floating point rounding errors
            if abs(tper-1)>0.001
                D(x,:)=0;
            end
        else
            Zi(x)=0;
        end
    end
    T = Zi(:);
    T = sum(D,2) .* T;
    if P.biased==1
        dhead = P.priorweight .* eye(length(Xt(:)));
        thead = P.priorweight .* P.OldZi(:);
        D = [dhead;D];
```

```
        T = [thead;T];
    end
    %D = sparse(D);
    %m = lscov(D,T);
    G = sparse(D);
    %[m,flag] = lsqr(D,T,[],[],[],[],P.OldZi(:));
    E = G' * G;
    F = G' * T;
    m = E\F;
    nmap = zeros(size(Xt));
    nmap(:) = m(:);
    trust.method = 'LMS';
    return;
end

if strcmp(P.method,'griddata')==1
    nmap = griddata( Xi, Yi, Zi, Xt, Yt' )';
    trust.method = 'griddata';
    return;
end

% cells lie between points, so subtract 1 from each dimension
ncell = zeros( length(Xt)-1, length(Yt)-1 );
% these are the solved cells. 4 corners for each cell.
icell = zeros( length(Xt)-1, length(Yt)-1, 4 );
trust.used = zeros( length(Xi), 1 );

for x=1:length(Xt)-1
    for y=1:length(Yt)-1
        % this cell goes from Xi(x) to Xi(x+1), Yi(y) to Yi(y+1)
        xmin = min(Xt(x),Xt(x+1));
        xmax = max(Xt(x),Xt(x+1));
        xdir = Xt(x)<Xt(x+1);
        ymin = min(Yt(y),Yt(y+1));
        ymax = max(Yt(y),Yt(y+1));
        ydir = Yt(y)<Yt(y+1);
        cellmask = (Xi>=xmin) & (Xi<=xmax) & (Yi>=ymin) & (Yi<=ymax);

        % SCH 25 Mar 2005: Number of points in this cell
        ncell(x,y) = sum(cellmask);

        icell(x,y,1) = 0;
        icell(x,y,2) = 0;
        icell(x,y,3) = 0;
        icell(x,y,4) = 0;

        if strcmp(P.merge, 'weightedcells')==1 && ncell(x,y)>P.threshold
            % extract all the data inside this cell
            xptsincell = Xi(cellmask~=0,:);
            yptsincell = Yi(cellmask~=0,:);
            zptsincell = Zi(cellmask~=0,:);
            % note how many points are in this cell
            %ncell(x,y) = size(xptsincell,1);

            trust.used = trust.used | cellmask;
        elseif P.interpfill==1 && ncell(x,y)>2 && ncell(x,y)<P.threshold
            % extract all the data inside this cell
            xptsincell = Xi(cellmask~=0,:);
            yptsincell = Yi(cellmask~=0,:);
            zptsincell = Zi(cellmask~=0,:);
            % note how many points are in this cell
            %ncell(x,y) = size(xptsincell,1);

            % SCH 14 Mar 2005: if pchip interpolation is allowed, build the
            % interpolation data point vectors using a combination of measured
            % points and points interpolated between the measurements.

            % SCH 14 Mar 2005: step 1: figure out how many actual points to read
            % before and after a point that falls inside the cell
                % SCH 14 Mar 2005: seems like a reasonable guess for now
                extrapoints = 10;
```

```
210.
211.             % SCH 14 Mar 2005: step 2: filter those points out of the entire
212.             % data set so we don't spend unnecessary (pointless) time upsampling
213.             % the entire set of measured data
214.                 % SCH 14 Mar 2005: make a padded version of cellmask
215.                 pad = zeros( extrapoints, 1 );
216.                 padcellmask = [pad;cellmask;pad];
217.                 for masking=1:(2*extrapoints+1)
218.                     cellmask = cellmask | padcellmask(masking:(masking+length(cellmask)-
     1));
219.                 end
220.                 xptsincell = Xi(cellmask~=0,:);
221.                 yptsincell = Yi(cellmask~=0,:);
222.                 zptsincell = Zi(cellmask~=0,:);
223.
224.             % SCH 14 Mar 2005: step 3: determine an appropriate oversampling
225.             % rate N
226.                 % SCH 14 Mar 2005: ncell(x,y) is the current number of points in the cell
227.                 nsects = ncell(x,y);
228.                 % SCH 14 Mar 2005: we'll go with triple the threshold as a goal
229.                 N = 2*P.threshold / nsects;
230.
231.             % SCH 14 Mar 2005: step 4: pchip interpolate (N-1) points between
232.             % each known point
233.                 ipts = 1:(1/N):length(xptsincell);
234.                 xptsincellgood = interp1(1:length(xptsincell),xptsincell,ipts,'pchip')';
235.                 yptsincellgood = interp1(1:length(yptsincell),yptsincell,ipts,'pchip')';
236.                 zptsincellgood = interp1(1:length(zptsincell),zptsincell,ipts,'pchip')';
237.
238.             % SCH 14 Mar 2005: step 5: filter the interpolated data set to only
239.             % include points that fall inside the cell at hand so those points
240.             % can be used in the ICI below
241.                 cellmask = (xptsincellgood>=xmin) & (xptsincellgood<=xmax) &
     (yptsincellgood>=ymin) & (yptsincellgood<=ymax);
242.                 % extract all the data inside this cell
243.                 xptsincell = xptsincellgood(cellmask~=0,:);
244.                 yptsincell = yptsincellgood(cellmask~=0,:);
245.                 zptsincell = zptsincellgood(cellmask~=0,:);
246.
247.             % SCH 14 Mar 2005: step 6: update the number of points for the cell
248.                 ncell(x,y) = length(xptsincell);
249.         elseif P.windowexpand==1 && ncell(x,y)>2 && ncell(x,y)<P.threshold
250.             % extract all the data inside this cell
251.             xptsincell = Xi(cellmask~=0,:);
252.             yptsincell = Yi(cellmask~=0,:);
253.             zptsincell = Zi(cellmask~=0,:);
254.             % note how many points are in this cell
255.             %ncell(x,y) = size(xptsincell,1);
256.
257.             % SCH 15 Mar 2005: expand the bilinear interpolating window to
258.             % include more points. The larger window might make it work better?
259.
260.             % SCH 15 Mar 2005: step 1: find the center of the cell
261.                 xcenter = (xmax-xmin)/2;
262.                 ycenter = (ymax-ymin)/2;
263.             % SCH 15 Mar 2005: step 2: find the distance from each point to the center
264.                 xdist = Xi-xcenter;
265.                 ydist = Yi-ycenter;
266.                 dist = sqrt(xdist.*xdist + ydist.*ydist);
267.             % SCH 15 Mar 2005: step 3: sort the distances
268.                 [distsort,sortix] = sort(dist);
269.             % SCH 15 Mar 2005: step 4: use the (P.threshold+2) nearest points to calculate
270.             % the corners. I had to add the +2 because just using the threshold didn't
271.             % PASS the threshold which is what is checked later.
272.                 xptsincell = Xi(sortix(1:P.threshold+2),:);
273.                 yptsincell = Yi(sortix(1:P.threshold+2),:);
274.                 zptsincell = Zi(sortix(1:P.threshold+2),:);
275.             % SCH 15 Mar 2005: update the number of points for the cell
276.                 ncell(x,y) = length(xptsincell);
277.         else
278.             if ncell(x,y)<4
```

```
                continue
            end
            % extract all the data inside this cell
            xptsincell = Xi(cellmask~=0,:);
            yptsincell = Yi(cellmask~=0,:);
            zptsincell = Zi(cellmask~=0,:);
            % note how many points are in this cell
            %ncell(x,y) = size(xptsincell,1);
        end
        % if there are any data point in the cell, solve the cell
        if ncell(x,y)>4
            Zt = invInterpCell( xptsincell, yptsincell, zptsincell, [Xt(x) Xt(x+1) Yt(y) Yt(y+1)] );
            icell(x,y,1) = Zt(1);
            icell(x,y,2) = Zt(2);
            icell(x,y,3) = Zt(3);
            icell(x,y,4) = Zt(4);
        end
    end
end

trust.ncell = ncell;
trust.icell = icell;

% SCH 15 Mar 2005: another possibility for WOT? bind the slope of the cells
% along the MAP axis, and use a linear polyfit to model the RPM axis?

% new merge code
trust.method = 'Threshold';
nmap = zeros( length(Xt), length(Yt) );
trust.cells = zeros( length(Xt), length(Yt) );
switch P.merge
    case {'weightedcells'}
%         % weighted cell merge by Will Thomas, 10 March 2005
%         pad=zeros(length(Xt)-1,1);
%         ipad=zeros(length(Xt)-1,1,4);
%         ncelltemp=cat(2,pad,ncell,pad);
%         icelltemp=cat(2,ipad,icell,ipad);
%         pad=zeros(1,length(Yt)+1);
%         ipad=zeros(1,length(Yt)+1,4);
%         ncelltemp=cat(1,pad,ncelltemp,pad);
%         icelltemp=cat(1,ipad,icelltemp,ipad);
%         for x=1:length(Xt)
%             for y=1:length(Yt)
%                 cellgood=zeros(2,2);
%                 cellgood(:,:)=ncelltemp(x:x+1,y:y+1)>P.threshold;
%                 cellgood(:,:)=cellgood.*ncelltemp(x:x+1,y:y+1);
%                 alln=cellgood(1,1)+cellgood(1,2)+cellgood(2,1)+cellgood(2,2);
%                 if alln>0
%                     nmap(x,y)=  cellgood(1,1)*icelltemp(x,y,2) + ...
%                                 cellgood(2,1)*icelltemp(x+1,y,1) + ...
%                                 cellgood(1,2)*icelltemp(x,y+1,4) + ...
%                                 cellgood(2,2)*icelltemp(x+1,y+1,3);
%                     nmap(x,y)=nmap(x,y)/alln;
%                     trust.cells(x,y) = alln-P.threshold;
%                 end
%             end
%         end
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % SCH 13 Mar 2005: for some reason I am thinking this can be done using
        % 4 shifted copies of the original matrix. This cell merge code by
        % Sam Harwell.

        goodweights = (ncell>P.threshold) .* ncell;

        % SCH 13 Mar 2005: we'll pad the x dimension first, then the y dimension
        padx = zeros( size(goodweights(1,:)) );
        pady = zeros( size(nmap(:,1)) );

        % copy 1: unshifted, pad the right (end of x) and top (end of y)
        % this goes with the bottom left corner (3)
```

```
        weightsUnshifted = cat(1, goodweights, padx);
        weightsUnshifted = cat(2, weightsUnshifted, pady);
        goodcorners = icell(:,:,3); % SCH 13 Mar 2005: yep I need to get this
        cornersUnshifted = cat(1, goodcorners, padx);
        cornersUnshifted = cat(2, cornersUnshifted, pady);
        % copy 2: shift right 1, pad the left (beginning of x) and top (end of y)
        % this goes with the bottom right corner (4)
        weightsShiftedRight = cat(1, padx, goodweights);
        weightsShiftedRight = cat(2, weightsShiftedRight, pady);
        goodcorners = icell(:,:,4); % SCH 13 Mar 2005: yep I need to get this
        cornersShiftedRight = cat(1, padx, goodcorners);
        cornersShiftedRight = cat(2, cornersShiftedRight, pady);
        % copy 3: shift up 1, pad the right (end of x) and bottom (beginning of y)
        % this goes with the top left corner (1)
        weightsShiftedUp = cat(1, goodweights, padx);
        weightsShiftedUp = cat(2, pady, weightsShiftedUp);
        goodcorners = icell(:,:,1); % SCH 13 Mar 2005: yep I need to get this
        cornersShiftedUp = cat(1, goodcorners, padx);
        cornersShiftedUp = cat(2, pady, cornersShiftedUp);
        % copy 4: shift up 1, right 1, pad the left (beginning of x) and bottom (beginning
    of y)
        % this goes with the top right corner (2)
        weightsShiftedRightUp = cat(1, padx, goodweights);
        weightsShiftedRightUp = cat(2, pady, weightsShiftedRightUp);
        goodcorners = icell(:,:,2); % SCH 13 Mar 2005: yep I need to get this
        cornersShiftedRightUp = cat(1, padx, goodcorners);
        cornersShiftedRightUp = cat(2, pady, cornersShiftedRightUp);

        cornersums = weightsUnshifted + weightsShiftedRight + ...
                weightsShiftedUp + weightsShiftedRightUp;
        nmap =  weightsUnshifted .* cornersUnshifted + ...
                weightsShiftedRight .* cornersShiftedRight + ...
                weightsShiftedUp .* cornersShiftedUp + ...
                weightsShiftedRightUp .* cornersShiftedRightUp;
        nmap = nmap ./ (cornersums+eps);

        % copy 1: unshifted, pad the right (end of x) and top (end of y)
        % this goes with the bottom left corner (3)
        allWeightsUnshifted = cat(1, ncell, padx);
        allWeightsUnshifted = cat(2, allWeightsUnshifted, pady);
        % copy 2: shift right 1, pad the left (beginning of x) and top (end of y)
        % this goes with the bottom right corner (4)
        allWeightsShiftedRight = cat(1, padx, ncell);
        allWeightsShiftedRight = cat(2, allWeightsShiftedRight, pady);
        % copy 3: shift up 1, pad the right (end of x) and bottom (beginning of y)
        % this goes with the top left corner (1)
        allWeightsShiftedUp = cat(1, ncell, padx);
        allWeightsShiftedUp = cat(2, pady, allWeightsShiftedUp);
        % copy 4: shift up 1, right 1, pad the left (beginning of x) and bottom (beginning
    of y)
        % this goes with the top right corner (2)
        allWeightsShiftedRightUp = cat(1, padx, ncell);
        allWeightsShiftedRightUp = cat(2, pady, allWeightsShiftedRightUp);

        coveragesums = allWeightsUnshifted + allWeightsShiftedRight + ...
                allWeightsShiftedUp + allWeightsShiftedRightUp;

        trust.cells = cornersums;
        trust.coverage = coveragesums;
    otherwise
end


return;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
418.  function Zt = invInterpCell( Xi, Yi, Zi, c )
419.
420.  % c holds the coordinates [xa xb yc ya]
421.  xa = c(1);
422.  xb = c(2);
423.  yc = c(3);
424.  ya = c(4);
425.
426.  S = double(length(Xi));
427.  Sx = sum(Xi);
428.  Sxx = sum(Xi.*Xi);
429.  Sxxy = sum(Xi.*Xi.*Yi);
430.  Sxxyy = sum(Xi.*Xi.*Yi.*Yi);
431.  Sxy = sum(Xi.*Yi);
432.  Sxyy = sum(Xi.*Yi.*Yi);
433.  Sxyz = sum(Xi.*Yi.*Zi);
434.  Sxz = sum(Xi.*Zi);
435.  Sy = sum(Yi);
436.  Syy = sum(Yi.*Yi);
437.  Syz = sum(Yi.*Zi);
438.  Sz = sum(Zi);
439.
440.  A = zeros(4,4);
441.  b = zeros(4,1);
442.
443.  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
444.  %% BEGIN PART FROM C++
445.
446.      A(1,1) = ...
447.          S * xb * xb * yc * yc - 2 * xb * yc * yc * Sx + yc * yc * Sxx - 2 * xb * xb * yc *
      Sy + ...
448.          4 * xb * yc * Sxy - 2 * yc * Sxxy + xb * xb * Syy - 2 * xb * Sxyy + Sxxyy;
449.
450.      A(1,2) = ...
451.          -S * xa * xb * yc * yc + xa * yc * yc * Sx + xb * yc * yc * Sx - yc * yc * Sxx +
      ...
452.          2 * xa * xb * yc * Sy - 2 * xa * yc * Sxy - 2 * xb * yc * Sxy + 2 * yc * Sxxy - ...
453.          xa * xb * Syy + xa * Sxyy + xb * Sxyy - Sxxyy;
454.
455.      A(1,3) = ...
456.          -S * xb * xb * ya * yc + 2 * xb * ya * yc * Sx - ya * yc * Sxx + xb * xb * ya * Sy
      - ...
457.          2 * xb * ya * Sxy + ya * Sxxy + xb * xb * yc * Sy - 2 * xb * yc * Sxy + ...
458.          yc * Sxxy - xb * xb * Syy + 2 * xb * Sxyy - Sxxyy;
459.
460.      A(1,4) = ...
461.          S * xa * xb * ya * yc - xa * ya * yc * Sx - xb * ya * yc * Sx + ...
462.          ya * yc * Sxx - xa * xb * ya * Sy + xa * ya * Sxy + xb * ya * Sxy - ...
463.          ya * Sxxy - xa * xb * yc * Sy + xa * yc * Sxy + xb * yc * Sxy - ...
464.          yc * Sxxy + xa * xb * Syy - xa * Sxyy - xb * Sxyy + Sxxyy;
465.
466.      A(2,1) = ...
467.          -S * xa * xb * yc * yc + xa * yc * yc * Sx + xb * yc * yc * Sx - yc * yc * Sxx +
      ...
468.          2 * xa * xb * yc * Sy - 2 * xa * yc * Sxy - 2 * xb * yc * Sxy + 2 * yc * Sxxy - ...
469.          xa * xb * Syy + xa * Sxyy + xb * Sxyy - Sxxyy;
470.
471.      A(2,2) = ...
472.          S * xa * xa * yc * yc - 2 * xa * yc * yc * Sx + yc * yc * Sxx - 2 * xa * xa * yc *
      Sy + ...
473.          4 * xa * yc * Sxy - 2 * yc * Sxxy + xa * xa * Syy - 2 * xa * Sxyy + Sxxyy;
474.
475.      A(2,3) = ...
476.          S * xa * xb * ya * yc - xa * ya * yc * Sx - xb * ya * yc * Sx + ya * yc * Sxx - ...
477.          xa * xb * ya * Sy + xa * ya * Sxy + xb * ya * Sxy - ya * Sxxy - ...
478.          xa * xb * yc * Sy + xa * yc * Sxy + xb * yc * Sxy - yc * Sxxy + ...
479.          xa * xb * Syy - xa * Sxyy - xb * Sxyy + Sxxyy;
480.
481.      A(2,4) = ...
482.          -S * xa * xa * ya * yc + 2 * xa * ya * yc * Sx - ya * yc * Sxx + xa * xa * ya * Sy
      - ...
```

```
        2 * xa * ya * Sxy + ya * Sxxy + xa * xa * yc * Sy - 2 * xa * yc * Sxy + ...
        yc * Sxxy - xa * xa * Syy + 2 * xa * Sxyy - Sxxyy;

    A(3,1) = ...
        -S * xb * xb * ya * yc + 2 * xb * ya * yc * Sx - ya * yc * Sxx + xb * xb * ya * Sy
- ...
        2 * xb * ya * Sxy + ya * Sxxy + xb * xb * yc * Sy - 2 * xb * yc * Sxy + ...
        yc * Sxxy - xb * xb * Syy + 2 * xb * Sxyy - Sxxyy;

    A(3,2) = ...
        S * xa * xb * ya * yc - xa * ya * yc * Sx - xb * ya * yc * Sx + ya * yc * Sxx - ...
        xa * xb * ya * Sy + xa * ya * Sxy + xb * ya * Sxy - ya * Sxxy - ...
        xa * xb * yc * Sy + xa * yc * Sxy + xb * yc * Sxy - yc * Sxxy + ...
        xa * xb * Syy - xa * Sxyy - xb * Sxyy + Sxxyy;

    A(3,3) = ...
        S * xb * xb * ya * ya - 2 * xb * ya * ya * Sx + ya * ya * Sxx - 2 * xb * xb * ya *
Sy + ...
        4 * xb * ya * Sxy - 2 * ya * Sxxy + xb * xb * Syy - 2 * xb * Sxyy + Sxxyy;

    A(3,4) = ...
        -S * xa * xb * ya * ya + xa * ya * ya * Sx + xb * ya * ya * Sx - ya * ya * Sxx +
...
        2 * xa * xb * ya * Sy - 2 * xa * ya * Sxy - 2 * xb * ya * Sxy + 2 * ya * Sxxy - ...
        xa * xb * Syy + xa * Sxyy + xb * Sxyy - Sxxyy;

    A(4,1) = ...
        S * xa * xb * ya * yc - xa * ya * yc * Sx - xb * ya * yc * Sx + ya * yc * Sxx - ...
        xa * xb * ya * Sy + xa * ya * Sxy + xb * ya * Sxy - ya * Sxxy - ...
        xa * xb * yc * Sy + xa * yc * Sxy + xb * yc * Sxy - yc * Sxxy + ...
        xa * xb * Syy - xa * Sxyy - xb * Sxyy + Sxxyy;

    A(4,2) = ...
        -S * xa * xa * ya * yc + 2 * xa * ya * yc * Sx - ya * yc * Sxx + xa * xa * ya * Sy
- ...
        2 * xa * ya * Sxy + ya * Sxxy + xa * xa * yc * Sy - 2 * xa * yc * Sxy + ...
        yc * Sxxy - xa * xa * Syy + 2 * xa * Sxyy - Sxxyy;

    A(4,3) = ...
        -S * xa * xb * ya * ya + xa * ya * ya * Sx + xb * ya * ya * Sx - ya * ya * Sxx +
...
        2 * xa * xb * ya * Sy - 2 * xa * ya * Sxy - 2 * xb * ya * Sxy + 2 * ya * Sxxy - ...
        xa * xb * Syy + xa * Sxyy + xb * Sxyy - Sxxyy;

    A(4,4) = ...
        S * xa * xa * ya * ya - 2 * xa * ya * ya * Sx + ya * ya * Sxx - 2 * xa * xa * ya *
Sy + ...
        4 * xa * ya * Sxy - 2 * ya * Sxxy + xa * xa * Syy - 2 * xa * Sxyy + Sxxyy;

    b(1,1) = ...
        ( xa * yc - xa * ya + xb * ya - xb * yc ) * ...
        ( xb * Syz + yc * Sxz - Sxyz - xb * yc * Sz );

    b(2,1) = ...
        ( xa * yc - xa * ya + xb * ya - xb * yc ) * ...
        ( xa * yc * Sz - xa * Syz - yc * Sxz + Sxyz );

    b(3,1) = ...
        ( xa * yc - xa * ya + xb * ya - xb * yc ) * ...
        ( xb * ya * Sz + Sxyz - xb * Syz - ya * Sxz );

    b(4,1) = ...
        ( xa * yc - xa * ya + xb * ya - xb * yc ) * ...
        ( ya * Sxz + xa * Syz - xa * ya * Sz - Sxyz );

%% END PART FROM C++
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Zt = A^-1 * b;
```

APPENDIX B. DATA ALIGNMENT CODE

datalign.m

```
 1. function [An, Bn] = datalign( n, A, B, p )
 2. % [An, Bn] = datalign( n, A, B )
 3. %
 4. %   Align data in time from two logs that have different sampling rates. To make
 5. %   use of this function, you need to goose the throttle somewhere near the
 6. %   beginning of the log period after both measurement systems are started. This
 7. %   sets an initial timing mark. Right before the car is shut off, goose the
 8. %   throttle one last time to set the out-point, then turn off both logging
 9. %   systems.
10. %
11. %   PARAMETERS:
12. %       n   Number of points to interpolate for the output
13. %       A   Raw data A, RPM in first column
14. %       B   Raw data B, RPM in first column
15. %
16. %   RETURNS:
17. %       An  Vector of n time-aligned points from A from the region where A and B overlap
18. %       Bn  Vector of n time-aligned points from B from the region where A and B overlap
19. %
20. % Created: 6 March 2005
21. % Updated: 24 March 2005
22. %
23. % This file & its contents Copyright © 2005 Sam Harwell unless otherwise stated
24.
25. An = zeros(n,size(A,2));
26. Bn = zeros(n,size(B,2));
27.
28. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
29. % 1) Find the first and last place where RPM is above the threshold
30. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
31. thresAbegin = find(A(:,1)>p.thresholdbegin);
32. thresAend = find(A(:,1)>p.thresholdend);
33. startA = thresAbegin(1);
34. endA = thresAend(length(thresAend));
35.
36. thresBbegin = find(B(:,1)>p.thresholdbegin);
37. thresBend = find(B(:,1)>p.thresholdend);
38. startB = thresBbegin(1);
39. endB = thresBend(length(thresBend));
40.
41. switch p.alignmethod
42. case {'endpeaks'}
43.     % SCH 15 Mar 2005: simplest and fastest, miserable accuracy on logs that
44.     % don't have a constant sampling period.
45.
46.     % SCH 15 Mar 2005: get the aligned points assuming constant sample periods
47.     samplePointsA = startA:((endA-startA)/(n-1)):endA;
48.     samplePointsB = startB:((endB-startB)/(n-1)):endB;
49. case {'xcorr'}
50.     % SCH 15 Mar 2005: we'll call this method a.
51.     %   1. Averaging filter
52.     %   2. Get the scaled window, with the scaling centered around point in
53.     %      question (interp1 cubic again)
54.     %   3. Correlation of a window around the point
55.     %   4. Time offset by location of the peak in the correlation
56.
57.     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
58.     % 2a) Averaging filter
59.     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
60.     % SCH 15 Mar 2005: the parameter is the 1-sided width
61.     w = p.avgfiltwidth;
62.     % SCH 15 Mar 2005: these will be the vectors after averaging
63.     filt = [0.0015888 0.0048979 0.0044396 -0.0099058 -0.039364 ...
64.             -0.060869 -0.033515 0.06545 0.20492 0.30787 0.30787 ...
65.             0.20492 0.06545 -0.033515 -0.060869 -0.039364 -0.0099058 ...
66.             0.0044396 0.0048979 0.0015888];
67.
```

```
68.      avgFiltA = A(:,1);
69.      avgFiltB = B(:,1);
70.      %avgFiltA = avgFilt( A(:,1), w );
71.      %avgFiltB = avgFilt( B(:,1), w );
72.
73.      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
74.      % 3a) Scaling
75.      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
76.      % SCH 15 Mar 2005: get the aligned points assuming constant sample periods
77.      samplePointsA = startA:((endA-startA)/(n-1)):endA;
78.      samplePointsB = startB:((endB-startB)/(n-1)):endB;
79.
80.      sampledA = interp1( 1:size(avgFiltA,1), avgFiltA, samplePointsA, 'cubic' );
81.      sampledB = interp1( 1:size(avgFiltB,1), avgFiltB, samplePointsB, 'cubic' );
82.
83.      filt = myfilter();
84.      sampledA = filter( filt, sampledA );
85.      sampledB = filter( filt, sampledB );
86.      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
87.      % 4a) Correlation
88.      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
89.      % SCH 15 Mar 2005: Find the offset that maximizes the correlation between
90.      % the two vectors. We could do this at every point, but it would be too
91.      % computationally expensive and would like give a result that isn't
92.      % perfectly monotonic. Instead, we'll test at offsets and use interp1 to
93.      % estimate the sample offset in the gaps.
94.      CorrN = 50; % SCH 15 Mar 2005: steps between actual correlation computations
95.      CorrW = 401; % SCH 15 Mar 2005: correlation window
96.      CorrS = 800; % SCH 15 Mar 2005: max offset correction
97.      CorrM = 'cubic'; % SCH 15 Mar 2005: interpolation method for finding offsets
98.                   % between correlation points. cubic (or pchip) maintains the
99.                   % monotonic nature of the data which is important since we know
100.                  % the points are in the correct order chronologically, just
101.                  % unequally spaced.
102.     offsets = findOffsetsCorr( sampledB, sampledA, CorrW, CorrS, CorrN, CorrM );
103.
104.     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
105.     % 5a) Adjustments
106.     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
107.     % SCH 15 Mar 2005: These points don't work as sample points because they are
108.     % not spaced equally in time due to the inconsistent sampling period in A.
109.     desiredPeriod = ((endA-startA)/(n-1));
110.     desiredPointsA = startA:desiredPeriod:endA;
111.     % SCH 15 Mar 2005: if the offset offsets(x) is y, then the actual sample
112.     % point is (desiredPointsA(x)+y*desiredPeriod)
113.     samplePointsA = desiredPointsA + offsets .* desiredPeriod;
114.     if issorted(samplePointsA)~=1
115.         fprintf(1, 'WARNING: SAMPLE POINTS ARE NOT MONOTONIC!!!\n');
116.     end
117. case {'peakfilter'}
118.     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
119.     % 2b) Expand to the endpoints
120.     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
121.     scaleA = (endA-startA)/(endB-startB)
122.     scaleB = 1/scaleA;
123.
124.     % did A start first?
125.     if (startA-1)/scaleA>startB-1 % yes
126.         startA = startA - (startB-1)*scaleA;
127.         startB = 1;
128.     else % no B started first
129.         startB = startB - (startA-1)*scaleB;
130.         startA = 1;
131.     end
132.
133.     % did A end first?
134.     if (size(A,1)-endA)/scaleA <= (size(B,1)-endB) % yes
135.         endB = (size(A,1)-endA)*scaleB + endB;
136.         endA = size(A,1);
137.     else % no B ended first
138.         endA = (size(B,1)-endB)*scaleA + endA;
```

```
139.          endB = size(B,1);
140.      end
141.
142.      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
143.      % 3b) Variable time correction parameters
144.      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
145.      %     % SCH 15 Mar 2005: peak search window
146.      %     searchwindowA = 20;
147.      %     searchwindowB = 20;
148.
149.      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
150.      % 4b) Find peaks
151.      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
152.      % SCH 15 Mar 2005: Vector A first
153.          % SCH 15 Mar 2005: Pick appropriate parameters. For now this is just guessing.
154.          searchwindow = 6;
155.          minoffset = 400; % SCH 15 Mar 2005: RPM
156.          minwidth = 5;
157.          peaksA = findPeaks( A(:,1), searchwindow, minoffset, minwidth );
158.
159.      % SCH 15 Mar 2005: Vector B second
160.          % SCH 15 Mar 2005: Pick appropriate parameters. For now this is just guessing.
161.          searchwindow = 6;
162.          minoffset = 400; % SCH 15 Mar 2005: RPM
163.          minwidth = 5;
164.          peaksB = findPeaks( B(:,1), searchwindow, minoffset, minwidth );
165.
166.      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
167.      % 5b) Find the time correction function
168.      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
169.      % SCH 15 Mar 2005: this isn't implemented yet and might not be if the xcorr
170.      % version works well (which I'm sure it will)
171.
172.      % SCH 15 Mar 2005: The actual index in A
173.      knownTimeIdxA = 1:max(size(A,1),size(B,1));
174.      % SCH 15 Mar 2005: would really fall at this index if A were linear in time.
175.      knownTimeTrueA = 1:max(size(A,1),size(B,1));
176.
177.      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
178.      % 6b) Correct the sampling vector
179.      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
180.      % SCH 15 Mar 2005: We want to take samples at evenly spaced points in time from
181.      % the vector A. Unfortunately, A is not linear in time, so we have to take these
182.      % points in time and correct them for the non-linear set by interpolating
183.      % between "known" points in time. The quotes are because the "known" points are
184.      % just best guesses from earlier in the function, not from a timestamp.
185.
186.      % SCH 15 Mar 2005: We are assuming B is linear in time. This is because in the
187.      % present application of this function, it is. In the end, this should be a safe
188.      % assumption, because the best we can do anyway is find the relative time shifts
189.      % between A and B. Without an absolute time scale for one of the vectors, it's
190.      % all we can do.
191.
192.      % SCH 15 Mar 2005: These assume A and B are linear in time (constant sampling
193.      % period.
194.      sampleWantedA = startA:((endA-startA)/(n-1)):endA;
195.      sampleWantedB = startB:((endB-startB)/(n-1)):endB;
196.
197.      samplePointsA = interp1( knownTimeIdxA, knownTimeTrueA, sampleWantedA, 'cubic');
198.      samplePointsB = sampleWantedB;
199.  otherwise % SCH 15 Mar 2005: same as 'endpeaks'
200.      % SCH 15 Mar 2005: simplest and fastest, miserable accuracy on logs that
201.      % don't have a constant sampling period.
202.
203.      % SCH 15 Mar 2005: get the aligned points assuming constant sample periods
204.      samplePointsA = startA:((endA-startA)/(n-1)):endA;
205.      samplePointsB = startB:((endB-startB)/(n-1)):endB;
206.  end
207.
208.  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
209.  % Final sampling
```

```
210.  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
211.  % SCH 15 Mar 2005: Resample the original input vector at points that really
212.  % are equally spaced in time (based on the correction method above).
213.
214.  % SCH 15 Mar 2005: change from spline interpolate to cubic. Spline can have
215.  % overshoot and more oscillations whereas cubic (pchip) maintains the monotonic
216.  % nature of sections between data points.
217.
218.  for x=1:size(A,2)
219.      An(:,x) = interp1( 1:size(A,1), A(:,x), samplePointsA, 'cubic' );
220.  end
221.
222.  for x=1:size(B,2)
223.      Bn(:,x) = interp1( 1:size(B,1), B(:,x), samplePointsB, 'cubic' );
224.  end
225.
226.  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
227.  % Just for plotting function output
228.  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
229.  if 1 % SCH 22 Mar 2005: turn me off to plot datalign alignment results
230.      return;
231.  end
232.
233.  f = figure;
234.  set(f,'name','RPM comparison into datalign().');
235.
236.  splotx = 2;
237.  sploty = 1;
238.
239.  subplot(splotx,sploty,1);
240.  plot(A(1:500,1),'b');
241.  title('Holley Log');
242.
243.  subplot(splotx,sploty,2);
244.  plot(B(1:500,1),'r');
245.  title('Wideband Log');
246.
247.  f = figure;
248.  set(f,'name','RPM comparison out of datalign().');
249.
250.  splotx = 2;
251.  sploty = 1;
252.
253.  subplot(splotx,sploty,1);
254.  plot(An((size(An,1)-5000):(size(An,1)-4500),1),'b');
255.  title('Holley Log');
256.
257.  subplot(splotx,sploty,2);
258.  plot(Bn((size(Bn,1)-5000):(size(Bn,1)-4500),1),'r');
259.  title('Wideband Log');
260.
261.  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
262.  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
263.  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
264.  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
265.  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
266.  function i = findPeaks( A, searchwindow, minoffset, minwidth )
267.
268.  % SCH 15 Mar 2005: Make sure we don't have a singleton first dimension
269.  A = squeeze(A);
270.
271.  i = zeros( size(A) );
272.
273.  % Necessary values:
274.  %   search window
275.  %   minimum peak width
276.  %   minimum peak offset
277.
278.
279.  % SCH 15 Mar 2005: if the maximum offset from any point that in the
280.  % (searchwindow) points following the current point to the current point is
```

```
% greater than a threshold, the current point is 1, otherwise 0. Then use a
% minimum filter to remove high short duration peaks. Finally, remove all 1
% values in the binary vector that don't mark the beginning of a peak.

% B is a binary vector that marks points in A where an offset greater than a
% known threshold occurs in the next (searchwindow) points of A
B = zeros( size(A) );

for x=1:size(A,1)
    % SCH 15 Mar 2005: take the next (searchwindow) points
    window = A(x:min(x+searchwindow,size(A,1)));

    % SCH 15 Mar 2005: get the absolute offset of each point from the current
    % point A(x)
    absoluteoffset = abs( window-A(x) );

    % SCH 15 Mar 2005: Check the maximum value against the threshold to get the
    % current binary value.
    B(x) = max(absoluteoffset)>minoffset;
end

% SCH 15 Mar 2005: Force a minimum peak width to remove high frequency  spikes
% by using a windowed minimum filter on the binary data
for x=1:size(B,1)
    % SCH 15 Mar 2005: take the next (minwidth) points
    window = A(x:min(x+minwidth,size(A,1)));

    % SCH 15 Mar 2005: use the minumum value
    B(x) = min(window);
end

% SCH 15 Mar 2005: We only want marks at the beginning of peaks. Shift and
% threshold.
shiftedB = [0; B(1:(length(B)-1))];
B = B>shiftedB; % same as B=B-shiftedB; B=B>0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function f = avgFilt( A, n )
% f = avgFilt( A, n )
%
% Filter A with a width n averaging filter

% SCH 15 Mar 2005: these will be the vectors after averaging
f = A;

% SCH 15 Mar 2005: the parameter is the window width. it must be odd (add 1 if necessary).
n = 2*floor(n/2)+1;

% SCH 15 Mar 2005: these are the normalizing vectors (before correcting endpoints)
normF = 1/n * ones(size(f));

% SCH 15 Mar 2005: fix the edges of the normalizing vectors
for x=1:floor(n/2)
    % SCH 15 Mar 2005: NOTE: this currently assumes length(A)>n

    % SCH 15 Mar 2005: fix left
    left = 1/(n-floor(n/2)+x-1);
    normF(x) = left;
    % SCH 15 Mar 2005: fix right
    normF(length(normF)-x+1) = left;

    % SCH 15 Mar 2005: at the same time, we can add up the windows
    % SCH 15 Mar 2005: move left and add
    f = f + [A((x+1):length(A));zeros(x,1)];
    % SCH 15 Mar 2005: move right and add
    f = f + [zeros(x,1);A(1:(length(A)-x),1)];
end
```

```
352.  f = f .* normF;
353.
354.  return;
355.
356.  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
357.  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
358.  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
359.  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
360.  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
361.  function offsets = findOffsetsCorr( A, B, w, s, n, m )
362.  % offsets = findOffsetsCorr( A, B, w, s, n, m )
363.  %
364.  %   Find the offsets of the indexes of B into A using cross-correlation. This
365.  %   assumes that the current indexes of the points in B are close to the correct
366.  %   points in A already, and is only designed to correct a small offset.
367.  %
368.  %   The actual correlation is only computed once every n points to reduce
369.  %   computational overhead. The unknown points are calculated at the end via
370.  %   interp1() with the interpolation method given in m.
371.  %
372.  %   B and A are vectors with the same length and represent samples from the same
373.  %   system with close to but not exactly the same sampling period, but averages
374.  %   out to the same sampling period across the entire data set (the endpoints
375.  %   are correctly aligned).
376.  %
377.  %   w is the correlation window width. s is the maximum offset correction.
378.
379.  % SCH 24 Mar 2005: some quick function options
380.  debug = 0;
381.  corr_diffmag = 1; % SCH 24 Mar 2005: normalize correlation power by power in differences
382.
383.  % SCH 15 Mar 2005: force the window width odd
384.  w = 2*floor(w/2)+1;
385.
386.  % SCH 24 Mar 2005: initialize the main vectors
387.  ncorr = length((s+1):n:(length(B)-w-s));
388.  if corr_diffmag==0
389.      % SCH 24 Mar 2005: preallocate arrays
390.      testLocations = zeros( ncorr+2, 1 );
391.      testOffsets = testLocations;
392.      cpower = zeros( ncorr, 1 );
393.      fprintf(1,'(%d corr tests)',ncorr);
394.  else
395.      % SCH 24 Mar 2005: let the arrays grow dynamically
396.      fprintf(1,'(%d corr tests, may skip some)',ncorr);
397.  end
398.
399.  % SCH 24 Mar 2005: Keeps track of how many offsets have been found
400.  current_location = 1;
401.
402.  % SCH 15 Mar 2005: the beginning and the end are aligned
403.  testLocations(current_location) = 1;
404.  testOffsets(current_location) = 0;
405.  current_location = current_location+1;
406.
407.  % SCH 24 Mar 2005: make sure we only calculate the average values of the input
408.  % vectors once
409.  meanA = mean(A);
410.  meanB = mean(B);
411.
412.  % Here we handle points in B where the search window is entirely within the
413.  % bounds of A.
414.  for x=(s+1):n:(length(B)-w-s)
415.      % SCH 15 Mar 2005: the window center is what we are calculating the offset of
416.      i = floor(w/2)+x;
417.      wA = A((x-s):(x+w+s-1)); % SCH 24 Mar 2005: window in A
418.      wB = B((x):(x+w-1)); % SCH 24 Mar 2005: window in B
419.      if corr_diffmag==1
420.          [r,d] = correlateInner( wA, wB );
421.          cratio = r ./ d;
422.          [c,idx] = max(cratio(1:(s+1)));
```

```
        if cratio(idx)>6 % SCH 24 Mar 2005: this is a hard coded threshold
            testLocations(current_location) = i;
            testOffsets(current_location) = -idx+s+1;
            if debug==1
                % cpower( current_location ) = c/sum(B((x):(x+w-1)) .* B((x):(x+w-1)));
                cpower( current_location ) = cratio(idx);
            end
            current_location = current_location+1;
        end
    else
        [r] = correlateInner( wA, wB );
        [c,idx] = max(r);
        testLocations(current_location) = i;
        testOffsets(current_location) = -idx+s+1;
        if debug==1
            cpower( current_location ) = c/sum(B((x):(x+w-1)) .* B((x):(x+w-1)));
        end
        current_location = current_location+1;
    end
end

% SCH 24 Mar 2005: the end is aligned
testLocations(current_location) = length(B);
testOffsets(current_location) = 0;

if debug==1
    figure; plot(testOffsets); title('testOffsets');
    figure; plot(testLocations); title('testLocations');
    figure; plot(cpower); title('cpower');
    pause
end

% SCH 15 Mar 2005: TODO: handle points where the search window goes
% partially out of bounds of A.

offsets = interp1( testLocations, testOffsets, 1:length(B), m );

return;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [r, d] = correlateInner( A, B )

% SCH 24 Mar 2005: hidden function options
debug = 0;

A = squeeze(A);
B = squeeze(B);

% SCH 16 Mar 2005: remove the DC component of B
meanB = mean(B);
mB = B-meanB;

% SCH 24 Mar 2005: preallocate output array
r = zeros( length(A)-length(B)+1, 1 );
d = ones(size(r)); % SCH 24 Mar 2005: sum of squares of differences in the window
p = ones(size(r)); % SCH 29 Mar 2005: for power maximization

for x=1:length(r)
    % SCH 24 Mar 2005: subtract the average value of B from A (this is what
    % should be the average value of A ideally, but it won't be exactly)
    mA = A((x):(x+length(mB)-1)) - meanB;
    s = mB .* mA;
    r(x) = sum(s);
    if debug==1 || nargout>=2
        s2 = mA .* mA;
```

```
494. %         p(x) = sum(s2);
495.          d(x) = sum((mA(:) - mB(:)) .^ 2);
496. %         d(x) = sqrt(sum(mA .* mA) * sum(mB .* mB));
497.     end
498. end
499.
500. if debug==1
501.     subplot(2,2,1);
502.     plot(r,'g');
503.     title('Straight peak correlation offset');
504.     r2 = zeros(size(r));
505.     [c,i] = max(r);
506.     r2(i) = r(i);
507.     hold on;
508.     plot(r2,'r');
509.     hold off;
510.
511.     subplot(2,2,3);
512.     nr = r ./ d;
513.     plot(nr,'g');
514.     title('Peak correlation to magnitude of differences ratio');
515.     r2 = zeros(size(nr));
516.     [c,i] = max(nr);
517.     r2(i) = nr(i);
518.     hold on;
519.     plot(r2,'r');
520.     hold off;
521.
522. %     subplot(2,2,3);
523. %     nr = r ./ p;
524. %     plot(nr,'g');
525. %     title('Power Maximization');
526. %     r2 = zeros(size(nr));
527. %     [c,i] = max(nr);
528. %     r2(i) = nr(i);
529. %     hold on;
530. %     plot(r2,'r');
531. %     hold off;
532. %
533.     subplot(2,2,2);
534.     plot(A);
535.     title('Window A');
536.
537.     subplot(2,2,4);
538.     plot([mean(B)*ones(1,(length(A)-length(B))/2) B mean(B)*ones(1,(length(A)-
     length(B))/2)]);
539.     title('Sub-Window B');
540.     if(max(r2)<0.1)
541.         pause;
542.     end
543. end
```

APPENDIX C. MAIN CONTROL CODE

core.m

```
 1.  function [CompOut] = core( Comp )
 2.  % [CompOut] = core( Comp )
 3.  %
 4.  % Loops through each of the targets and passes them to core functions for the
 5.  % specified platform.
 6.  %
 7.  % Created: 8 March 2005
 8.  % Updated: 10 March 2005
 9.  %
10.  % This file & its contents Copyright © 2005 Sam Harwell unless otherwise stated
11.  %
12.
13.  for t=1:size(Comp.Targets,1)
14.      % find out what platform this target belongs to, and pass it to the proper
15.      % processing core for that platform
16.      switch Comp.Targets(t).Platform
17.          case {'Commander950'} % Holley Commander 950
18.              Comp = core_Commander950( Comp, t );
19.          otherwise
20.      end
21.  end
22.
23.  CompOut = Comp;
24.
25.  return;
```

adjust_Commander950.m

```
 1. function [CompOut, err] = adjust_Commander950( Comp, targetidx, configidx, typeidx )
 2. % [CompOut, err] = adjust_Commander950( Comp, targetidx, configidx, typeidx )
 3. %
 4. % Created: 8 March 2005
 5. % Updated: 10 March 2005
 6. %
 7. % This file & its contents Copyright © 2005 Sam Harwell unless otherwise stated
 8.
 9. % find the relevant data
10. rData = [];
11. for i=1:size(Comp.Data, 1)
12.     if Comp.Data(i).Config==configidx
13.         rData = [rData; i];
14.     end
15. end
16.
17. if size(rData,1)==0
18.     CompOut = Comp;
19.     err = 'NoData';
20.     return;
21. end
22.
23. % remember that switch blocks in MATLAB don't fall through from case to case
24. switch Comp.Configs(configidx).ConfigType{typeidx}
25.     case {'FuelMap'} % update the specified fuel map
26.         % Check for a relevant Commander950 log in the data
27.         % c950log = find(Comp.Data(rData(1)).DataType=={'RawLog'},1); % TODO: search across
    more data
28.         c950log = 1;
29.         while strcmp(Comp.Data(rData(1)).DataType{c950log}, 'RawLog')~=1
30.             c950log = c950log + 1;
31.             if c950log>size(Comp.Data(rData(1)).DataType, 1)
32.                 fprintf(1, 'Error. No raw data log found.\n');
33.                 return
34.             end
35.         end
36.         if size(c950log,1)==0
37.             err = 'NoData';
38.             CompOut = Comp;
39.             return;
40.         end
41.         % Check for a matching LS1 wideband log
42.         % lm1log = find(Comp.Data(rData(1)).DataType=={'WidebandLog'},1); % TODO: search
    across more data
43.         lm1log = 1;
44.         while strcmp(Comp.Data(rData(1)).DataType{lm1log}, 'WidebandLog')~=1
45.             fprintf(1, 'Verbose: %s log found.\n', Comp.Data(rData(1)).DataType{lm1log} );
46.             lm1log = lm1log + 1;
47.             if lm1log>length(Comp.Data(rData(1)).DataType)
48.                 fprintf(1, 'Verbose: No wideband log found.\n');
49.                 break
50.             end
51.         end
52.
53.         % If there is a wideband log, we need to merge and interpolate the logs
54.         % to make sure we are using time aligned data. Otherwise, we will just
55.         % use the data straight from the Holley log. If unless the wideband log
56.         % present, data points in the raw log where O2 compensation is off need
57.         % be removed. Data we need is as follows:
58.         %   rpm
59.         %   map
60.         %   afr
61.         %   Current fuel map...
62.         %   Target AFR map... interpolate to get the targets for each data point
63.         rpm = [];
64.         map = [];
65.         afr = [];
```

```
        afrt = [];
        if size(lm1log,1)~=0 % if we have a wideband log
            hrpm = Comp.Data(rData(1)).Data{c950log}.raw(:,3)*50;
            hmap = Comp.Data(rData(1)).Data{c950log}.raw(:,14)*300/255;
            lrpm = Comp.Data(rData(1)).Data{lm1log}.rpm;
            lafr = Comp.Data(rData(1)).Data{lm1log}.afr;
            loga = [hrpm hmap]; % the Commander950 raw log
            logb = [lrpm lafr]; % the LM-1 wideband log
            % A 2500 spike marks the beginning and end of each log
            P.method = 'EndPeaks';
            P.threshold = 2500;
            % Interpolate 25000 samples along the merged logs
            P.numpts = 25000;
            [mloga, mlogb] = mergeLogs( loga, logb, P );
            % mloga is the time aligned, interpolated points from loga
            % mlogb is the time aligned, interpolated points from logb
            rpm = mlogb(:,1); % we trust the LM-1 rpm values more
            map = mloga(:,2); % map values from the Commander950
            afr = mlogb(:,2); % wideband afr
        else % else we have to rely on the narrowband log
            % filter the log to remove places where o2 compensation is off
            filtlog = processRawLog( Comp.Data(rData(1)).Data{c950log}.raw );
            % get the rpm
            rpm = filtlog(:,3)*50;
            % get the map values
            map = filtlog(:,14)*300/255;
            % get the afr from the o2mod
            o2mod = filtlog(:,24)*200/256;
            afr = o2mod .* 14.68/100;
        end
        % bilinear interpolation get the afr targets from the rpm and map values
        % t subscripts mean values from the target afr map
        % targetidx(1) =
        % targetidx(2) =
        Xt = Comp.Targets(targetidx(1)).Data{targetidx(2)}.rpm;
        Yt = Comp.Targets(targetidx(1)).Data{targetidx(2)}.map;
        Zt = Comp.Targets(targetidx(1)).Data{targetidx(2)}.afr;
        Xi = rpm;
        Yi = map;
        afrt = interp2(Xt,Yt,Zt,Xi,Yi,'linear');

        % Use the inverse bilinear interpolation MMSE algorithm to update the
        % fuel map.
        % y values of the data (rpm here)
        Yi = rpm;
        % x values of the data (map here)
        Xi = map;
        % z values of the data (AFRmeasured/AFRtarget)*OriginalFuel
        Yt = Comp.Configs(configidx).Data{typeidx}.rpm;
        Xt = Comp.Configs(configidx).Data{typeidx}.map;
        Zt = Comp.Configs(configidx).Data{typeidx}.fuel;
        fuelt = interp2(Xt,Yt,Zt',Xi,Yi,'linear');
        Zi = fuelt .* afr ./ afrt;
        % tick marks on the y axis (rpm bins)
        Yt = Comp.Configs(configidx).Data{typeidx}.rpm;
        % tick marks on the x axis (map bins)
        Xt = Comp.Configs(configidx).Data{typeidx}.map;
        % processing parameters
        P.threshold = 150;
        P.merge = 'weightedcells';
        P.extrapolate = 0; % values outside the known cells are assumed zero
        [nmap, trust] = invInterp2Lin( Xi, Yi, Zi, Xt, Yt, P );
        nmap
        trust.cells
        % for now, if it's trusted at all, we'll go ahead and update - convert
        switch trust.method
            case {'Threshold'}
                % the trust table to a binary trust table
                btrust = trust.cells>0; % binarized trust table
                nmap = nmap.*btrust;
                Comp.Configs(configidx).Data{typeidx}.fuel
```

```
                Comp.Configs(configidx).Adjustments{typeidx}.fuel = nmap ./
    Comp.Configs(configidx).Data{typeidx}.fuel;
                Comp.Configs(configidx).Adjustments{typeidx}.trust = trust;
                Comp.Configs(configidx).Adjustments{typeidx}.fuel
            otherwise
                nmap = nmap .* 0;
                Comp.Configs(configidx).Adjustments{typeidx}.fuel = nmap ./
    Comp.Configs(configidx).Data{typeidx}.fuel;
                Comp.Configs(configidx).Adjustments{typeidx}.trust = trust;
        end

    case {'AirMod'} % update the specified air modifier table

    case {'CltMod'} % update the specified coolant modifier table
    otherwise
        return;
end


```

adjust_Commander950_Fuel.m

```
 1. function [CompOut, d] = adjust_Commander950_Fuel( Comp, tidx )
 2. % [d] = adjust_Commander950_Fuel( Comp, tidx )
 3. %
 4. % Processes the 'Fuel' for the Holley Commander 950 platform. This is a
 5. % combination target including the main fuel map and all of the fuel related
 6. % modifiers.
 7. %
 8. % INPUTS:
 9. %   Comp    Main computing structure
10. %   tidx    Index of the fuel target (needed for target AFR table and processing
11. %           parameters). This is a vector of 2 elements that give the platform
12. %           target index and the parameter target index.
13. %
14. % OUTPUTS:
15. %   d   Optional diagnostic structure (trust & performance info., etc)
16. %
17. % Created: 12 March 2005
18. % Updated: 15 March 2005
19. %
20. % This file & its contents Copyright © 2005 Sam Harwell unless otherwise stated
21.
22. % SCH 10 Mar 2005: Start by pulling the target information so we know what we're
23. % doing here. Make sure it is a fuel target for the Commander950.
24. if strcmp(Comp.Targets(tidx(1)).Platform, 'Commander950')~=1
25.     fprintf(1, 'Error: Target was not for the Commander950 platform.\n');
26.     return
27. end
28. if strcmp(Comp.Targets(tidx(1)).TargetType{tidx(2)}, 'Fuel')~=1
29.     fprintf(1, 'Error: Target was not a fuel target.\n');
30.     return
31. end
32. targ = Comp.Targets(tidx(1)).Data{tidx(2)};
33. % SCH 10 Mar 2005: Note that the following parameters are now available as part
34. % of the target:
35. %   targetafr   AFR target structure with rpm, map, and afr members
36.
37.
38. % TODO: find all relevant log information
39. %   For now, we just use the first data found.
40. idx = findRelevantData(Comp);
41. if idx==0
42.     % SCH 10 Mar 2005: an error occurred in findRelevantData
43.     return;
44. end
45.
46. % SCH 10 Mar 2005:  We now have the data. Get the indexes of the matching
47. % configuration information and make sure valid original fuel maps are
48. % accessible (might have to look in base configurations for them).
49. % SCH 10 Mar 2005: TODO: We assume the fuel map is NOT inherited!! FIX THIS
50. configidx = Comp.Data(idx(1,1)).Config;
51. typeidxfuel = 1;
52. while strcmp(Comp.Configs(idx(1,1)).ConfigType{typeidxfuel},'FuelMap')~=1
53.     typeidxfuel = typeidxfuel+1;
54.     if typeidxfuel>length(Comp.Configs(idx(1,1)).ConfigType)
55.         fprintf(1, 'Error: Original fuel map configuration not found!\n');
56.         return;
57.     end
58. end
59. typeidxairmod = 1;
60. while strcmp(Comp.Configs(idx(1,1)).ConfigType{typeidxairmod},'AirMod')~=1
61.     typeidxairmod = typeidxairmod+1;
62.     if typeidxairmod>length(Comp.Configs(idx(1,1)).ConfigType)
63.         fprintf(1, 'Error: Original air temp modifier configuration not found!\n');
64.         return;
65.     end
66. end
67. typeidxcltmod = 1;
```

```
while strcmp(Comp.Configs(idx(1,1)).ConfigType{typeidxcltmod},'CltMod')~=1
    typeidxcltmod = typeidxcltmod+1;
    if typeidxcltmod>length(Comp.Configs(idx(1,1)).ConfigType)
        fprintf(1, 'Error: Original coolant temp modifier configuration not found!\n');
        return;
    end
end
c = [configidx typeidxfuel typeidxairmod typeidxcltmod];

% SCH 10 Mar 2005: Pull the required information from the logs. If wideband
% information is included, the logs will need to be time matched and merged.
% Otherwise, the log will need to be filtered to remove all samples where the
% narrowband oxygen sensor modifier is not active. Obviously the wideband
% readings are preferable.
rpm = [];  % SCH 10 Mar 2005: RPM data from the logs
map = [];  % SCH 10 Mar 2005: Manifold air pressure data from the logs
mat = [];  % SCH 11 Mar 2005: Manifold air temp data (for filtering)
clt = [];  % SCH 11 Mar 2005: Coolant temp data (for filtering)
afr = [];  % SCH 10 Mar 2005: AFR data from the logs (converted from o2mod if necessary)
afrt = []; % SCH 10 Mar 2005: Calculated target AFR for the given RPM & MAP readings
tps = [];  % SCH 15 Mar 2005: Throttle position sensor
if idx(1,3)~=0
    % SCH 10 Mar 2005: wideband data y4y!!
    fprintf(1, 'adjust_Commander950_Fuel.m: Extracting logs (w/ wideband)...');
    hrpm = Comp.Data(idx(1,1)).Data{idx(1,2)}.raw(:,3)*50;
    % TODO: this is hard coded for a 3 BAR MAP sensor!!! FIX THIS!
    hmap = Comp.Data(idx(1,1)).Data{idx(1,2)}.raw(:,14)*300/255;
    hclt = Comp.Data(idx(1,1)).Data{idx(1,2)}.raw(:,20);
    hmat = Comp.Data(idx(1,1)).Data{idx(1,2)}.raw(:,21);
    htps = Comp.Data(idx(1,1)).Data{idx(1,2)}.raw(:,15);
    lrpm = Comp.Data(idx(1,1)).Data{idx(1,3)}.rpm;
    lafr = Comp.Data(idx(1,1)).Data{idx(1,3)}.afr;
    % SCH 29 Mar 2005: get rid of sensor spikes with a narrow median filter
    hmap = medfilt1( hmap, 3 );
    hclt = medfilt1( hclt, 3 );
    hmat = medfilt1( hmat, 3 );
    htps = medfilt1( htps, 3 );
    lrpm = medfilt1( lrpm, 3 );
    lafr = medfilt1( lafr, 3 );

    loga = [hrpm hmap hmat hclt htps]; % the Commander950 raw log
    logb = [lrpm lafr]; % the LM-1 wideband log
    clear P;
    % A 2500rpm peak marks the beginning and end of each log
    P.alignmethod = 'xcorr';
    P.avgfiltwidth = 9;
%     P.thresholdbegin = 1300;
%     P.thresholdend = 1300;
    P.thresholdbegin = 2500;
    P.thresholdend = 3000;
%     % Interpolate 25000 samples along the merged logs for the 05 March data set
%     P.numpts = 25000;
    % Interpolate 40000 samples along the merged logs for the 01 April data set
    % P.numpts = 40000;
    % Interpolate numpts in the logs... round up to the next 5000pt boundary
    P.numpts = ceil( max(length(hrpm),length(lrpm))/5000 )*5000;
    [mloga, mlogb] = mergeLogs( loga, logb, P );
    % mloga is the time aligned, interpolated points from loga
    % mlogb is the time aligned, interpolated points from logb
    rpm = mlogb(:,1); % we trust the LM-1 rpm values more
    map = mloga(:,2); % map values from the Commander950
    mat = mloga(:,3); % mat values from the Commander950
    clt = mloga(:,4); % clt temp values from the Commander950
    afr = mlogb(:,2); % wideband afr
    tps = mloga(:,5); % SCH 15 Mar 2005: from Commander950 log
else
    % SCH 10 Mar 2005: no wideband data :Owned:
    % filter the log to remove places where o2 compensation is off
    fprintf(1, 'adjust_Commander950_Fuel.m: Extracting logs (w/o wideband)...');
    filtlog = processRawLog( Comp.Data(idx(1,1)).Data{idx(1,2)}.raw );
    % get the rpm
```

```
139.      rpm = filtlog(:,3)*50;
140.      % get the map values
141.      % TODO: this is hard coded for a 3 BAR MAP sensor!!! FIX THIS!
142.      map = filtlog(:,14)*300/255;
143.      clt = filtlog(:,20); % clt temp values from the Commander950
144.      mat = filtlog(:,21); % mat values from the Commander950
145.      % SCH 15 Mar 2005: note that in this mode, we won't be seeing any high TPS
146.      % readings because the car goes open loop at that time so those times are
147.      % filtered out above in the processRawLog function.
148.      tps = filtlog(:,15); % mat values from the Commander950
149.      % get the afr from the o2mod
150.      o2mod = filtlog(:,24)*200/256;
151.      afr = o2mod .* 14.68/100;
152.  end
153.
154.  % bicubic (because we can) interpolation get the afr targets from the rpm and
155.  % map values
156.  % t subscripts mean values from the target afr map
157.  Xt = targ.targetafr.rpm;
158.  Yt = targ.targetafr.map;
159.  Zt = targ.targetafr.afr;
160.  Xi = rpm;
161.  Yi = map;
162.  % SCH 14 Mar 2005: TODO: Account for possible NaN outputs here
163.  afrt = interp2(Xt,Yt,Zt,Xi,Yi,'linear');
164.
165.  fprintf(1, 'done.\n'); % SCH 10 Mar 2005: Extracting logs
166.
167.  % SCH 10 Mar 2005: Prepare to send the data to the inverted cell interpolator
168.  % (ICI). This includes specifying any processing parameters that are needed such
169.  % as trust table, cell merge, and motion compensation parameters. Motion
170.  % compensation based on sensors OTHER than those used in the ICI like dTPS/dt
171.  % will need to be adjusted before sending data to the ICI. Also, histogram data
172.  % filtering on modifier parameters needs to be done here. Examples are removing
173.  % all points that do not lie within a given percentage of the standard deviation
174.  % of the mean of the overall air or coolant temperature sensor readings.
175.
176.  % SCH 12 Mar 2005: that comment is straight from the original fuel map only
177.  % calculation function. This function now calculates the modifier updates as
178.  % well.
179.
180.  % SCH 14 Mar 2005: I'll just continue here with my notes on the subject I guess.
181.  % A possible idea would be to filter the data when making the fuel map, and then
182.  % only use points from the entire original data set that fall into areas of the
183.  % fuel map that were actually adjusted. This certainly wouldn't be the easiest
184.  % thing in the world to do, and I don't see it being guaranteed to work, but
185.  % it's certainly not beyond the scope of this project to try either.
186.
187.  % SCH 14 Mar 2005: And now--an idea for solving cells that contain minimal
188.  % numbers of data points. Since the points are really samples of a continuous
189.  % system, I should be able to connect a spline through the cells and interpolate
190.  % new points on it and make the assumption that these points are as accurate as
191.  % "known" points which are measured but have error.
192.
193.  fprintf(1, 'adjust_Commander950_Fuel.m: Preparing data for ICI...');
194.  % SCH 12 Mar 2005: The data filtering step will be moved later in the function
195.  % to speed up incremental processing
196.  % First step: data filtering
197.
198.
199.  if 0
200.      % SCH 26 Mar 2005: here is the filtering
201.      % SCH 26 Mar 2005: smooth the RPM, MAP, MAT, CLT, and AFR readings
202.          filt = testfilter();
203.          filtrpm = filter( filt, rpm );
204.          filtmap = filter( filt, map );
205.          filtmat = filter( filt, mat );
206.          filtclt = filter( filt, clt );
207.          filtafr = filter( filt, afr );
208.          % SCH 25 Mar 2005: some of the sections below don't use the filt*
209.          % vectors, so we need to update the originals
```

```
210.          rpm = filtrpm;
211.          map = filtmap;
212.          mat = filtmat;
213.          clt = filtclt;
214.          afr = filtafr;
215.      % SCH 26 Mar 2005: recalculate the AFRT values
216.          % bicubic (because we can) interpolation get the afr targets from the
217.          % rpm and map values
218.          % t subscripts mean values from the target afr map
219.          Xt = targ.targetafr.rpm;
220.          Yt = targ.targetafr.map;
221.          Zt = targ.targetafr.afr;
222.          Xi = filtrpm;
223.          Yi = filtmap;
224.          % SCH 14 Mar 2005: TODO: Account for possible NaN outputs here
225.          afrt = interp2(Xt,Yt,Zt,Xi,Yi,'linear');
226.          filtafrt = afrt;
227.  else
228.      % SCH 11 Mar 2005: filtering disabled
229.      filtrpm = rpm;
230.      filtmap = map;
231.      filtmat = mat;
232.      filtclt = clt;
233.      filtafr = afr;
234.      filtafrt = afrt;
235.  end
236.
237.  % SCH 15 Mar 2005: save the known log values in the data section of Comp. This
238.  % is especially beneficial in cases where the Wideband log is used because we
239.  % can look at the aligned data after processing is finished.
240.  Comp.Data(idx(1,1)).Data{idx(1,2)}.rpm = filtrpm;
241.  Comp.Data(idx(1,1)).Data{idx(1,2)}.map = filtmap;
242.  Comp.Data(idx(1,1)).Data{idx(1,2)}.mat = filtmat;
243.  Comp.Data(idx(1,1)).Data{idx(1,2)}.clt = filtclt;
244.  Comp.Data(idx(1,1)).Data{idx(1,2)}.afr = filtafr;
245.  Comp.Data(idx(1,1)).Data{idx(1,2)}.afrt = filtafrt;
246.  Comp.Data(idx(1,1)).Data{idx(1,2)}.tps = tps;
247.
248.  % Use the inverse bilinear interpolation MMSE algorithm to update the
249.  % fuel map.
250.  % SCH 10 Mar 2005: RPM may look like the X axis, but MATLAB stores matrices in
251.  % column-major order.
252.  % SCH 12 Mar 2005: These are the Fuel Map values
253.  % y values of the data (rpm here)
254.  Yi = filtrpm;
255.  % x values of the data (map here)
256.  Xi = filtmap;
257.  % z values of the data (AFRmeasured/AFRtarget)*OriginalFuel
258.  Yt = Comp.Configs(configidx).Data{typeidxfuel}.rpm;
259.  Xt = Comp.Configs(configidx).Data{typeidxfuel}.map;
260.  Zt = Comp.Configs(configidx).Data{typeidxfuel}.fuel';
261.  OriginalFuelMap = Zt';
262.  % SCH 15 Mar 2005: pad Xt and Zt to prevent NaN's returned from interp2
263.  if Xt(1)<Xt(length(Xt))
264.      Xt = [0;Xt];
265.      Zt = [zeros(length(Yt),1) Zt];
266.  else
267.      Xt = [Xt;0];
268.      Zt = [Zt zeros(length(Yt),1)];
269.  end
270.  filtfueltarg = interp2(Xt,Yt,Zt,Xi,Yi,'linear');
271.  % tick marks on the y axis (rpm bins)
272.  Yt = Comp.Configs(configidx).Data{typeidxfuel}.rpm;
273.  % tick marks on the x axis (map bins)
274.  Xt = Comp.Configs(configidx).Data{typeidxfuel}.map;
275.  % SCH 12 Mar 2005: make it known that these are the parameters for the fuel map
276.  % calculation
277.  XiFuelMap = Xi;
278.  YiFuelMap = Yi;
279.  XtFuelMap = Xt;
280.  YtFuelMap = Yt;
```

```
[XtFuelMap,YtFuelMap] = meshgrid( Xt, Yt );
% processing parameters for the fuel map
PFuelMap.method = 'LMS';
PFuelMap.biased = 1;
PFuelMap.priorweight = 20;
PFuelMap.OldZi = OriginalFuelMap';
% PFuelMap.method = 'threshold';
PFuelMap.threshold = 150;
PFuelMap.merge = 'weightedcells';
PFuelMap.extrapolate = 0; % values outside the known cells are assumed zero
PFuelMap.interpfill = targ.interpfill;
PFuelMap.windowexpand = targ.windowexpand;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SCH 13 Mar 2005: these are the air modifier values
% y values of the data (AFRmeasured/AFRtarget)*(OrigFuel/NewFuel)*OrigMod
% x values of the data (modifier temp)
XiAirMod = mat;
% tick marks on the x axis (temp bins)
Xt = Comp.Configs(configidx).Data{typeidxairmod}.temp;
% original modifier map
Yt = Comp.Configs(configidx).Data{typeidxairmod}.modifier;
oairmod = interp1q(Xt, Yt, XiAirMod);
OriginalAirMod = Yt;
% tick marks on the x axis (temp bins)
XtAirMod = Comp.Configs(configidx).Data{typeidxairmod}.temp;
% processing parameters
%PAirMod.method = 'threshold';
PAirMod.method = 'LMS';
PAirMod.biased = 1;
PAirMod.priorweight = 20;
PAirMod.OldYi = Yt;
PAirMod.threshold = 150;
PAirMod.merge = 'weightedcells';
PAirMod.extrapolate = 0; % values outside the known cells are assumed zero
if targ.manualmod~=1
    Comp.Targets(tidx(1)).Adjustments{tidx(2)}.current.airmod = OriginalAirMod;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SCH 13 Mar 2005: these are the coolant modifier values
% y values of the data (AFRmeasured/AFRtarget)*(OrigFuel/NewFuel)*OrigMod
% x values of the data (modifier temp)
XiCltMod = clt;
% tick marks on the x axis (temp bins)
Xt = Comp.Configs(configidx).Data{typeidxcltmod}.temp;
% original modifier map
Yt = Comp.Configs(configidx).Data{typeidxcltmod}.modifier;
OriginalCltMod = Yt;
ocltmod = interp1q(Xt, Yt, XiCltMod);
% tick marks on the x axis (temp bins)
XtCltMod = Comp.Configs(configidx).Data{typeidxcltmod}.temp;
% processing parameters
%PCltMod.method = 'threshold';
PCltMod.method = 'LMS';
PCltMod.biased = 1;
PCltMod.priorweight = 20;
PCltMod.OldYi = Yt;
PCltMod.threshold = 150;
PCltMod.merge = 'weightedcells';
PCltMod.extrapolate = 0; % values outside the known cells are assumed zero
if targ.manualmod~=1
    Comp.Targets(tidx(1)).Adjustments{tidx(2)}.current.cltmod = OriginalCltMod;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


fprintf(1, 'done.\n');

% SCH 13 Mar 2005: here is where we'll eventually loop for the incremental
```

```
% processing
for x=1:targ.incremental
    maxidx = round(min(length(XiFuelMap),length(XiFuelMap)*x/targ.incremental));
    fprintf(1, 'adjust_Commander950_Fuel.m: Calling ICI (round %.2d/%d)...', x,
targ.incremental);
    tic;

    % SCH 24 Mar 2005: fuel table processing
    for pass = 1:targ.numpasses

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % SCH 13 Mar 2005: Fuel map
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    ZiFuelMap = filtfueltarg(1:maxidx) .* filtafr(1:maxidx) ./ filtafrt(1:maxidx);
    % SCH 13 Mar 2005: make sure to adjust for any updated modifiers
    if pass==1
        % remove modifier offsets
        % ZiFuelMap = ZiFuelMap ./ (ocltmod(1:maxidx) .* oairmod(1:maxidx));
    end
    % SCH 24 Mar 2005: coolant modifier adjustment
        % tick marks on the x axis (temp bins)
        Xt = Comp.Configs(configidx).Data{typeidxcltmod}.temp;
        % original modifier map
        Yt = Comp.Targets(tidx(1)).Adjustments{tidx(2)}.current.cltmod;
        ncltmod = interp1(Xt, Yt, XiCltMod(1:maxidx));
        ZiFuelMap = ZiFuelMap .* ocltmod(1:maxidx) ./ ncltmod;
    % SCH 24 Mar 2005: mat modifier adjustment
        % tick marks on the x axis (temp bins)
        Xt = Comp.Configs(configidx).Data{typeidxairmod}.temp;
        % original modifier map
        Yt = Comp.Targets(tidx(1)).Adjustments{tidx(2)}.current.airmod;
        nairmod = interp1(Xt, Yt, XiAirMod(1:maxidx));
        ZiFuelMap = ZiFuelMap .* oairmod(1:maxidx) ./ nairmod;
    if 0 % data filtering
        % remove values that are outside the primary coolant temp range
        goodpoints = (clt>170) & (clt<185);
        goodpoints = goodpoints(1:maxidx);
        gpidx = find(goodpoints==1);
    else
        goodpoints = ones(size(clt));
        goodpoints = goodpoints(1:maxidx);
        gpidx = find(goodpoints==1);
    end
        XiFuelMapGood = XiFuelMap(1:maxidx);
        YiFuelMapGood = YiFuelMap(1:maxidx);
        ZiFuelMapGood = ZiFuelMap;
        XiFuelMapGood = XiFuelMapGood(goodpoints~=0);
        YiFuelMapGood = YiFuelMapGood(goodpoints~=0);
        ZiFuelMapGood = ZiFuelMapGood(goodpoints~=0);
    % SCH 13 Mar 2005: calculate
    [nmapfuel, trustfuel] = invInterp2Lin( XiFuelMapGood, YiFuelMapGood, ZiFuelMapGood,
XtFuelMap, YtFuelMap, PFuelMap );
    %trustfuelused = zeros(size(goodpoints));
    %trustfuelused(gpidx) = trustfuel.used;
    %trustfuel.used = trustfuelused;
    % SCH 13 Mar 2005: Save the new tables
    nmapfuel = nmapfuel';
    bmapfuel = ~(nmapfuel>0);
    NewFuelMap = nmapfuel + (OriginalFuelMap .* bmapfuel);
    Comp.Targets(tidx(1)).Adjustments{tidx(2)}.incremental(x).fuel = NewFuelMap;
    Comp.Targets(tidx(1)).Adjustments{tidx(2)}.incremental(x).trustfuel = trustfuel;
    Comp.Targets(tidx(1)).Adjustments{tidx(2)}.current.fuel = NewFuelMap;
    Comp.Targets(tidx(1)).Adjustments{tidx(2)}.current.trustfuel = trustfuel;

    % SCH 13 Mar 2005: modifiers can have more than one iteration to account for
    % changes between them.
    for moditer=1:targ.moditerations
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % SCH 13 Mar 2005: air temp modifier
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % SCH 24 Mar 2005: added the maxidx restriction
```

```
421.          YiAirMod = afr(1:maxidx) .* oairmod(1:maxidx) ./ afrt(1:maxidx);
422.          % SCH 13 Mar 2005: make sure to adjust for any updated modifiers (fuel map here)
423.              if(moditer==1) % SCH 13 Mar 2005: fuel table doesn't change between iterations
424.                  ofuel = ones( size(afr(1:maxidx)) ); % SCH 11 Mar 2005: original fuel map
      (interp)
425.                  nfuel = ones( size(afr(1:maxidx)) ); % SCH 11 Mar 2005: new fuel map
      (interp)
426.                  % SCH 10 Mar 2005: RPM may look like the X axis, but MATLAB stores matrices
      in
427.                  % column-major order.
428.                  % y values of the data (rpm here)
429.                  Yi = rpm(1:maxidx);
430.                  % x values of the data (map here)
431.                  Xi = map(1:maxidx);
432.                  % SCH 11 Mar 2005: the ofuel/nfuel ratio might not be 1
433.                  Yt = Comp.Configs(configidx).Data{typeidxfuel}.rpm;
434.                  Xt = Comp.Configs(configidx).Data{typeidxfuel}.map;
435.                  Zt = OriginalFuelMap';
436.                  ofuel = interp2(Xt,Yt,Zt,Xi,Yi,'linear',-1);
437.                  Zt = NewFuelMap';
438.                  nfuel = interp2(Xt,Yt,Zt,Xi,Yi,'linear',-1);
439.              end
440.              % SCH 13 Mar 2005: clt modifier DOES change
441.                  % tick marks on the x axis (temp bins)
442.                  Xt = Comp.Configs(configidx).Data{typeidxcltmod}.temp;
443.                  % original modifier map
444.                  Yt = Comp.Targets(tidx(1)).Adjustments{tidx(2)}.current.cltmod;
445.                  ncltmod = interp1q(Xt, Yt, XiCltMod(1:maxidx));
446.                  YiAirMod = YiAirMod .* ocltmod(1:maxidx) ./ ncltmod;
447.              YiAirMod = YiAirMod .* ofuel ./ nfuel;
448.              % remove extrapolated values
449.              goodpoints = (nfuel>0);
450.              goodpoints = goodpoints(1:maxidx);
451.              % SCH 13 Mar 2005: also remove values that aren't in updated sections of the
      fuel map
452.              %goodpoints = goodpoints & trustfuel.used;
453.              %XiAirModGood = XiAirMod(1:maxidx);
454.              %YiAirModGood = YiAirMod(1:maxidx);
455.              XiAirModGood = XiAirMod(goodpoints~=0);
456.              YiAirModGood = YiAirMod(goodpoints~=0);
457.          % SCH 13 Mar 2005: calculate
458.          [nairmod, trustairmod] = invInterp1Lin( XiAirModGood, YiAirModGood, XtAirMod,
      PAirMod );
459.  %         for i=1:length(trustairmod.ncell)
460.  %             if trustairmod.ncell(i)<=PAirMod.threshold
461.  %                 % SCH 26 Apr 2005: fix left
462.  %                 if nairmod(i)>0
463.  %                     nairmod(i) = (nairmod(i)*trustairmod.ncell(i-
      1)+OriginalAirMod(i)*trustairmod.ncell(i))/(trustairmod.ncell(i-1)+trustairmod.ncell(i));
464.  %                 end
465.  %                 % SCH 26 Apr 2005: fix right
466.  %                 if nairmod(i+1)>0
467.  %                     nairmod(i+1) =
      (nairmod(i+1)*trustairmod.ncell(i+1)+OriginalAirMod(i+1)*trustairmod.ncell(i))/(trustairmod
      .ncell(i+1)+trustairmod.ncell(i));
468.  %                 end
469.  %             end
470.  %         end
471.          % SCH 13 Mar 2005: Save the new tables
472.          bairmod = ~(nairmod>0);
473.          NewAirMod = nairmod + (OriginalAirMod .* bairmod);
474.          Comp.Targets(tidx(1)).Adjustments{tidx(2)}.incremental(x).airmod = NewAirMod;
475.          Comp.Targets(tidx(1)).Adjustments{tidx(2)}.current.airmod = NewAirMod;
476.          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
477.          % SCH 13 Mar 2005: coolant temp modifier
478.          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
479.          % SCH 24 Mar 2005: added the maxidx restriction
480.          YiCltMod = afr(1:maxidx) .* ocltmod(1:maxidx) ./ afrt(1:maxidx);
481.          % SCH 13 Mar 2005: make sure to adjust for any updated modifiers (fuel map and air
      mod here)
482.              % tick marks on the x axis (temp bins)
```

```
            Xt = Comp.Configs(configidx).Data{typeidxairmod}.temp;
            % original modifier map
            Yt = Comp.Targets(tidx(1)).Adjustments{tidx(2)}.current.airmod;
            nairmod = interp1q(Xt, Yt, XiAirMod(1:maxidx));
            YiCltMod = YiCltMod .* oairmod(1:maxidx) ./ nairmod;
            YiCltMod = YiCltMod .* ofuel ./ nfuel;
            % remove extrapolated values
            goodpoints = (nfuel>0);
            goodpoints = goodpoints(1:maxidx);
            % SCH 13 Mar 2005: also remove values that aren't in updated sections of the
    fuel map
            %goodpoints = goodpoints & trustfuel.used;
            %XiCltModGood = XiCltMod(1:maxidx);
            %YiCltModGood = YiCltMod(1:maxidx);
            XiCltModGood = XiCltMod(goodpoints);
            YiCltModGood = YiCltMod(goodpoints);
        % SCH 13 Mar 2005: calculate
        [ncltmod, trustcltmod] = invInterp1Lin( XiCltModGood, YiCltModGood, XtCltMod,
    PCltMod );
%         for i=1:length(trustcltmod.ncell)
%             if trustcltmod.ncell(i)<=PCltMod.threshold
%                 % SCH 26 Apr 2005: fix left
%                 if ncltmod(i)>0
%                     ncltmod(i) = (ncltmod(i)*trustcltmod.ncell(i-
    1)+OriginalCltMod(i)*trustcltmod.ncell(i))/(trustcltmod.ncell(i-1)+trustcltmod.ncell(i));
%                 end
%                 % SCH 26 Apr 2005: fix right
%                 if ncltmod(i+1)>0
%                     ncltmod(i+1) =
    (ncltmod(i+1)*trustcltmod.ncell(i+1)+OriginalCltMod(i+1)*trustcltmod.ncell(i))/(trustcltmod
    .ncell(i+1)+trustcltmod.ncell(i));
%                 end
%             end
%         end
        % SCH 13 Mar 2005: Save the new tables
        bcltmod = ~(ncltmod>0);
        NewCltMod = ncltmod + (OriginalCltMod .* bcltmod);
        Comp.Targets(tidx(1)).Adjustments{tidx(2)}.incremental(x).cltmod = NewCltMod;
        Comp.Targets(tidx(1)).Adjustments{tidx(2)}.current.cltmod = NewCltMod;
    end % SCH 13 Mar 2005: modifier iteration loopy
    [original_mse,adjusted_mse]=afrmse(Comp,maxidx);
    if pass==1
        fprintf(1,'%f...',original_mse);
    end
    fprintf(1,'%f...',adjusted_mse);
    end % SCH 24 Mar 2005: COMPLETE PASS LOOP
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % SCH 13 Mar 2005: processing for this round is done
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    fprintf(1, 'done.\n');
    toc;
    if 0
        % SCH 15 Mar 2005: remove bias from air temperature modifier
        a = sum(Comp.Targets(tidx(1)).Adjustments{tidx(2)}.incremental(x).airmod) / ...
            (100*length(Comp.Targets(tidx(1)).Adjustments{tidx(2)}.incremental(x).airmod));
        Comp.Targets(tidx(1)).Adjustments{tidx(2)}.incremental(x).airmod = ...
            Comp.Targets(tidx(1)).Adjustments{tidx(2)}.incremental(x).airmod * 1/a;
        Comp.Targets(tidx(1)).Adjustments{tidx(2)}.current.airmod = ...
            Comp.Targets(tidx(1)).Adjustments{tidx(2)}.current.airmod * 1/a;
        Comp.Targets(tidx(1)).Adjustments{tidx(2)}.incremental(x).fuel = ...
            Comp.Targets(tidx(1)).Adjustments{tidx(2)}.incremental(x).fuel * a;
        Comp.Targets(tidx(1)).Adjustments{tidx(2)}.current.fuel = ...
            Comp.Targets(tidx(1)).Adjustments{tidx(2)}.current.fuel * a;
        % SCH 15 Mar 2005: remove bias from coolant temperature modifier
        a = sum(Comp.Targets(tidx(1)).Adjustments{tidx(2)}.incremental(x).cltmod) / ...
            (100*length(Comp.Targets(tidx(1)).Adjustments{tidx(2)}.incremental(x).cltmod));
        Comp.Targets(tidx(1)).Adjustments{tidx(2)}.incremental(x).cltmod = ...
            Comp.Targets(tidx(1)).Adjustments{tidx(2)}.incremental(x).cltmod * 1/a;
        Comp.Targets(tidx(1)).Adjustments{tidx(2)}.current.cltmod = ...
            Comp.Targets(tidx(1)).Adjustments{tidx(2)}.current.cltmod * 1/a;
        Comp.Targets(tidx(1)).Adjustments{tidx(2)}.incremental(x).fuel = ...
```

```
            Comp.Targets(tidx(1)).Adjustments{tidx(2)}.incremental(x).fuel * a;
        Comp.Targets(tidx(1)).Adjustments{tidx(2)}.current.fuel = ...
            Comp.Targets(tidx(1)).Adjustments{tidx(2)}.current.fuel * a;
    end
end;

CompOut = Comp;

if nargout>=2
    d = []; % SCH 13 Mar 2005: might return diagnostic information in the future
end

return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [idx] = findRelevantData( Comp )
% This function is just used to find and return the first instance of a Holley
% Commander 950 log in the data subsection of the Comp structure. idx contains
% three elements: the data index, the sub index of the Holley raw log, and, if
% found, the sub index of the matching LM-1 wideband log.
% *** FOR LATER UPGRADES TO MULTIPLE LOG PROCESSING ABILITY: idx is a ROW
%     vector!!! ***
if length(Comp.Data)<1
    fprintf(1, 'Error: No measured data found for processing.\n');
    idx = 0;
    return;
end

fprintf(1, 'adjust_Commander950_Fuel.m: Finding relevant data...');

for tmpidx=1:length(Comp.Data)
    lmlog = 0;
    h950log = 0;
    for tmpsubidx=1:length(Comp.Data(tmpidx).Platform)
        if strcmp(Comp.Data(tmpidx).Platform{tmpsubidx}, 'Commander950')==1 && ...
                strcmp(Comp.Data(tmpidx).DataType{tmpsubidx}, 'RawLog')==1
            h950log = tmpsubidx;
        end
        if strcmp(Comp.Data(tmpidx).Platform{tmpsubidx}, 'LM1')==1 && ...
                strcmp(Comp.Data(tmpidx).DataType{tmpsubidx}, 'WidebandLog')==1
            lmlog = tmpsubidx;
        end
    end
    if h950log~=0
        idx = [tmpidx h950log lmlog];
        fprintf(1, 'done.\n');
        return;
    end
end

fprintf(1, 'NONE FOUND!\n');
idx = 0;
return;
```

APPENDIX D. OPTIMIZATION PARAMETER SETTINGS CODE

loadSettings1.m

```
 1.  function [Comp] = loadSettings1()
 2.  % [Comp] = loadSettings1()
 3.  %
 4.  % Created: 6 March 2005
 5.  % Updated: 15 March 2005
 6.  %
 7.  % This file & its contents Copyright © 2005 Sam Harwell unless otherwise stated
 8.
 9.  Comp.Configs = struct([]);
10.  Comp.Targets = struct([]);
11.  Comp.Data = struct([]);
12.
13.  %%%%%%%%%%
14.  % config %
15.  %%%%%%%%%%
16.  Comp.Configs(1).BaseID = uint32(0);
17.  Comp.Configs(1).Platform = 'Commander950';
18.  Comp.Configs(1).ConfigType = { };
19.  Comp.Configs(1).Data = { };
20.  Comp.Configs(1).Adjustments = { };
21.
22.  % Fuel map
23.  index = uint32(1); % 1
24.  Comp.Configs(1).ConfigType{index} = 'FuelMap';
25.  Comp.Configs(1).Data{index}.rpm =
     [400;800;1000;1200;1500;1700;2000;2500;3000;3500;4000;4500;5000;5500;6000;6500];
26.  Comp.Configs(1).Data{index}.map = [19;38;56;75;94;113];
27.  OrigFuelMap = zeros( size(Comp.Configs(1).Data{index}.map,1),
     size(Comp.Configs(1).Data{index}.rpm,1) );
28.  OrigFuelMap(6,:) = [50 70 95 115 130 130 143 143 143 143 143 143 143 143 140 128]; % 113
29.  OrigFuelMap(5,:) = [50 70 95 115 130 130 143 143 143 143 143 143 143 143 140 128]; %  94
30.  OrigFuelMap(4,:) = [58 47 48  79  88  80  78  89  89 110 110 105 105 100  85  85]; %  75
31.  OrigFuelMap(3,:) = [29 38 37  50  53  48  48  52  56  79  79  79  79  74  58  58]; %  56
32.  OrigFuelMap(2,:) = [13 27 26  34  33  30  28  31  36  48  56  54  54  52  42  42]; %  38
33.  OrigFuelMap(1,:) = [ 8  8  8  15  15  15  17  16  18  30  30  27  26  23  18  18]; %  19
34.  Comp.Configs(1).Data{index}.fuel = OrigFuelMap;
35.  Comp.Configs(1).Adjustments{index}.fuel = ones(size(Comp.Configs(1).Data{index}.fuel));
36.  Comp.Configs(1).Adjustments{index}.ready = 0;
37.  Comp.Configs(1).Adjustments{index}.trust = zeros(size(Comp.Configs(1).Data{index}.fuel));
38.
39.  % Air temperature enrichment modifier
40.  index = index+1; % 2
41.  Comp.Configs(1).ConfigType{index} = 'AirMod';
42.  Comp.Configs(1).Data{index}.temp = [0;11;32;41;54;66;75;87;98;110;121;138;156;184;212;235];
43.  Comp.Configs(1).Data{index}.modifier =
     [125.0;125.0;123.4;121.9;118.8;115.6;114.1;110.9;109.4;107.0;104.7;101.6;99.2;94.5;90.6;87.
     5];
44.  Comp.Configs(1).Adjustments{index}.modifier =
     ones(size(Comp.Configs(1).Data{index}.modifier));
45.
46.  % Coolant temperature enrichment modifer
47.  index = index+1; % 3
48.  Comp.Configs(1).ConfigType{index} = 'CltMod';
49.  Comp.Configs(1).Data{index}.temp = [0;11;32;41;54;66;75;87;98;110;121;138;156;184;212;235];
50.  Comp.Configs(1).Data{index}.modifier =
     [106.3;103.1;100.0;100.0;100.0;100.0;100.0;100.0;100.0;100.0;100.0;100.0;100.0;100.0;
     100.0];
51.  Comp.Configs(1).Adjustments{index}.modifier =
     ones(size(Comp.Configs(1).Data{index}.modifier));
52.
53.
54.  %%%%%%%%%%
55.  % targets %
56.  %%%%%%%%%%
57.  Comp.Targets(1).Platform = 'Commander950';
58.
59.  % SCH 14 Mar 2005: added the 0 map row to prevent NaN values from interp2
```

```
 60.  targetafr.rpm =
      [400;800;1000;1200;1500;1700;2000;2500;3000;3500;4000;4500;5000;5500;6000;6500];
 61.  targetafr.map = [0;19;38;56;75;94;113];
 62.  targetafr.afr = zeros( size(targetafr.map,1), size(targetafr.rpm,1) );
 63.  % SCH 25 Mar 2005: made more adjustments...dropped idle afr to stoich
 64.  % SCH 25 Mar 2005:    400   800   1000  1200  1500  1700  2000  2500  3000  3500  4000
      4500  5000  5500  6000  6500
 65.  targetafr.afr(7,:) = [13.00 13.00 13.00 12.20 12.20 12.20 12.20 12.20 12.20 12.20 12.20
      12.20 12.20 12.20 12.20 12.20]; % 113
 66.  targetafr.afr(6,:) = [13.00 13.00 13.00 12.20 12.20 12.20 12.20 12.20 12.20 12.20 12.20
      12.20 12.20 12.20 12.20 12.20]; %  94
 67.  targetafr.afr(5,:) = [14.00 14.00 14.00 12.20 12.20 12.20 12.20 12.20 12.20 12.20 12.20
      12.20 12.20 12.20 12.20 12.20]; %  75
 68.  targetafr.afr(4,:) = [14.68 14.68 14.68 14.68 15.50 15.50 15.50 15.00 15.00 14.30 14.30
      14.30 14.30 14.30 14.30 14.30]; %  56
 69.  targetafr.afr(3,:) = [14.68 14.68 14.68 14.68 15.40 15.40 16.00 16.00 16.00 16.00 16.00
      16.00 16.00 16.00 16.00 16.00]; %  38
 70.  targetafr.afr(2,:) = [14.68 14.68 14.68 14.68 15.40 15.40 16.00 16.00 16.00 16.00 16.00
      16.00 16.00 16.00 16.00 16.00]; %  19
 71.  targetafr.afr(1,:) = [14.68 14.68 14.68 14.68 15.40 15.40 16.00 16.00 16.00 16.00 16.00
      16.00 16.00 16.00 16.00 16.00]; %   0
 72.
 73.  % SCH 12 Mar 2005: quick update - "Fuel" is now one of the available targets. If
 74.  % specified, it means that the fuel map and all of the fuel modifiers should be
 75.  % computed for the current platform.
 76.  index = uint32(1);
 77.  Comp.Targets(1).TargetType{index} = 'Fuel';
 78.  Comp.Targets(1).Data{index}.targetafr = targetafr;
 79.  % do 100 builds leading up to the final solution to watch progress
 80.  Comp.Targets(1).Data{index}.incremental = 10;
 81.  % SCH 24 Mar 2005: number of complete passes
 82.  Comp.Targets(1).Data{index}.numpasses = 1;
 83.  % SCH 24 Mar 2005: number of modifier iterations per pass
 84.  Comp.Targets(1).Data{index}.moditerations = 1;
 85.  Comp.Targets(1).Data{index}.manualmod = 0;
 86.  % SCH 15 Mar 2005: one method for dealing with sparse cells (unsuccessful so far)
 87.  Comp.Targets(1).Data{index}.interpfill = 0;
 88.  % SCH 15 Mar 2005: another method for dealing with sparse cells (unsuccessful so far)
 89.  Comp.Targets(1).Data{index}.windowexpand = 0;
 90.
 91.  % SCH 12 Mar 2005: these lines just show where the results will go
 92.  % Comp.Targets(1).Adjustments{index}.incremental(i).fuel;
 93.  % Comp.Targets(1).Adjustments{index}.incremental(i).airmod;
 94.  % Comp.Targets(1).Adjustments{index}.incremental(i).cltmod;
 95.  % Comp.Targets(1).Adjustments{index}.current.fuel;
 96.  % Comp.Targets(1).Adjustments{index}.current.airmod;
 97.  % Comp.Targets(1).Adjustments{index}.current.cltmod;
 98.
 99.
100.  % index = index+1;
101.  % Comp.Targets(1).TargetType{index} = 'FuelMap';
102.  % Comp.Targets(1).Data{index}.targetafr = targetafr;
103.  %
104.  % index = index+1;
105.  % Comp.Targets(1).TargetType{index} = 'AirMod';
106.  % Comp.Targets(1).Data{index}.targetafr = targetafr;
107.  %
108.  % index = index+1;
109.  % Comp.Targets(1).TargetType{index} = 'CltMod';
110.  % Comp.Targets(1).Data{index}.targetafr = targetafr;
111.
112.  %%%%%%%%
113.  % data %
114.  %%%%%%%%
115.  index = uint32(1);
116.  Comp.Data(1).Config = 1;
117.  Comp.Data(1).Platform{index} = 'Commander950';
118.  Comp.Data(1).DataType{index} = 'RawLog';
119.  S = load('RawHolleyLog.mat');
120.  Comp.Data(1).Data{index}.raw = S.RawHolleyLog;
121.
```

```
122.  index = index+1;
123.  Comp.Data(1).Config = 1;
124.  Comp.Data(1).Platform{index} = 'LM1';
125.  Comp.Data(1).DataType{index} = 'WidebandLog';
126.  S = load('RawWBLog.mat');
127.  Comp.Data(1).Data{index}.afr = S.RawWBLog(:,1);
128.  Comp.Data(1).Data{index}.rpm = S.RawWBLog(:,2);
129.
130.
131.  return;
132.
```

loadSettings2.m

```
 1.  function [Comp] = loadSettings2()
 2.  % [Comp] = loadSettings2()
 3.  %
 4.  % Loads the data and settings from my Apr 01 drive to Buda log data
 5.  %
 6.  % Created: 6 March 2005
 7.  % Updated: 01 April 2005
 8.  %
 9.  % This file & its contents Copyright © 2005 Sam Harwell unless otherwise stated
10.
11.  Comp.Configs = struct([]);
12.  Comp.Targets = struct([]);
13.  Comp.Data = struct([]);
14.
15.  %%%%%%%%%%
16.  % config %
17.  %%%%%%%%%%
18.  Comp.Configs(1).BaseID = uint32(0);
19.  Comp.Configs(1).Platform = 'Commander950';
20.  Comp.Configs(1).ConfigType = { };
21.  Comp.Configs(1).Data = { };
22.  Comp.Configs(1).Adjustments = { };
23.
24.  % Fuel map
25.  index = uint32(1); % 1
26.  Comp.Configs(1).ConfigType{index} = 'FuelMap';
27.  Comp.Configs(1).Data{index}.rpm =
     [400;800;1000;1200;1500;1700;2000;2500;3000;3500;4000;4500;5000;5500;6000;6500];
28.  Comp.Configs(1).Data{index}.map = [19;38;56;75;94;113];
29.  OrigFuelMap = zeros( size(Comp.Configs(1).Data{index}.map,1),
     size(Comp.Configs(1).Data{index}.rpm,1) );
30.  OrigFuelMap(6,:) = [56 79 107 129 146 146 161 161 161 161 161 161 161 161 157 144]; % 113
31.  OrigFuelMap(5,:) = [56 79 107 129 146 146 161 161 161 161 161 161 161 161 157 144]; %  94
32.  OrigFuelMap(4,:) = [65 43  38  89  99  90  96 107 103 124 124 118 118 112  96  96]; %  75
33.  OrigFuelMap(3,:) = [33 37  36  43  48  46  51  55  56  89  89  89  89  83  65  65]; %  56
34.  OrigFuelMap(2,:) = [15 24  28  34  29  27  27  32  34  54  63  61  61  58  47  47]; %  38
35.  OrigFuelMap(1,:) = [ 9  9   9  17  17  16  18  15  16  34  34  30  29  26  20  20]; %  19
36.  % OrigFuelMap(6,:) = [50 70 95 115 130 130 143 143 143 143 143 143 143 143 140 128]; % 113
37.  % OrigFuelMap(5,:) = [50 70 95 115 130 130 143 143 143 143 143 143 143 143 140 128]; %  94
38.  % OrigFuelMap(4,:) = [58 47 48  79  88  80  78  89  89 110 110 105 105 100  85  85]; %  75
39.  % OrigFuelMap(3,:) = [29 38 37  50  53  48  48  52  56  79  79  79  79  74  58  58]; %  56
40.  % OrigFuelMap(2,:) = [13 27 26  34  33  30  28  31  36  48  56  54  54  52  42  42]; %  38
41.  % OrigFuelMap(1,:) = [ 8  8  8  15  15  15  17  16  18  30  30  27  26  23  18  18]; %  19
42.  Comp.Configs(1).Data{index}.fuel = OrigFuelMap;
43.  Comp.Configs(1).Adjustments{index}.fuel = ones(size(Comp.Configs(1).Data{index}.fuel));
44.  Comp.Configs(1).Adjustments{index}.ready = 0;
45.  Comp.Configs(1).Adjustments{index}.trust = zeros(size(Comp.Configs(1).Data{index}.fuel));
46.
47.  % Air temperature enrichment modifier
48.  index = index+1; % 2
49.  Comp.Configs(1).ConfigType{index} = 'AirMod';
50.  Comp.Configs(1).Data{index}.temp = [0;11;32;41;54;66;75;87;98;110;121;138;156;184;212;235];
51.  Comp.Configs(1).Data{index}.modifier =
     [89.1;90.6;93.8;95.3;97.7;99.2;100.8;101.6;103.9;106.3;106.3;109.4;111.7;115.6;118.8;121.9]
     ;
52.  % Comp.Configs(1).Data{index}.modifier =
     [125.0;125.0;123.4;121.9;118.8;115.6;114.1;110.9;109.4;107.0;104.7;101.6;99.2;94.5;90.6;87.
     5];
53.  Comp.Configs(1).Adjustments{index}.modifier =
     ones(size(Comp.Configs(1).Data{index}.modifier));
54.
55.  % Coolant temperature enrichment modifer
56.  index = index+1; % 3
57.  Comp.Configs(1).ConfigType{index} = 'CltMod';
58.  Comp.Configs(1).Data{index}.temp = [0;11;32;41;54;66;75;87;98;110;121;138;156;184;212;235];
```

```
 59.  Comp.Configs(1).Data{index}.modifier =
      [115.6;114.8;112.5;111.7;110.2;109.4;107.8;107.0;105.5;104.7;103.1;100.8;98.4;94.5;89.8;85.
      2];
 60.  % Comp.Configs(1).Data{index}.modifier =
      [106.3;103.1;100.0;100.0;100.0;100.0;100.0;100.0;100.0;100.0;100.0;100.0;100.0;100.0;
      100.0];
 61.  Comp.Configs(1).Adjustments{index}.modifier =
      ones(size(Comp.Configs(1).Data{index}.modifier));
 62.
 63.
 64.  %%%%%%%%%%
 65.  % targets %
 66.  %%%%%%%%%%
 67.  Comp.Targets(1).Platform = 'Commander950';
 68.
 69.  % SCH 14 Mar 2005: added the 0 map row to prevent NaN values from interp2
 70.  targetafr.rpm =
      [400;800;1000;1200;1500;1700;2000;2500;3000;3500;4000;4500;5000;5500;6000;6500];
 71.  targetafr.map = [0;19;38;56;75;94;113];
 72.  targetafr.afr = zeros( size(targetafr.map,1), size(targetafr.rpm,1) );
 73.  % SCH 25 Mar 2005: made more adjustments...dropped idle afr to stoich
 74.  % SCH 25 Mar 2005:    400   800   1000  1200  1500  1700  2000  2500  3000  3500  4000
      4500  5000  5500  6000  6500
 75.  targetafr.afr(7,:) = [13.00 13.00 13.00 12.20 12.20 12.20 12.20 12.20 12.20 12.20 12.20
      12.20 12.20 12.20 12.20 12.20]; % 113
 76.  targetafr.afr(6,:) = [13.00 13.00 13.00 12.20 12.20 12.20 12.20 12.20 12.20 12.20 12.20
      12.20 12.20 12.20 12.20 12.20]; %  94
 77.  targetafr.afr(5,:) = [14.00 14.00 14.00 12.20 12.20 12.20 12.20 12.20 12.20 12.20 12.20
      12.20 12.20 12.20 12.20 12.20]; %  75
 78.  targetafr.afr(4,:) = [14.68 14.68 14.68 15.50 15.50 15.50 15.00 15.00 14.30 14.30
      14.30 14.30 14.30 14.30 14.30]; %  56
 79.  targetafr.afr(3,:) = [14.68 14.68 14.68 14.68 15.40 15.40 16.00 16.00 16.00 16.00 16.00
      16.00 16.00 16.00 16.00 16.00]; %  38
 80.  targetafr.afr(2,:) = [14.68 14.68 14.68 14.68 15.40 15.40 16.00 16.00 16.00 16.00 16.00
      16.00 16.00 16.00 16.00 16.00]; %  19
 81.  targetafr.afr(1,:) = [14.68 14.68 14.68 14.68 15.40 15.40 16.00 16.00 16.00 16.00 16.00
      16.00 16.00 16.00 16.00 16.00]; %   0
 82.
 83.  % SCH 12 Mar 2005: quick update - "Fuel" is now one of the available targets. If
 84.  % specified, it means that the fuel map and all of the fuel modifiers should be
 85.  % computed for the current platform.
 86.  index = uint32(1);
 87.  Comp.Targets(1).TargetType{index} = 'Fuel';
 88.  Comp.Targets(1).Data{index}.targetafr = targetafr;
 89.  % do 100 builds leading up to the final solution to watch progress
 90.  Comp.Targets(1).Data{index}.incremental = 10;
 91.  % SCH 24 Mar 2005: number of complete passes
 92.  Comp.Targets(1).Data{index}.numpasses = 1;
 93.  % SCH 24 Mar 2005: number of modifier iterations per pass
 94.  Comp.Targets(1).Data{index}.moditerations = 1;
 95.  Comp.Targets(1).Data{index}.manualmod = 0;
 96.  % SCH 15 Mar 2005: one method for dealing with sparse cells (unsuccessful so far)
 97.  Comp.Targets(1).Data{index}.interpfill = 0;
 98.  % SCH 15 Mar 2005: another method for dealing with sparse cells (unsuccessful so far)
 99.  Comp.Targets(1).Data{index}.windowexpand = 0;
100.
101.  % SCH 12 Mar 2005: these lines just show where the results will go
102.  % Comp.Targets(1).Adjustments{index}.incremental(i).fuel;
103.  % Comp.Targets(1).Adjustments{index}.incremental(i).airmod;
104.  % Comp.Targets(1).Adjustments{index}.incremental(i).cltmod;
105.  % Comp.Targets(1).Adjustments{index}.current.fuel;
106.  % Comp.Targets(1).Adjustments{index}.current.airmod;
107.  % Comp.Targets(1).Adjustments{index}.current.cltmod;
108.
109.
110.  % index = index+1;
111.  % Comp.Targets(1).TargetType{index} = 'FuelMap';
112.  % Comp.Targets(1).Data{index}.targetafr = targetafr;
113.  %
114.  % index = index+1;
115.  % Comp.Targets(1).TargetType{index} = 'AirMod';
```

```
116.  % Comp.Targets(1).Data{index}.targetafr = targetafr;
117.  %
118.  % index = index+1;
119.  % Comp.Targets(1).TargetType{index} = 'CltMod';
120.  % Comp.Targets(1).Data{index}.targetafr = targetafr;
121.
122.  %%%%%%%%
123.  % data %
124.  %%%%%%%%
125.  index = uint32(1);
126.  Comp.Data(1).Config = 1;
127.  Comp.Data(1).Platform{index} = 'Commander950';
128.  Comp.Data(1).DataType{index} = 'RawLog';
129.  S = load('RawHolleyLog2.mat');
130.  Comp.Data(1).Data{index}.raw = S.RawHolleyLog2;
131.
132.  index = index+1;
133.  Comp.Data(1).Config = 1;
134.  Comp.Data(1).Platform{index} = 'LM1';
135.  Comp.Data(1).DataType{index} = 'WidebandLog';
136.  S = load('RawWBLog2.mat');
137.  Comp.Data(1).Data{index}.afr = S.RawWBLog2(:,1);
138.  Comp.Data(1).Data{index}.rpm = S.RawWBLog2(:,2);
139.
140.
141.  return;
142.
```

loadSettings3.m

```
 1.  function [Comp] = loadSettings3()
 2.  % [Comp] = loadSettings3()
 3.  %
 4.  % Loads the data and settings from my Apr 01 drive to Buda log data
 5.  %
 6.  % Created: 6 March 2005
 7.  % Updated: 26 April 2005
 8.  %
 9.  % This file & its contents Copyright © 2005 Sam Harwell unless otherwise stated
10.
11.  Comp.Configs = struct([]);
12.  Comp.Targets = struct([]);
13.  Comp.Data = struct([]);
14.
15.  %%%%%%%%%%
16.  % config %
17.  %%%%%%%%%%
18.  Comp.Configs(1).BaseID = uint32(0);
19.  Comp.Configs(1).Platform = 'Commander950';
20.  Comp.Configs(1).ConfigType = { };
21.  Comp.Configs(1).Data = { };
22.  Comp.Configs(1).Adjustments = { };
23.
24.  % Fuel map
25.  index = uint32(1); % 1
26.  Comp.Configs(1).ConfigType{index} = 'FuelMap';
27.  Comp.Configs(1).Data{index}.rpm =
     [400;800;1000;1200;1500;1700;2000;2500;3000;3500;4000;4500;5000;5500;6000;6500];
28.  Comp.Configs(1).Data{index}.map = [19;38;56;75;94;113];
29.  OrigFuelMap = zeros( size(Comp.Configs(1).Data{index}.map,1),
     size(Comp.Configs(1).Data{index}.rpm,1) );
30.  OrigFuelMap(6,:) = [56 79 107 129 146 146 161 161 161 161 161 161 161 161 157 144]; % 113
31.  OrigFuelMap(5,:) = [56 79 107 129 146 146 161 161 161 161 161 161 161 161 157 144]; %  94
32.  OrigFuelMap(4,:) = [49 49  59  98  98  99 101 106 106 106 106 106 106 106 106 106]; %  75
33.  OrigFuelMap(3,:) = [38 38  41  43  50  51  52  57  57  57  57  57  57  57  57  57]; %  56
34.  OrigFuelMap(2,:) = [26 26  28  35  29  30  30  34  34  34  34  34  34  34  34  34]; %  38
35.  OrigFuelMap(1,:) = [ 8  8  12  17  23  22  22  16  16  16  16  16  16  16  16  16]; %  19
36.  % OrigFuelMap(6,:) = [56 79 107 129 146 146 161 161 161 161 161 161 161 161 157 144]; % 113
37.  % OrigFuelMap(5,:) = [56 79 107 129 146 146 161 161 161 161 161 161 161 161 157 144]; %  94
38.  % OrigFuelMap(4,:) = [65 43  38  89  99  90  96 107 103 124 124 118 118 112  96  96]; %  75
39.  % OrigFuelMap(3,:) = [33 37  36  43  48  46  51  55  56  89  89  89  89  83  65  65]; %  56
40.  % OrigFuelMap(2,:) = [15 24  28  34  29  27  27  32  34  54  63  61  61  58  47  47]; %  38
41.  % OrigFuelMap(1,:) = [ 9  9   9  17  17  16  18  15  16  34  34  30  29  26  20  20]; %  19
42.  % OrigFuelMap(6,:) = [50 70 95 115 130 130 143 143 143 143 143 143 143 143 140 128]; % 113
43.  % OrigFuelMap(5,:) = [50 70 95 115 130 130 143 143 143 143 143 143 143 143 140 128]; %  94
44.  % OrigFuelMap(4,:) = [58 47 48  79  88  80  78  89  89 110 110 105 105 100  85  85]; %  75
45.  % OrigFuelMap(3,:) = [29 38 37  50  53  48  48  52  56  79  79  79  79  74  58  58]; %  56
46.  % OrigFuelMap(2,:) = [13 27 26  34  33  30  28  31  36  48  56  54  54  52  42  42]; %  38
47.  % OrigFuelMap(1,:) = [ 8  8  8  15  15  15  17  16  18  30  30  27  26  23  18  18]; %  19
48.  Comp.Configs(1).Data{index}.fuel = OrigFuelMap;
49.  Comp.Configs(1).Adjustments{index}.fuel = ones(size(Comp.Configs(1).Data{index}.fuel));
50.  Comp.Configs(1).Adjustments{index}.ready = 0;
51.  Comp.Configs(1).Adjustments{index}.trust = zeros(size(Comp.Configs(1).Data{index}.fuel));
52.
53.  % Air temperature enrichment modifier
54.  index = index+1; % 2
55.  Comp.Configs(1).ConfigType{index} = 'AirMod';
56.  Comp.Configs(1).Data{index}.temp = [0;11;32;41;54;66;75;87;98;110;121;138;156;184;212;235];
57.  Comp.Configs(1).Data{index}.modifier =
     [89.1;90.6;93.8;95.3;97.7;99.2;100.8;101.6;103.9;106.3;106.3;109.4;111.7;115.6;118.8;121.9]
     ;
58.  % Comp.Configs(1).Data{index}.modifier =
     [125.0;125.0;123.4;121.9;118.8;115.6;114.1;110.9;109.4;107.0;104.7;101.6;99.2;94.5;90.6;87.
     5];
59.  Comp.Configs(1).Adjustments{index}.modifier =
     ones(size(Comp.Configs(1).Data{index}.modifier));
60.
```

```
% Coolant temperature enrichment modifer
index = index+1; % 3
Comp.Configs(1).ConfigType{index} = 'CltMod';
Comp.Configs(1).Data{index}.temp = [0;11;32;41;54;66;75;87;98;110;121;138;156;184;212;235];
Comp.Configs(1).Data{index}.modifier =
[115.6;114.8;112.5;111.7;110.2;109.4;107.8;107.0;105.5;104.7;103.1;100.8;98.4;94.5;89.8;85.
2];
% Comp.Configs(1).Data{index}.modifier =
[106.3;103.1;100.0;100.0;100.0;100.0;100.0;100.0;100.0;100.0;100.0;100.0;100.0;100.0;
100.0];
Comp.Configs(1).Adjustments{index}.modifier =
ones(size(Comp.Configs(1).Data{index}.modifier));


%%%%%%%%%%
% targets %
%%%%%%%%%%
Comp.Targets(1).Platform = 'Commander950';

% SCH 14 Mar 2005: added the 0 map row to prevent NaN values from interp2
targetafr.rpm =
[400;800;1000;1200;1500;1700;2000;2500;3000;3500;4000;4500;5000;5500;6000;6500];
targetafr.map = [0;19;38;56;75;94;113];
targetafr.afr = zeros( size(targetafr.map,1), size(targetafr.rpm,1) );
% SCH 25 Mar 2005: made more adjustments...dropped idle afr to stoich
% SCH 25 Mar 2005:    400   800   1000  1200  1500  1700  2000  2500  3000  3500  4000
4500  5000  5500  6000  6500
targetafr.afr(7,:) = [13.00 13.00 13.00 12.20 12.20 12.20 12.20 12.20 12.20 12.20 12.20
12.20 12.20 12.20 12.20 12.20]; % 113
targetafr.afr(6,:) = [13.00 13.00 13.00 12.20 12.20 12.20 12.20 12.20 12.20 12.20 12.20
12.20 12.20 12.20 12.20 12.20]; %  94
targetafr.afr(5,:) = [14.00 14.00 14.00 12.20 12.20 12.20 12.20 12.20 12.20 12.20 12.20
12.20 12.20 12.20 12.20 12.20]; %  75
targetafr.afr(4,:) = [14.68 14.68 14.68 14.68 15.50 15.50 15.50 15.00 15.00 14.30 14.30
14.30 14.30 14.30 14.30 14.30]; %  56
targetafr.afr(3,:) = [14.68 14.68 14.68 14.68 15.40 15.40 16.00 16.00 16.00 16.00 16.00
16.00 16.00 16.00 16.00 16.00]; %  38
targetafr.afr(2,:) = [14.68 14.68 14.68 14.68 15.40 15.40 16.00 16.00 16.00 16.00 16.00
16.00 16.00 16.00 16.00 16.00]; %  19
targetafr.afr(1,:) = [14.68 14.68 14.68 14.68 15.40 15.40 16.00 16.00 16.00 16.00 16.00
16.00 16.00 16.00 16.00 16.00]; %   0

% SCH 12 Mar 2005: quick update - "Fuel" is now one of the available targets. If
% specified, it means that the fuel map and all of the fuel modifiers should be
% computed for the current platform.
index = uint32(1);
Comp.Targets(1).TargetType{index} = 'Fuel';
Comp.Targets(1).Data{index}.targetafr = targetafr;
% do 100 builds leading up to the final solution to watch progress
Comp.Targets(1).Data{index}.incremental = 1;
% SCH 24 Mar 2005: number of complete passes
Comp.Targets(1).Data{index}.numpasses = 1;
% SCH 24 Mar 2005: number of modifier iterations per pass
Comp.Targets(1).Data{index}.moditerations = 0;
Comp.Targets(1).Data{index}.manualmod = 0;
% SCH 15 Mar 2005: one method for dealing with sparse cells (unsuccessful so far)
Comp.Targets(1).Data{index}.interpfill = 0;
% SCH 15 Mar 2005: another method for dealing with sparse cells (unsuccessful so far)
Comp.Targets(1).Data{index}.windowexpand = 0;

% SCH 12 Mar 2005: these lines just show where the results will go
% Comp.Targets(1).Adjustments{index}.incremental(i).fuel;
% Comp.Targets(1).Adjustments{index}.incremental(i).airmod;
% Comp.Targets(1).Adjustments{index}.incremental(i).cltmod;
% Comp.Targets(1).Adjustments{index}.current.fuel;
% Comp.Targets(1).Adjustments{index}.current.airmod;
% Comp.Targets(1).Adjustments{index}.current.cltmod;


% index = index+1;
% Comp.Targets(1).TargetType{index} = 'FuelMap';
```

```
% Comp.Targets(1).Data{index}.targetafr = targetafr;
%
% index = index+1;
% Comp.Targets(1).TargetType{index} = 'AirMod';
% Comp.Targets(1).Data{index}.targetafr = targetafr;
%
% index = index+1;
% Comp.Targets(1).TargetType{index} = 'CltMod';
% Comp.Targets(1).Data{index}.targetafr = targetafr;

%%%%%%%%
% data %
%%%%%%%%
index = uint32(1);
Comp.Data(1).Config = 1;
Comp.Data(1).Platform{index} = 'Commander950';
Comp.Data(1).DataType{index} = 'RawLog';
S = load('RawHolleyLog3.mat');
Comp.Data(1).Data{index}.raw = S.RawHolleyLog3;

index = index+1;
Comp.Data(1).Config = 1;
Comp.Data(1).Platform{index} = 'LM1';
Comp.Data(1).DataType{index} = 'WidebandLog';
S = load('RawWBLog3.mat');
Comp.Data(1).Data{index}.afr = S.RawWBLog3(:,1);
Comp.Data(1).Data{index}.rpm = S.RawWBLog3(:,2);


return;
```

APPENDIX E. MEAN-SQUARE-ERROR CALCULATION CODE

afrmse.m

```
 1.  function [original_mse,adjusted_mse]=afrmse(Comp,maxidx)
 2.      %  [adjusted_mse]=afrmse(Comp);
 3.      %
 4.      %  takes in the Comp structure using the car calculator and outputs the
 5.      %  mse between the target afr and the modified afr with our actual
 6.      %  measurements
 7.
 8.  if nargin<2
 9.      maxidx = length(Comp.Data(1).Data{1}.afr);
10.  end
11.
12.  %the following code was copied from the plot_adjusted.m code written earlier by SCH
13.  afr = Comp.Data(1).Data{1}.afr(1:maxidx);
14.  afrt = Comp.Data(1).Data{1}.afrt(1:maxidx);
15.  rpm = Comp.Data(1).Data{1}.rpm(1:maxidx);
16.  map = Comp.Data(1).Data{1}.map(1:maxidx);
17.  mat = Comp.Data(1).Data{1}.mat(1:maxidx);
18.  clt = Comp.Data(1).Data{1}.clt(1:maxidx);
19.
20.  % SCH 25 Mar 2005: get the target AFR
21.  targetafr = Comp.Targets(1).Data{1}.targetafr;
22.
23.  % bicubic (because we can) interpolation get the afr targets from the rpm and
24.  % map values
25.  % t subscripts mean values from the target afr map
26.  Xt = targetafr.rpm;
27.  Yt = targetafr.map;
28.  Zt = targetafr.afr;
29.  Xi = rpm;
30.  Yi = map;
31.  % SCH 14 Mar 2005: TODO: Account for possible NaN outputs here
32.  %tafr = interp2(Xt,Yt,Zt,Xi,Yi,'linear');
33.
34.
35.  % SCH 25 Mar 2005: get the estimated AFR after adjustments
36.  % tafr * olds / news
37.
38.      % y values of the data (rpm here)
39.      Yi = rpm;
40.      % x values of the data (map here)
41.      Xi = map;
42.      % SCH 11 Mar 2005: the ofuel/nfuel ratio might not be 1
43.      Yt = Comp.Configs(1).Data{1}.rpm;
44.      Xt = Comp.Configs(1).Data{1}.map;
45.      Zt = Comp.Configs(1).Data{1}.fuel';
46.      ofuel = interp2(Xt,Yt,Zt,Xi,Yi,'linear',-1);
47.      Zt = Comp.Targets(1).Adjustments{1}.current.fuel';
48.      nfuel = interp2(Xt,Yt,Zt,Xi,Yi,'linear',-1);
49.
50.      % tick marks on the x axis (temp bins)
51.      Xt = Comp.Configs(1).Data{2}.temp;
52.      % original modifier map
53.      Yt = Comp.Configs(1).Data{2}.modifier;
54.      oair = interp1q(Xt, Yt, mat);
55.      Yt = Comp.Targets(1).Adjustments{1}.current.airmod;
56.      nair = interp1q(Xt, Yt, mat);
57.
58.      % tick marks on the x axis (temp bins)
59.      Xt = Comp.Configs(1).Data{3}.temp;
60.      % original modifier map
61.      Yt = Comp.Configs(1).Data{3}.modifier;
62.      oclt = interp1q(Xt, Yt, clt);
63.      Yt = Comp.Targets(1).Adjustments{1}.current.cltmod;
64.      nclt = interp1q(Xt, Yt, clt);
65.
66.  nafr = afr .* ofuel .* oair .* oclt ./ ( nfuel .* nair .* nclt );
67.
```

```
68.  original_mse=mse(afr-afrt);
69.  adjusted_mse=mse(nafr-afrt);
```