

# **Compressor Performance Map Generation and Testing per SAE J1723**

A Senior Project  
presented to  
the Faculty of the Aerospace Engineering Department  
California Polytechnic State University, San Luis Obispo

In Partial Fulfillment  
of the Requirements for the Degree  
Bachelor of Science

by

Jeffrey Lee Freeman

February, 2011

A MATLAB program was written to plot compressor performance maps for a set of test data that was collected in accordance with SAE J1723 at Vortech Engineering, Inc. Paxton Automotive Corp.'s N2500 supercharger was used as a case example for the program, which was carried through from test stand installation to finalized compressor performance map. A sequence was also developed to interpolate the efficiency of the compressor for a given operational setting. The program was shown to be a great improvement from the previously applied technique for accomplishing the same tasks; it is more accurate in plotting the given data, and the time spent performing the process is reduced by approximately seven hours.

## Nomenclature

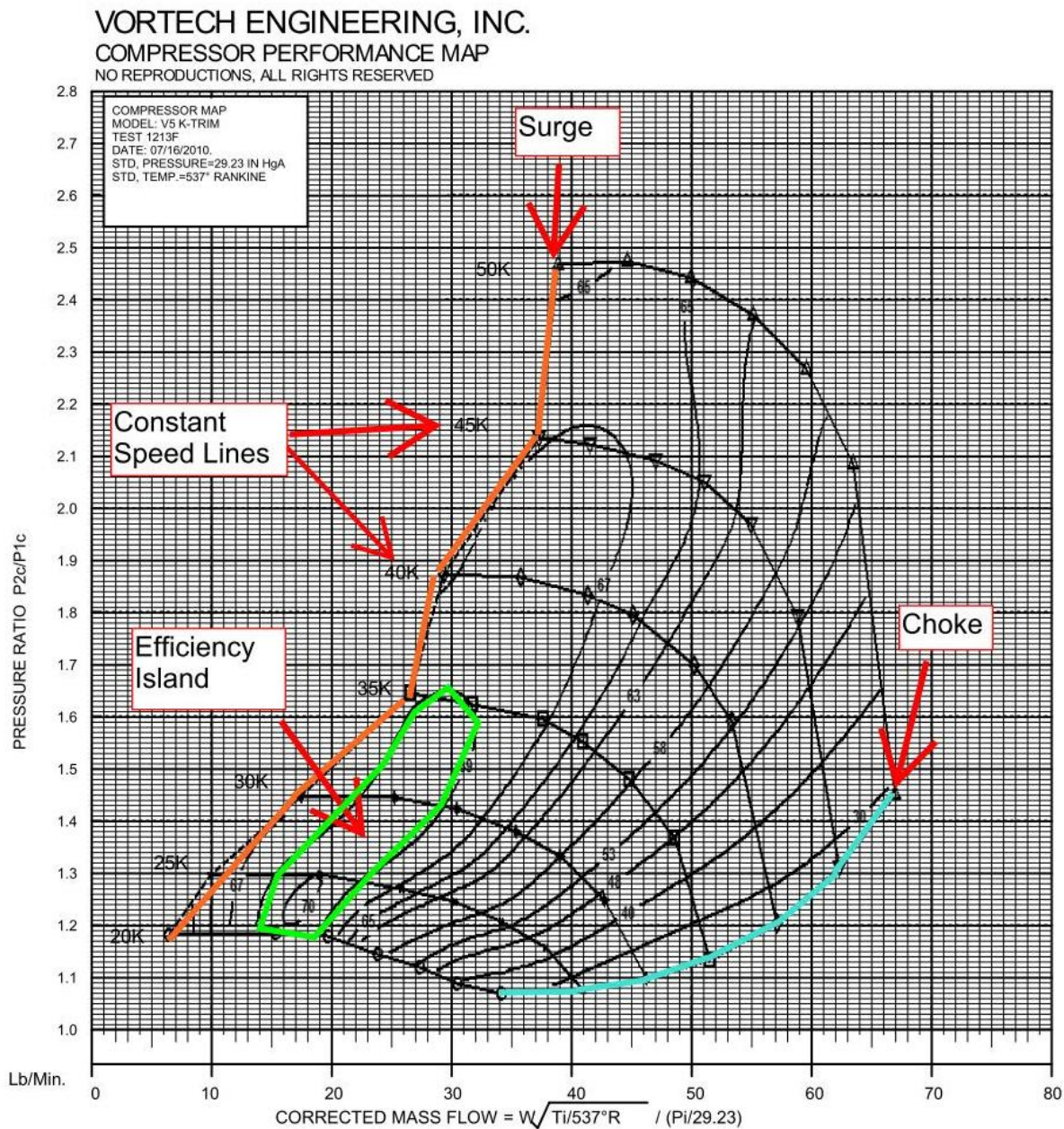
$A$	= area [ $in.^2$ ]
$C_P$	= specific heat at constant pressure of air, $6008.065 \text{ ft} - \text{lb}_f/\text{slug} - R$
$D$	= pipe inner diameter [ $in$ ]
$P$	= pressure [ $in. Hg, psi$ ]
$PR$	= pressure ratio, $P_{o,2}/P_{o,1}$
$Q$	= volumetric flow rate [ $ft^3/min$ ]
$T$	= temperature [ $^{\circ}T, R$ ]
$W$	= SAE corrected mass flow rate [ $lb_m/min$ ]
$\dot{m}$	= observed mass flow rate [ $lb_m/min$ ]
$v$	= velocity [ $ft/sec$ ]
$\rho$	= density [ $lb_m/ft^3$ ]
$\eta$	= isentropic efficiency [%]
<i>Subscript</i>	
$o$	= total, stagnation condition
$1$	= inlet section
$2$	= discharge section
$baro$	= barometric (station)
$BM$	= bell mouth

## I. Introduction

AIR compressors are engine components whose sole purpose is to increase the pressure of the air that passes through them. By pressurizing the air prior to the intake of an engine, the fuel is capable of combusting more rapidly and effectively. Compressors are used in the transportation industry both for jet engines and as automobile superchargers. The three main metrics of performance for an air compressor are the range of mass flow rate,  $W$ , and pressure ratio,  $PR$ , in which it can operate and the isentropic efficiency,  $\eta$  at which it does so. The consumer has a particular interest in finding an air compressor that performs efficiently at the range of  $W$  and  $PR$  in which the engine operates. This report focuses primarily on automotive superchargers, but the processes could be applied without much modification to jet engine air compressors as well. Particularly, the equations used apply to centrifugal, nonpositive displacement superchargers.

In order to determine whether to use a particular compressor, the consumer will typically ask to view a compressor performance map, which is a contour plot comparing the three main performance metrics. Figure 1 highlights the key items of importance in one of these maps. The lines that start horizontal on the left and droop to vertical on the right are constant speed lines, where the speed of rotation for the impeller is constant. As the speed increases, so too does the pressure ratio and the potential mass flow rate. A supercharger's performance can fluctuate for a given speed based on the amount of restriction present in the pipes after its discharge. The orange line on the left represents surge, which is defined by severe air flow reversal.<sup>1</sup> Essentially, surge is the highest pressure and lowest mass flow rate that the supercharger can produce at a given speed, and it is caused by the greatest amount of blockage in the discharge flow. On the other end of the constant speed line is a blue line which represents choke, which occurs when no blockage is present in the discharge flow. A supercharger operating at choke is producing little to no pressure rise, but moves a maximum amount of air. The green shape in the middle is a peak efficiency island, which defines the region of mass flow rate and pressure ratio for which the compressor is most efficient. Knowing this

region also tells the consumer what speed the supercharger should operate at for optimal performance and, hence, how to design the gearbox. The location and magnitude of the peak efficiency island is easily one of the most important aspects of a supercharger in that if everything else is the same, the consumer will select the product with the higher efficiency and/or the one which is most efficient at the desired operating conditions.



**Figure 1: A typical compressor performance map highlights key features of its corresponding compressor.<sup>2</sup>**

## A. SAE J1723

The Society of Automotive Engineers (SAE) created standard number J1723<sup>1</sup> in an effort to create a method of comparison for all superchargers on the market. This standard specifies the equipment that should be used while testing superchargers for their performance, basic definitions of each of the three performance

parameters, corrective formulae so the elevation and atmospheric conditions during the test do not affect the end result, and methods for presenting the results.

## **B. Preliminary State of Affairs**

Before this project, compressor performance map generation at Vortech Engineering, Inc. was a slow, unpleasant experience whose final product often misrepresented the calculated performance of the supercharger. Two waves of technology had already passed through the company for this task; the first was hand drawn and the second was computer generated using splines in Vellum. Splines are a drawing tool in which the user can set the location and slope of a curve at multiple key points. With anywhere between ten and twenty long splines that needed to match a data field exactly, this method took one person an entire day of headaches and frustration struggling to get the curves to pan out the way he or she wanted them to, and the accuracy to which the end result matched the data from which it was derived relied entirely on human capabilities. This method will be referred to as the spline method.

## **C. Objective**

This project was expected to improve upon the spline method for generating compressor performance maps in a way that maintained the smoothness of the curves. The key items for improvement were speed of generation and accuracy of the final product, and both could be achieved by using a computational plotter such as MATLAB. Furthermore, a numerical method for extracting the efficiency of the supercharger for a particular mass flow rate and pressure ratio was desired so that a tested compressor could then be used in an engine simulation.

# **II. Apparatus and Procedure: Hardware**

This section discusses the tools and techniques used to produce the raw test data for an air compressor. Most of these were already established before the current project began and they are listed here, courtesy of Vortech Engineering, Inc., to allow for future reproduction of the test results.

## **A. Test Unit Installation**

The test unit consisted of the test stand and the control panel. A simplified rendition of the test stand is shown in Fig. 2. Ambient air entered through the calibrated flow nozzle (bell mouth), was pressurized by the supercharger, and was then discharged. The supercharger was powered by a 250 HP electric motor, and the temperature was regulated by an active oil system. The flow rate was regulated by a set of control valves downstream from the discharge section. Pressure and temperature measurements were taken at the bell mouth, inlet section, and discharge section according to SAE J1723. Compressor speed and torque were also recorded, along with a number of oil temperatures and pressures that were used to ensure that the setup was operating appropriately.

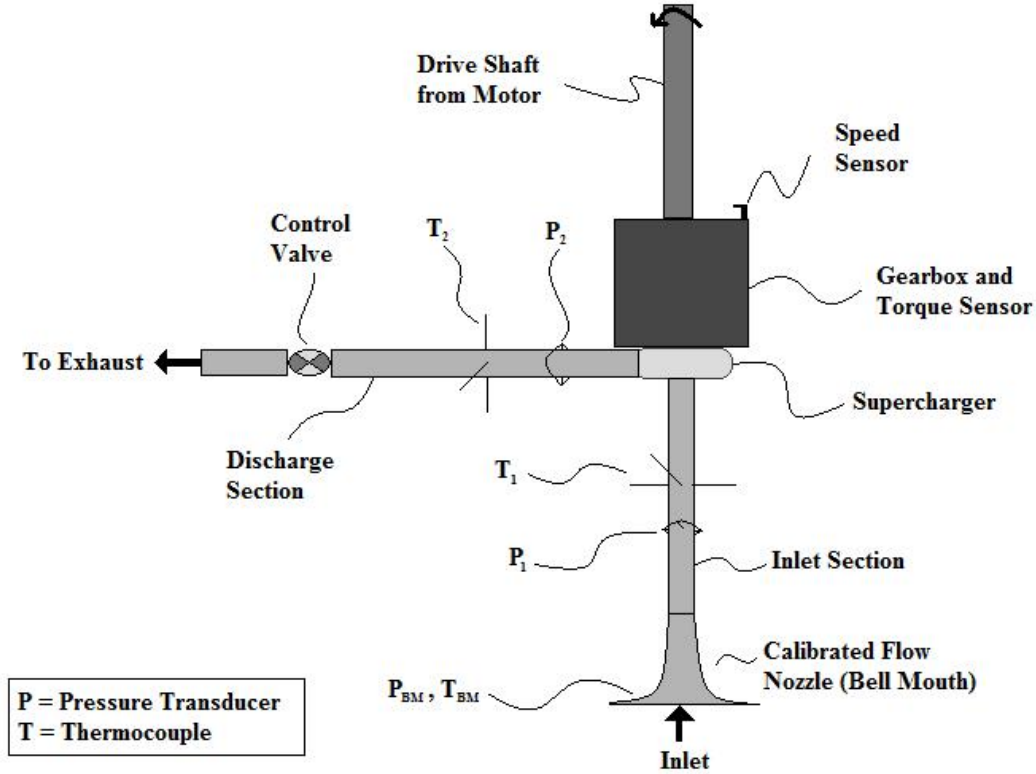
### *1. Permanent Structures*

The supercharger was run by a 250 HP electric motor that ran a driveshaft into the gearbox, which had a step-up ratio of 3.5. Oil was used as an active cooling system to regulate the temperature and lubricate the supercharger, motor, and gearbox. The shaft speed was measured using the laser sensor shown in Fig. 3. All of these items were permanent additions to the test room.

### *2. Supercharger*

The supercharger discussed for the purposes of this report was a Paxton Automotive Corp. model N2500 (Paxton is a subsidiary of Vortech Engineering, Inc.). The N2500 is a centrifugal compressor which, according to the language of SAE J1723, is a nonpositive displacement supercharger. This is in contrast to positive displacement superchargers through which a set amount of volume is displaced with each revolution.

With the gearbox and motor permanently mounted into the test room, the supercharger was the first object to be installed for any particular test. First, it was mounted onto a backing plate and indicated to its surface in order to make it flush and level. A tolerance of approximately 0.008 in. total deviation from



center was allowed. Next, the backing plate with supercharger attached was mounted onto the gearbox with special care to align the drive-spline. With the supercharger firmly in place, the oil connections and air ports could be established. Figure 4 shows a mounted supercharger that is ready to test.

### 3. Inlet Section

The inlet pipe was connected directly to the the inlet of the supercharger using silicon-rubber hose fittings. It spanned between the calibrated flow nozzle and the supercharger inlet. In the streamwise direction, it first had a pressure tap and then a set of three thermocouples. This inner diameter of the inlet section was sized in correspondance with SAE J1723.

### 4. Calibrated Flow Nozzle

The calibrated flow nozzle, or bell mouth, was used to intake air with minimal pressure loss and to present a repeatable method for determining the flow rate through the supercharger. It was mounted onto the end of the inlet pipe. The size of the nozzle was set by the size of the inlet pipe. In this experiment, the bell mouth had a throat area of  $38.760 \text{ in}^2$ . Temperature was measured in front of the inlet and the difference in pressure accross the nozzle was measured using a manometer. A picture of the flow nozzle used in this test with the corresponding sensors is shown in Fig. 5.

### 5. Discharge Section

The discharge pipe was approximately the same size as the discharge orifice of the supercharger and made of steel to withstand the high temperatures associated with air compression. As with the inlet section, there was a pressure tap upstream and a set of three thermocouples downstream to measure the discharge pressure and temperature. The entire discharge section needed to be secured with nylon webbing that was

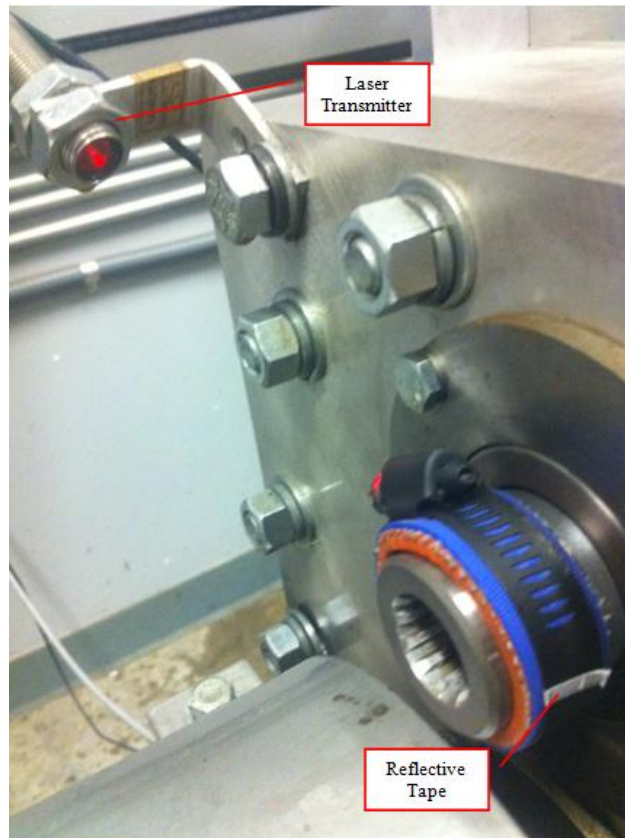


Figure 3: This laser sensor was used to measure shaft rotation speed.<sup>2</sup>

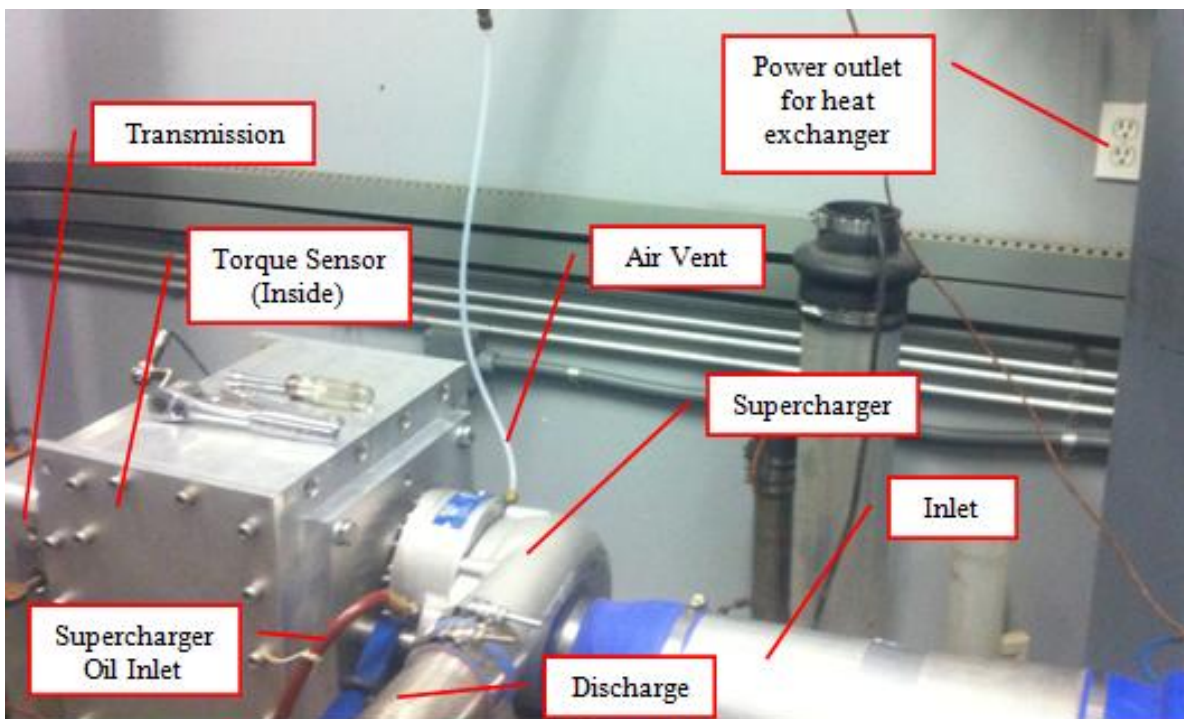


Figure 4: The supercharger in the test cell is well hidden by other test equipment.<sup>2</sup>

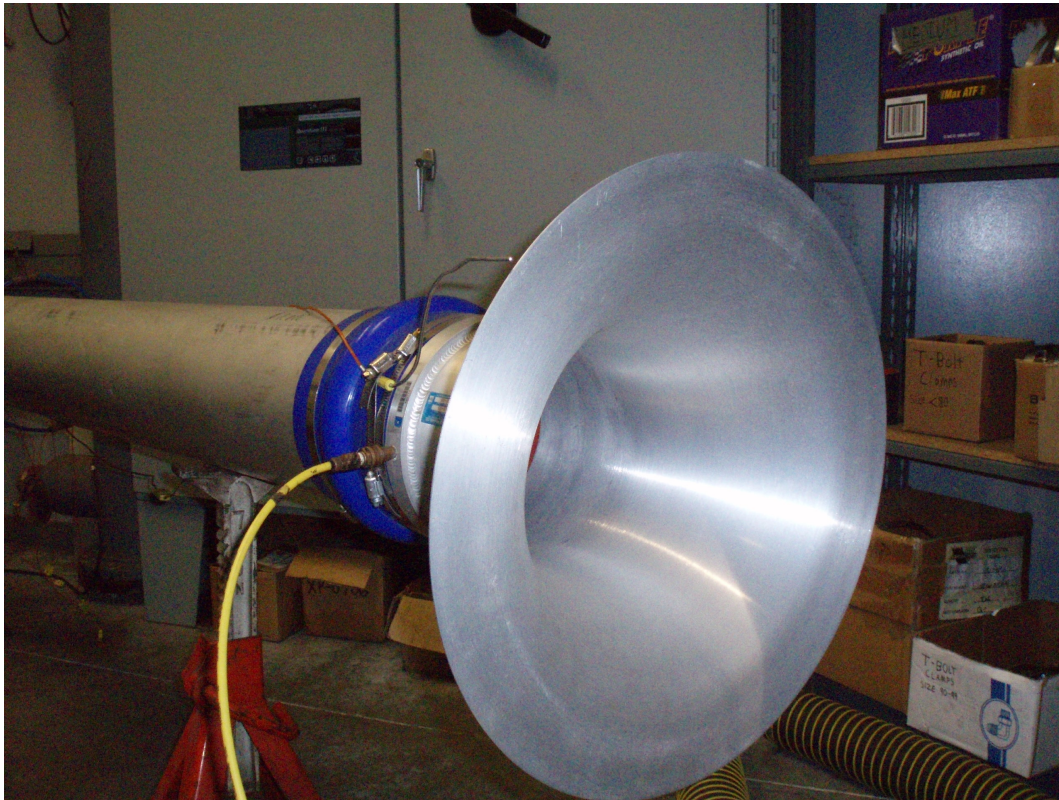


Figure 5: The calibrated flow nozzle acted as an inlet for the system.<sup>2</sup>

rated for temperatures in excess of  $350^{\circ}F$  because violent vibrations and high temperatures occurred during high-speed surge conditions.

At the end of the discharge section, there was a pair of control valves that were used to regulate the flow rate within a particular speed. One valve was actuated by a button with slow reaction time, and it controlled a majority of the area. The other valve was actuated with a knob. This smaller valve was used for fine tuning near surge conditions.

#### 6. Calibration

Each sensor had to be calibrated to ensure that it operated within the tolerances specified by SAE J1723. Specifically, torque must be measured within  $\pm 0.5\%$ , supercharger shaft speed within  $\pm 0.2\%$ , temperature within  $\pm 1^{\circ}C$ , pressures within  $\pm 0.5 kPa$ , and air flow rate within  $\pm 1\%$  of the measured value. The torque, speed, pressure, and air flow rate sensors were all calibrated by the manufacturer and inspected as needed to ensure quality. The thermocouples were calibrated on site every time they were installed. This was accomplished by letting the thermocouples come to equilibrium in an insulated cup of ice water. If the thermocouple read  $\pm 1^{\circ}C$ , then it passed the go-nogo test and could be included in the apparatus.

#### B. Test Procedure

After the hardware was installed, the actual test could begin. The oil pump and temperature regulator were turned on along with the room fan and the electric motor. Hearing protection was used by the operators. Right before the supercharger was started, the speed sensor was tared. Throughout the entire test, four items were checked to avoid emergency: first, oil temperature was not to exceed  $250^{\circ}F$ , second, the oil pressure was to be relatively constant at around 45 psig, third, the engine load must not reach the red line, and fourth, the bell mouth pressure was to be kept low enough so the meniscus in the manometer would not overflow.

Because the system was regulated by oil, it needed to be warmed up appropriately to not damage any

of the equipment. This was accomplished by stepping the speed from 0 to the first test speed in intervals of 5,000 rpm with approximately 5 minutes between each transition.

The test itself involved recording a collection of data for a predetermined number of nearly equally spaced control valve settings ranging from surge to choke for several supercharger speeds. The test presented in this report used seven control valve settings and seven speeds, but any set of numbers greater than 1 could work as long as the number of control valve settings does not vary from speed to speed. A denser concentration of data points tested yielded greater accuracy but came at the cost of more time spent performing the test.

The minimum amount of data recorded for each point included the following: impeller shaft speed (supercharger speed),  $n$ , barometric pressure,  $P_{baro}$ , ambient (bell mouth) temperature,  $T_{BM}$ , three readings of inlet section temperature,  $T_1$ , three readings of discharge temperature,  $T_2$ , change in pressure across the bell mouth,  $\Delta P_{BM}$ , inlet section pressure,  $P_1$ , discharge pressure,  $P_2$ , bell mouth area,  $A_{BM}$ , inlet pipe diameter,  $D_1$ , discharge pipe diameter,  $D_2$ , and torque. Additionally, oil pressures and temperatures were also recorded to monitor the apparatus. The torque was not used in the primary calculations but was used for troubleshooting as it provided another method for finding the efficiency of the supercharger through mechanical power. Barometric pressure was determined no more frequently than at the beginning of each speed run, but usually occurred only once in the morning and once in the afternoon because variations throughout the day were minute. Impeller shaft speed did not need to be exact because of the relatively large tolerances, so the recorded speed was rounded to the nearest thousand rpm. It was also necessary to record the area of the calibrated flow nozzle and the inner diameters of the inlet and discharge pipes,  $D_1$  and  $D_2$ , respectively.

### 1. Finding Barometric Pressure

A small sequence of events was necessary for determining the barometric pressure. First, the current, local altimeter pressure was found online for the naval base at Pt. Mugu.<sup>3</sup> Then the altimeter pressure was converted to station pressure according to the altitude of approximately 2m using an online converter.<sup>4</sup> Finally, the station pressure was recorded with the other raw data.

## III. Computational Methodology

The MATLAB sequence, named *CPM* (Compressor Performance Mapper), was broken down into three steps. First, the raw data was used to calculate the three main performance parameters in a code called *CPM\_compute.m*. Then, those parameters were plotted. As somewhat of a separate step, the plotted results could be interpolated to find the efficiency at any point in the domain. Both of these portions were accomplished by the main code, called *CPM.m*.

### A. Finding Performance Metrics from Raw Data

#### 1. Preliminary Calculations

Before the main performance parameters could be calculated, the total pressures and temperatures were needed. It was important to modify these equations as needed to make the units match appropriately. First, the inlet and discharge temperatures were taken as the average of the three recordings at each location and then converted to Rankine. All of the pressures were absolute and calculated in inches of mercury. The inlet gauge pressure was converted to absolute using

$$P_1 = P_{baro} - \frac{P_{1,gauge}}{13.608} \quad (1)$$

A specialized form of the YFCA equation was used to find the mass flow rate through the calibrated flow nozzles.<sup>5</sup> It assumes that the velocity of approach factor is equal to 1 because the pipe diameter approximates infinity for the shape used. Furthermore, it assumes that the coefficient of discharge multiplied by the adiabatic expansion factor is approximately equal to 2.06. These assumptions resulted in a simple and useful equation for finding mass flow rate,

$$\dot{m} = 2.06 A_{BM} \sqrt{\left(17.35 \frac{P_{baro}}{T_{BM}}\right) \Delta P_{BM}} \quad (2)$$

With the density at any location known by the temperature and pressure through the ideal gas law, the volumetric flow rate could be found using

$$Q = \frac{\dot{m}}{\rho} \quad (3)$$

Next, dimensional analysis shows that the velocity at any point in the apparatus was known from

$$v = \frac{Q}{A} \quad (4)$$

where the area of circular cross sections was

$$A = \pi (D/2)^2 \quad (5)$$

Finally, the total pressure was derived from Bernoulli's Equation,

$$P_o = P + \frac{\rho v^2}{2} \quad (6)$$

and the total temperature was derived from the First Law of Thermodynamics,

$$T_o = T + \frac{v^2}{2C_P} \quad (7)$$

## 2. SAE Defined Performance Parameters<sup>1</sup>

With the mass flow rate and total pressures and temperatures readily available, the equations provided by SAE J1723 were used to find the performance parameters. The SAE corrected mass flow rate,  $W$ , was found using

$$W = \dot{m} \left( \frac{29.236 \text{ in. Hg}}{P_{baro}} \right) \sqrt{\frac{T_1}{537^\circ R}} \quad (8)$$

the pressure ratio was simply

$$PR = \frac{P_{o,2}}{P_{o,1}} \quad (9)$$

and the isentropic efficiency was

$$\eta = \frac{T_{o,1} PR^{0.286} - T_{o,1}}{T_{o,2} - T_{o,1}} * 100\% \quad (10)$$

## B. Plotting the Map

The basic set of performance parameters could not be simply plotted using MATLAB's *contour* function because it created sharp corners where rounded edges were expected. An example of what this inadequate method produces is shown in Fig. 6. Therefore, an alternate method was desired for appropriately smoothing the corners while still fitting the physical phenomenon that the data represented. Essentially, the program needed to predict where the curves should be placed for the given data in a way that matched the results people produced in the past when they drew the maps by hand. It also needed to be capable of filling in an arbitrary domain which may be convex or concave.

The solution was John D'Errico's *gridfit.m*, which uses the gradient of the data to fill in a number of nodes within a rectangle encompassing the domain while balancing smoothness with accuracy.<sup>6</sup> Its *smooth* parameter allowed the user to define the level to which the result preferred smoothness or accuracy; essentially, it added the "human judgement" input. The only undesirable aspect of this program was how the contours within a convex portion of the domain were modified near the edges in order to extrapolate to the edge of the rectangle. This particular drawback was deemed acceptable because the consumer is more interested in the peak efficiency island than the performance at the boundaries of operation.

The output of *gridfit.m* is a set of matrices corresponding to the  $x$ ,  $y$ , and  $z$  axes that are ready to be used by *contour* to plot a smooth surface of rectangular cross-section. This rectangle must next be trimmed to show only the original domain, and more of D'Errico's functions are used for this purpose.<sup>7</sup> First, the perimeter of the domain is collected into a pair of vectors which are triangulated into a simplicial complex

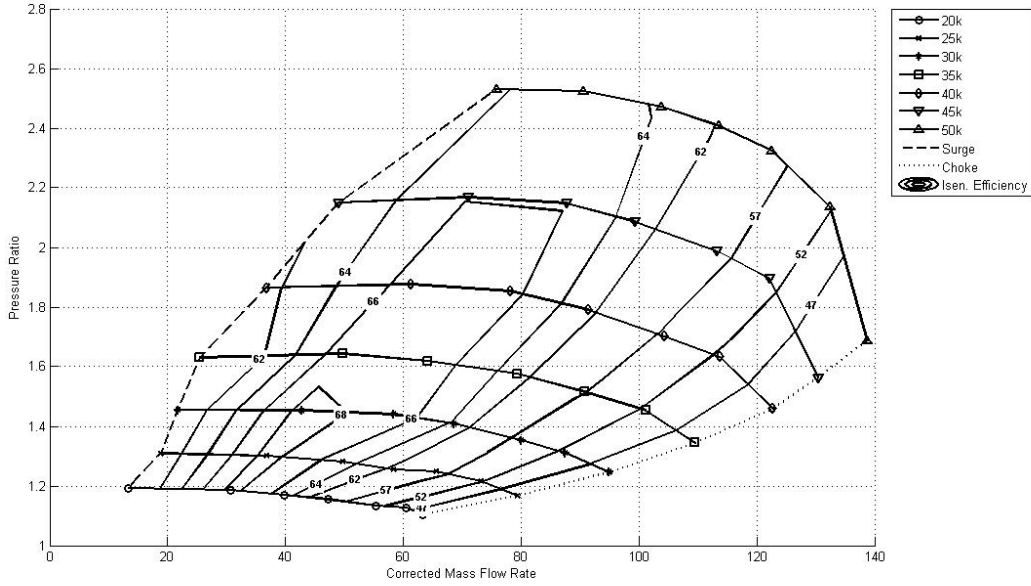


Figure 6: The sharp corners from MATLAB's *contour* function were undesirable.<sup>2</sup>

using *poly2tri.m*. Then the complete collection of nodes produced by *gridfit.m* are checked by *insc.m*, which determines whether a data point is inside the triangulated polygon or not. Any points in the grid that lie outside of the polygon are erased. The functions *simplicialcomplex.m* and *buildbccdata.m* are used by *poly2tri.m* and *insc.m*.

Finally, the figure is created with a combination of several plots. The trimmed grid data is used to create efficiency curves using *contour*, the speed lines are drawn as separate two-dimensional plots, and the surge and choke lines are added in a similar fashion. MATLAB's *clabel* can be used to add efficiency labels to the plot and *legend* is used to produce a legend calling out the different components of the figure.

### C. Interpolation: Numerically Reading the Plot

The use of *gridfit.m* also made interpolating between the data points rather easy because its output conforms to the requirements for MATLAB's *interp2*, which performs bilinear interpolation within a grid. For each desired interpolation point (W, PR), the point is first checked to be within the domain using *insc.m*, and then is compared to the grid-shaped, smooth data to find the matching efficiency.

## IV. Producing the Map

### A. Necessary Programs

The method presented in this report that was used to produce supercharger performance maps needed the following software: MATLAB, a text editor such as Microsoft Notepad, an image editor such as Microsoft Paint, and a CAD program such as Vellum. Each program could be replaced by another program with similar functions. For instance the Microsoft programs could be substituted by their Macintosh counterparts, or MATLAB could be replaced (with some modification to the codes) by Octave.

### B. Data Input

After the test was finished, a large amount of data had to be input into MATLAB in order to plot the performance map. This was done by copying all of the data into a text file with 15 columns, one for each of the pieces of information recorded at each point (See Appendix A). The first column, title "r.s", was a two-digit number where the first digit denoted the index for the impeller shaft speed, or "speed Run" where 1

was the first speed run, 2 was the second and so on. Meanwhile, the second digit denoted in a similar fashion the control valve “Setting”, where 1 was the first setting for each run (surge), 2 was the second setting, and so on. The filename and the first line of the file were the same and defined the test number for in-house bookkeeping. This input file was read by MATLAB through a code, named *CPM\_read\_data.m*, which uses Gerald Recktenwald’s *readColData.m*.<sup>8</sup> These codes collectively convert the data into a MATLAB structure, where each element of the structure contained all the information associated with one speed/setting configuration.

### C. Running the MATLAB Program

The entire collection of MATLAB codes could be run through the script titled *CPM\_Call.m*. The variables that could be changed are titled *fname*, *smooth*, *plot\_on*, *legstring*, *clabel\_on*, *interp\_on*, *XI*, *wi*, and *pri*. *fname* is a string which is equal to the filename of the input file. *smooth* is a parameter that is used by John D’Errico’s *gridfit.m* to set the balance between smoothness and accuracy of the efficiency contours.<sup>6</sup> *plot\_on*, *clabel\_on*, and *interp\_on* are all logical operators equal to either 1 for on or 0 for off. *plot\_on* determines whether or not to plot a map, *clabel\_on* defines whether or not to include contour labels in the plot, and *interp\_on* defines whether or not to numerically “read” the map by interpolating for the efficiency of a particular mass flow rate/pressure ratio pair. *legstring* is a cell of strings that denote the impeller speed of each speed run as it will appear in the legend. *XI* is a two-column matrix of data points where the first column is mass flow rate in  $lb_m/min$  and the second column is pressure ratio. This matrix is a list of the points on the plot for which interpolation is requested. If *XI* is created from a meshgrid of two vectors, those vectors need to be included as *wi* and *pri* for the mass flow rate and pressure ratio locations at which interpolation is requested. All inputs must be defined in the calling sequence, even if the function for which they operate is turned off.

Since running the program takes only a few seconds, the map was generated first without labels because the end result looked better if the labels are added in Vellum, and then second with the labels so there was no confusion about what efficiency each contour represents. After all of the variables are set appropriately for the test, running *CPM\_Call.m* will result in a MATLAB figure with all of the curves for the performance map, shown in Fig. 7, as well as the efficiency of any requested interpolation points.

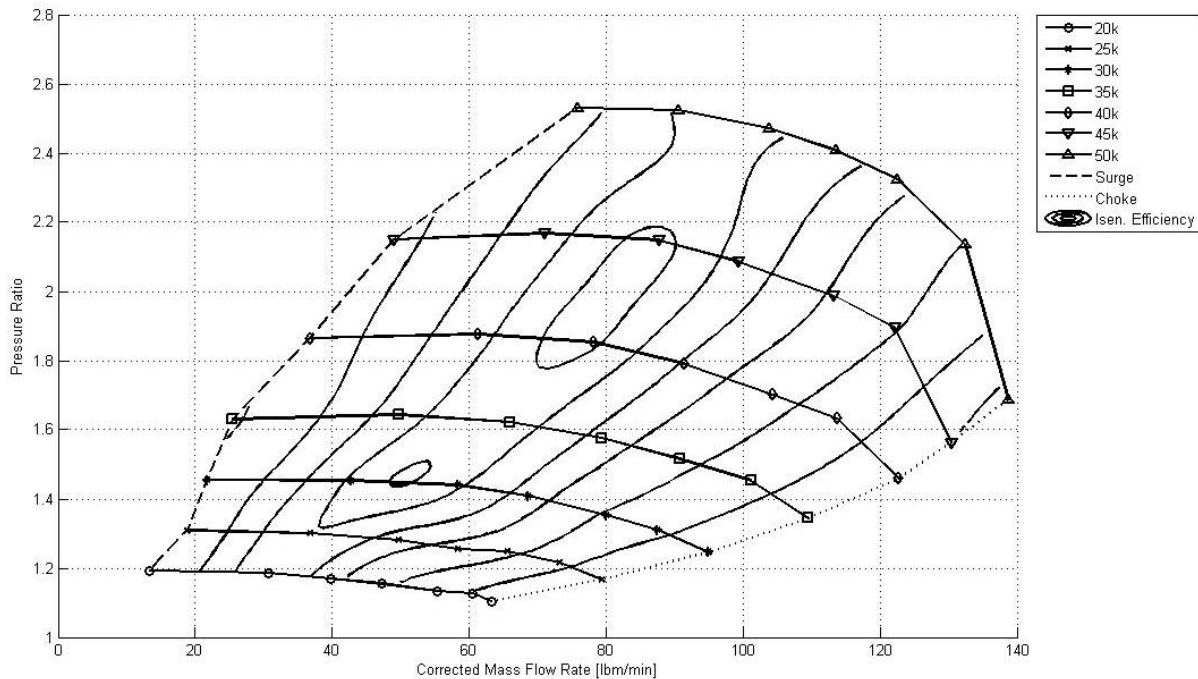


Figure 7: This unaltered MATLAB output does not include efficiency contour labels.<sup>2</sup>

#### D. Modifying the Image

After the MATLAB figure was saved as an image file such as a bitmap, it was opened into Microsoft Paint. In Paint, the figure was cropped along the axes of the chart so that only the curves and gridlines were included. The x-axis location of the crop was to be one gridline beyond the furthest curve. Also, the upper left corner of the grid was erased so it would not interfere with a textbox which resides in that location. Finally, very small peak efficiency islands, known as “phantom efficiency islands”, were removed at the discretion of the operator. This cropped image was then saved as a new bitmap file, which is shown in Fig. 8.

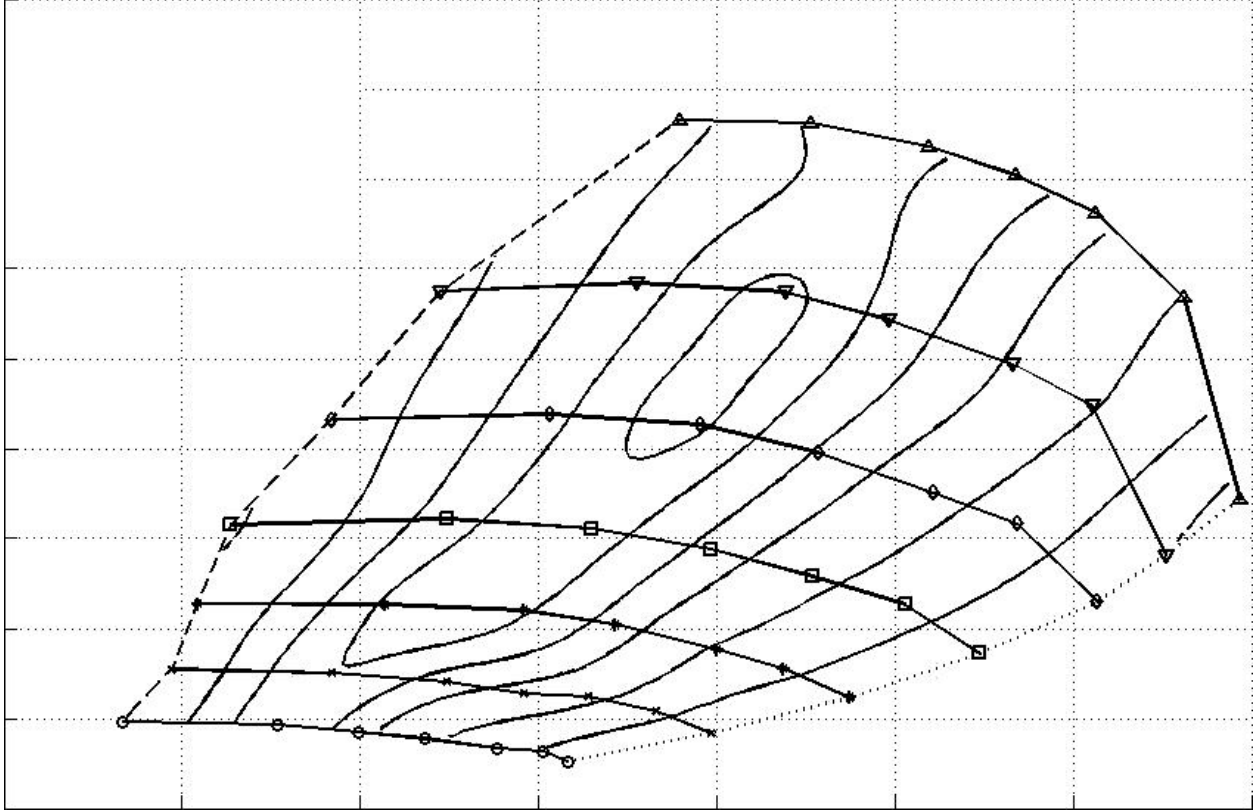


Figure 8: The MATLAB output was modified to be imported into Vellum.<sup>2</sup>

#### E. Fitting the Image Onto a Vellum Template

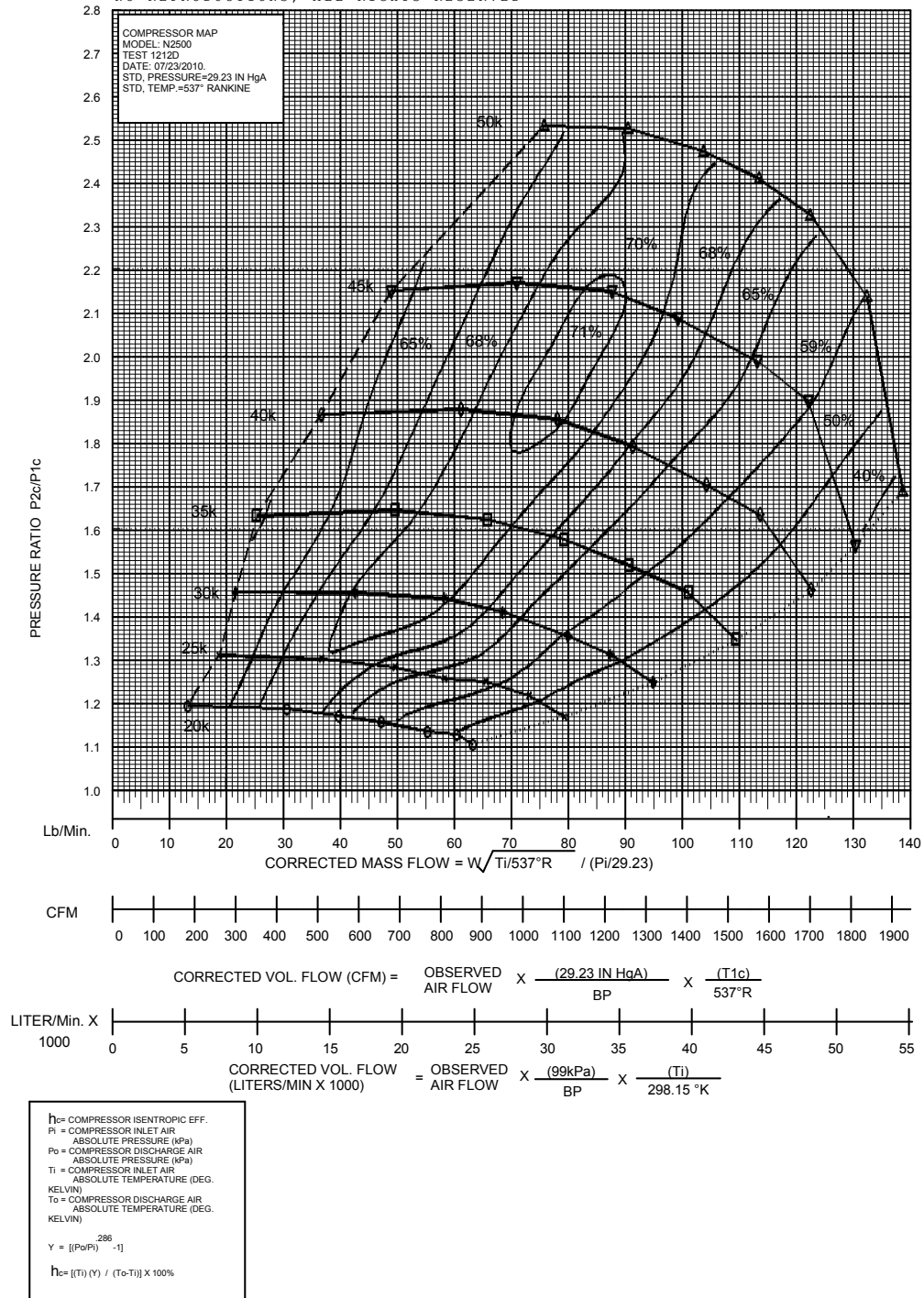
A set of templates for the formal supercharger performance map were created to include title boxes, gridlines, multiple x-axis units, and notes. The templates differed from one another in the range of mass flow rate and pressure ratio. The MATLAB output figure was used to determine which template to use. The template that was used for this particular supercharger is shown in Appendix B.

After opening the appropriate template in Vellum, the modified bitmap image was imported onto the template. Then, the corners of the imported image were dragged until the gridlines on the Vellum template matched the MATLAB generated gridlines to a reasonable accuracy. Next, textboxes were added where needed to label the speed lines and efficiency contours, and the titles were adjusted to fit the current test. The final product was then saved as a PDF in both color and grayscale formats for distribution. The grayscale map is shown in Fig. 9, and the color version is shown in Appendix C.

### V. Validation

Of the three stages that make up this project, the mapping and interpolating stages could be validated, but the stage in which the performance parameters were calculated could only be subjected to rigorous scientific scrutiny. Detailed investigation into each of the equations involved in this stage suggests that they

PAXTON AUTOMOTIVE CORP.  
COMPRESSOR PERFORMANCE MAP  
NO REPRODUCTIONS, ALL RIGHTS RESERVED



NOTES: PERFORMANCE OBTAINED AND CORRECTED IN ACCORDANCE WITH SAE J1723

Figure 9: The mapping stage culminated in this final product.<sup>2</sup>

are correct.

## A. Validation: Plotting Sequence

A good compressor performance map represents the data from which it was plotted clearly and accurately. A validation of this capability must show that the 49 data points that were collected in the test cell are accurately depicted and that the curves between them represent reality. To accomplish this, the map was manually checked against the calculated parameters and marked either green if the two match or red if they do not. Then, anywhere that the curve does not behave as expected between the points is marked by an orange, correct curve. This was done for both the map produced by the CPM program and the one made from the old “spline” method, and both are shown in Fig. 10.

The former method for plotting the compressor performance map is shown here to have been very inaccurate. Moreover, the level of accuracy achieved through this method depended entirely on the diligence of the operator. By plotting the compressor map with a numerical algorithm, human error is eliminated from the system. Unfortunately, the error that remains is more difficult to fix. The efficiency curves near the edge of the domain are skewed because *gridfit.m* extrapolates to approach a level surface rather than fitting the current trends. Mixing this extrapolation and the fact that smoothness must be maintained produces a result where the curves just inside the edge of the domain are not what they should be.

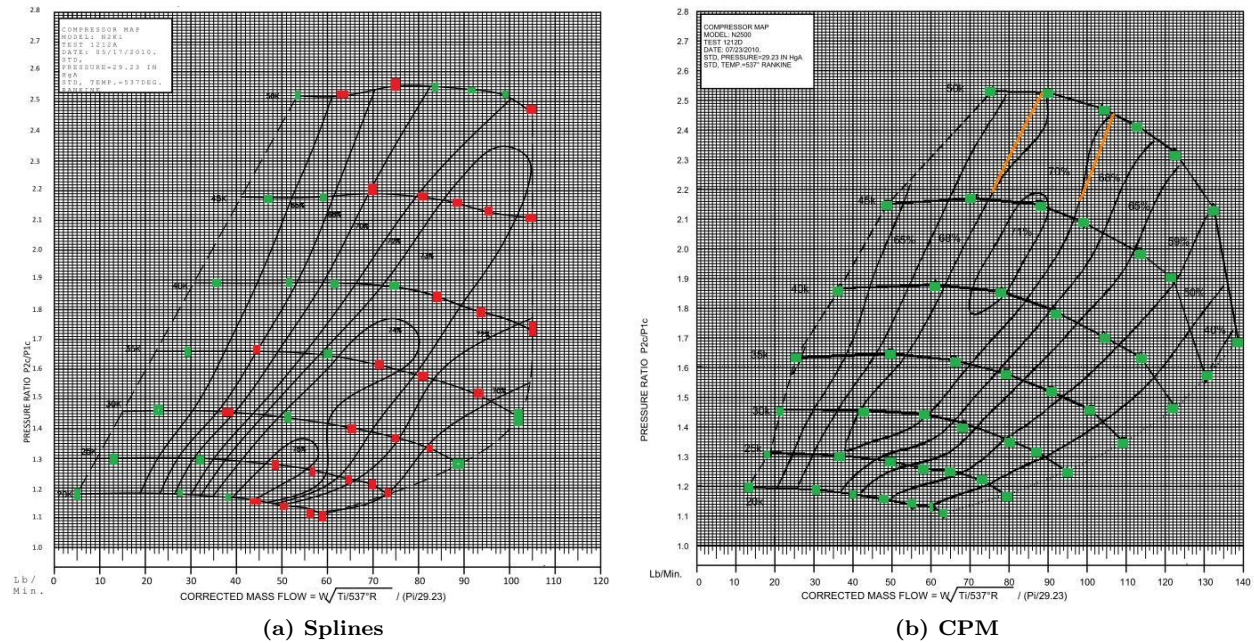


Figure 10: The CPM method is much more accurate than the splines method.<sup>2</sup>

## B. Validation: Interpolation

The interpolation method is valid if the interpolated data accurately matches the plotted data. In order to check this, a dense grid of data points that covers the entire domain of the map was used. After using the interpolation calculation to find the efficiency at all of these points, they were plotted on top of the previously generated map. If the contours from the interpolation match the contours of the original map, then the interpolation method accurately reads the performance of the supercharger. The validation plot described is shown in Fig. 11, which shows that the interpolated data does in fact match the original map even at intervals in between the formerly plotted contours.

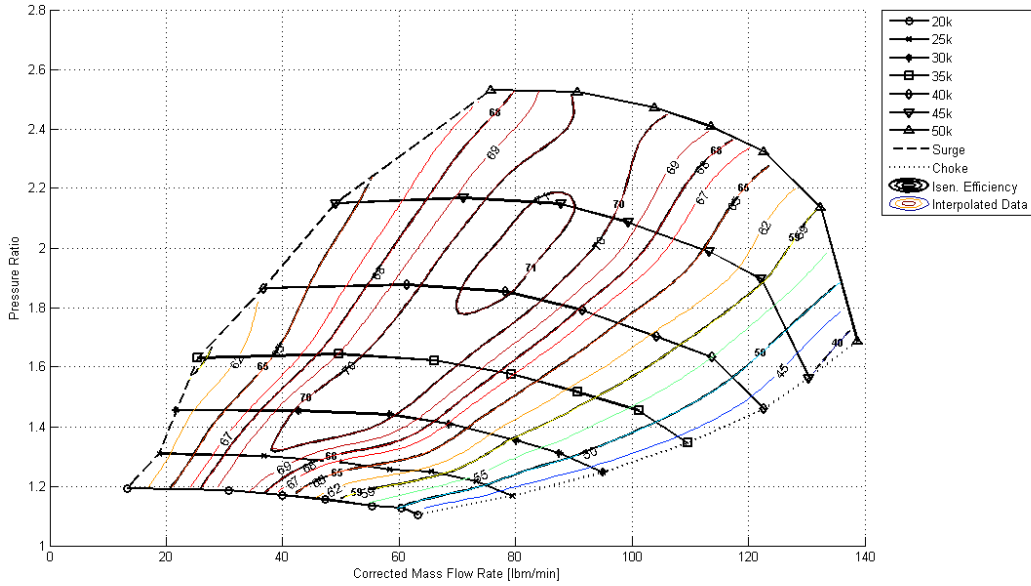


Figure 11: The interpolated data “reads” the map almost exactly.<sup>2</sup>

## VI. Conclusion

The Compressor Performance Map program effectively generates and reads compressor performance maps in compliance with SAE J1723. Furthermore, it performs significantly quicker than the previously accepted method for plotting performance maps. With the program capable of producing a plot in under thirty seconds, several renditions can be compared in a way that the manual methods could not allow. It is estimated that a company can reduce the amount of time to create a finished compressor performance map by seven hours if they switch from plotting with splines on a CAD program to this numerical method. In addition, the numerically generated map fits the data almost perfectly, whereas the manually generated map is subject to human folly. As a separate operational function, the program is readily capable of determining the efficiency of a compressor for any given mass flow rate and pressure ratio within the tested domain. This capability can be applied to engine system simulations in both automobiles and jet engines.

Both the plotting and the reading portions of this project could be sources of future work. For plotting, *gridfit.m* could be modified to smooth perfectly within the boundaries of the domain, regardless of whether the hull was convex or concave. The goal of this modification would be to resolve the issue with the efficiency curves near the edge of the domain. For reading the map, the program could be optimized to be a part of an engine simulation. If the code is being called a large number of times for individual data points, the map “writing” portion which does the smoothing could be translated to an input in order to save a significant amount of time per call.

## Acknowledgements

This project was far from a one man show. I would like to acknowledge my family and friends, whose support has encouraged me to pursue excellence in all of my work. Furthermore, I am grateful for Jim Middlebrook, Rob Anderson, Jeff James, and everyone else at Vortech Engineering for assigning me this task during my internship, mentoring me through the entire process, and allowing me to develop upon and publish it for my senior project. In addition, I sincerely appreciate the thoughtful and unrestricted advice I received from John D’Errico, which was paramount to my success in this project. Finally, many thanks go out to Dr. David Marshall for being my senior project advisor and seeing this project through to completion.

## References

- <sup>1</sup>Society of Automotive Engineers, Inc. "SAE J1723: Supercharger Testing Standard", Surface Vehicle Standard, 1995.
- <sup>2</sup>Images courtesy of Vortech Engineering, Inc. and Paxton Automotive Corp., Oxnard, CA.
- <sup>3</sup>National Oceanic and Atmospheric Administration. "Current Weather Conditions: NAWCWPNS Point Mugu, CA, United States", National Weather Service, <http://weather.noaa.gov/weather/current/KNTD.html>
- <sup>4</sup>Brice, Tim, Hall, Todd, "Station Pressure", National Weather Service, [http://www.srh.noaa.gov/epz/?n=wxcalc\\_stationpressure](http://www.srh.noaa.gov/epz/?n=wxcalc_stationpressure)
- <sup>5</sup>Alfano, D.. "Flow Measurement with Aid Flow Nozzles", Garrett, Aireseach Industrial Division, April 1975.
- <sup>6</sup>DErrico, John. "Surface Fitting using gridfit" ), MATLAB Central File Exchange, 2005, <http://www.mathworks.com/matlabcentral/fileexchange/8998>, Retrieved July 9, 2010.
- <sup>7</sup>D'Errico, John. Personal Communications. July 9-14, 2010.
- <sup>8</sup>Recktenwald, Gerald. "Loading Data into MATLAB for Plotting", Mechanical Engineering Department, Portland State University, 24 August 1995, <http://web.cecs.pdx.edu/~gerry/MATLAB/plotting/loadingPlotData.html>

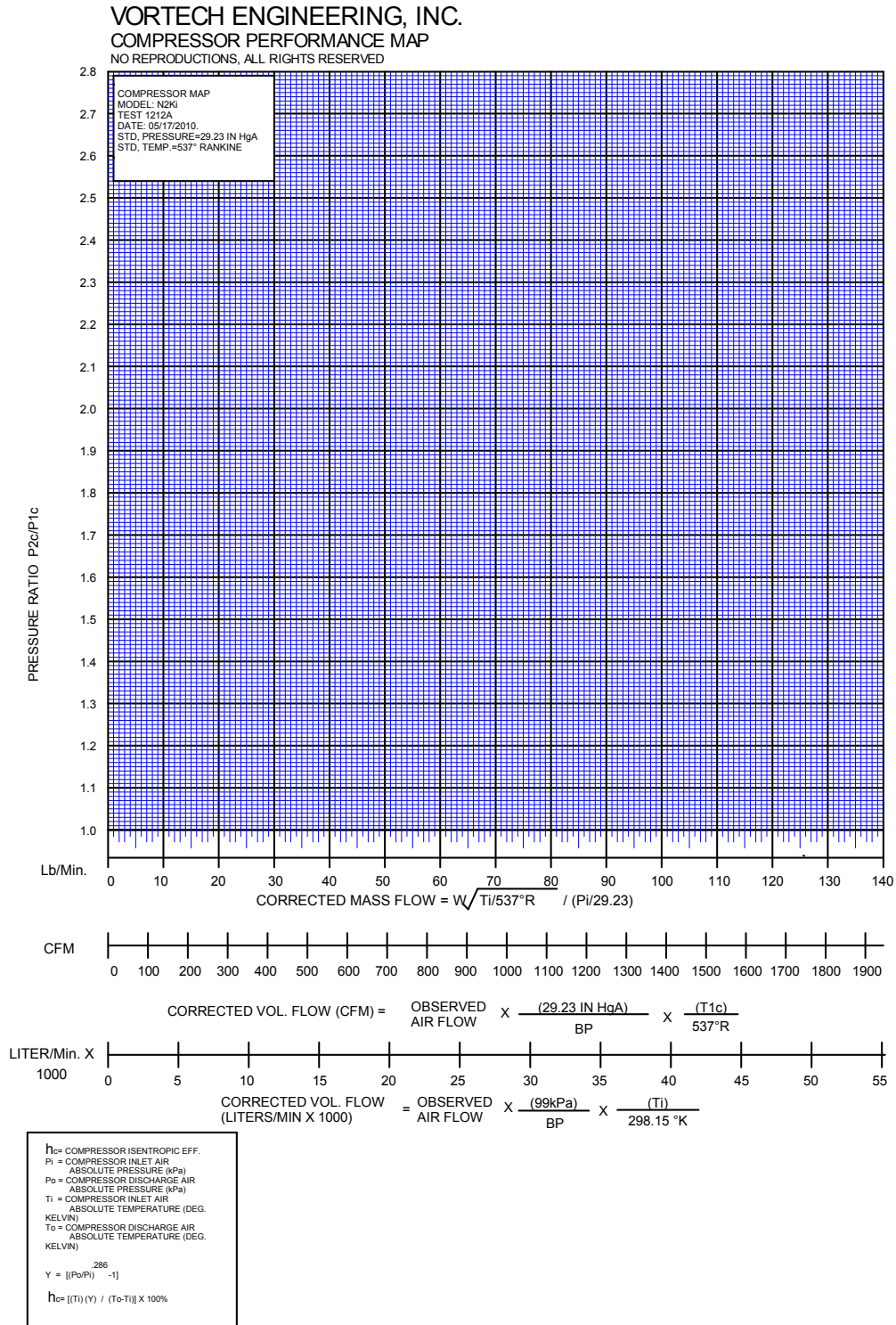
## Appendix

### A. Input File

1212D\_Input

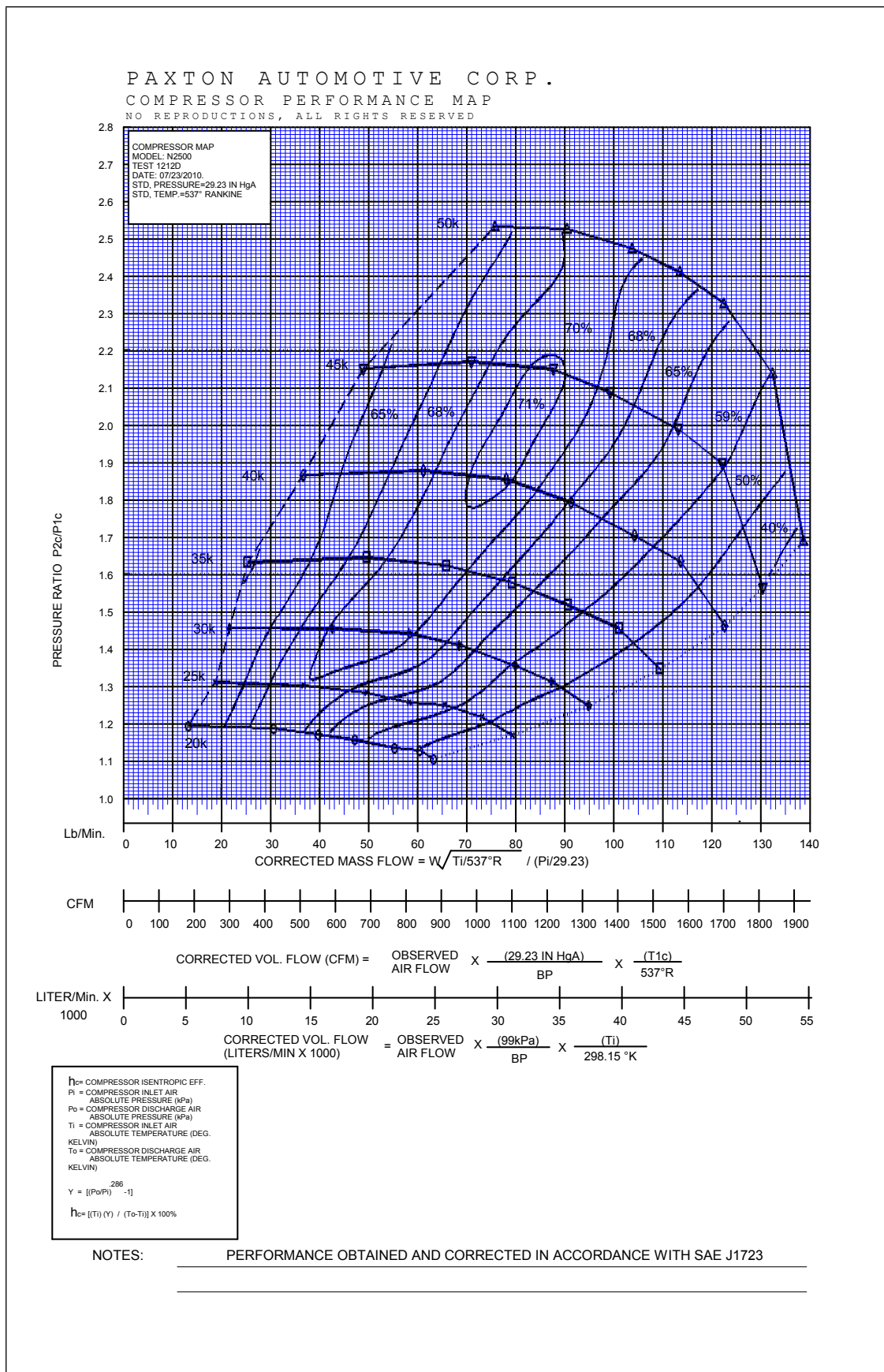
r_s	T_BM	T_11	T_12	T_13	T_21	T_22	T_23	P_BM	P1	P2	P_baro	D1	D2	A_BM
11	60	60	60	105	105	105	0.03	0.05	17.47	29.91	6.987	2.600	38.760	
12	61	61	61	96	96	96	0.16	0.20	17.15	29.91	6.987	2.600	38.760	
13	61	61	61	60	92	93	92	0.27	0.40	16.71	29.91	6.987	2.600	38.760
14	61	61	60	60	91	90	90	0.38	0.50	16.30	29.91	6.987	2.600	38.760
15	61	61	61	61	88	86	88	0.52	0.70	15.70	29.91	6.987	2.600	38.760
16	61	61	61	61	85	86	87	0.62	0.70	15.41	29.91	6.987	2.600	38.760
17	61	61	61	61	83	85	85	0.68	0.80	14.93	29.91	6.987	2.600	38.760
21	61	61	61	61	129	129	129	0.06	0.10	19.13	29.91	6.987	2.600	38.760
22	61	61	61	61	116	116	116	0.23	0.30	18.74	29.91	6.987	2.600	38.760
23	62	61	61	60	112	112	112	0.42	0.50	18.14	29.91	6.987	2.600	38.760
24	62	62	61	61	108	108	107	0.58	0.70	17.46	29.91	6.987	2.600	38.760
25	62	62	62	61	107	106	106	0.73	0.80	17.07	29.91	6.987	2.600	38.760
26	62	62	62	61	104	99	104	0.91	1.00	16.21	29.91	6.987	2.600	38.760
27	62	62	62	61	98	96	99	1.07	1.30	15.08	29.91	6.987	2.600	38.760
31	62	62	62	62	159	159	159	0.08	0.10	21.25	29.92	6.987	2.600	38.760
32	62	62	62	62	142	142	142	0.31	0.40	20.88	29.92	6.987	2.600	38.760
33	63	62	62	62	137	136	137	0.58	0.70	20.28	29.92	6.987	2.600	38.760
34	63	63	63	63	133	133	133	0.80	0.90	19.45	29.92	6.987	2.600	38.760
35	64	63	63	63	129	127	127	1.09	1.30	18.05	29.92	6.987	2.600	38.760
36	64	64	64	63	127	114	125	1.30	1.60	16.95	29.92	6.987	2.600	38.760
37	64	64	64	64	119	104	119	1.53	1.80	15.36	29.92	6.987	2.600	38.760
41	64	64	64	64	198	198	198	0.11	0.20	23.81	29.93	6.987	2.600	38.760
42	65	64	64	65	176	176	176	0.42	0.50	23.58	29.93	6.987	2.600	38.760
43	65	65	65	65	167	167	168	0.74	0.90	22.79	29.93	6.987	2.600	38.760
44	65	65	65	65	161	161	161	1.07	1.20	21.57	29.93	6.987	2.600	38.760
45	66	65	65	65	156	156	154	1.40	1.70	20.06	29.93	6.987	2.600	38.760
46	66	65	66	65	153	142	150	1.74	2.00	18.38	29.93	6.987	2.600	38.760
47	67	66	66	66	142	119	140	2.04	2.40	15.81	29.93	6.987	2.600	38.760
51	69	68	69	68	232	231	232	0.23	0.30	27.11	29.94	6.987	2.600	38.760
52	69	68	68	68	212	212	212	0.64	0.70	26.76	29.94	6.987	2.600	38.760
53	69	69	69	69	203	203	203	1.04	1.20	25.88	29.94	6.987	2.600	38.760
54	69	69	69	69	196	196	196	1.42	1.70	24.35	29.94	6.987	2.600	38.760
55	70	69	69	69	189	189	188	1.85	2.20	22.26	29.94	6.987	2.600	38.760
56	71	70	70	70	187	184	182	2.20	2.60	20.46	29.94	6.987	2.600	38.760
57	71	70	70	70	169	153	168	2.56	3.00	16.52	29.94	6.987	2.600	38.760
61	70	70	69	69	269	269	269	0.41	0.50	31.12	29.94	6.987	2.600	38.760
62	71	70	70	70	253	253	253	0.86	1.00	30.85	29.94	6.987	2.600	38.760
63	71	71	71	70	243	243	243	1.31	1.50	30.00	29.94	6.987	2.600	38.760
64	72	71	71	71	236	236	237	1.68	2.00	28.56	29.94	6.987	2.600	38.760
65	73	72	73	72	230	231	231	2.18	2.60	26.30	29.94	6.987	2.600	38.760
66	74	73	73	72	224	225	223	2.54	2.90	24.22	29.94	6.987	2.600	38.760
67	75	74	74	73	202	190	201	2.89	3.50	17.44	29.94	6.987	2.600	38.760
71	70	69	69	69	303	303	303	0.98	1.10	36.19	29.98	6.987	2.600	38.760
72	71	70	70	69	290	290	290	1.40	1.70	35.62	29.98	6.987	2.600	38.760
73	72	70	70	70	281	281	282	1.84	2.20	34.32	29.98	6.987	2.600	38.760
74	73	72	72	71	276	276	277	2.20	2.60	32.90	29.98	6.987	2.600	38.760
75	73	72	72	71	271	271	272	2.56	3.00	31.12	29.98	6.987	2.600	38.760
76	74	72	72	71	261	263	262	2.99	3.50	27.39	29.98	6.987	2.600	38.760
77	75	73	73	72	244	232	243	3.28	3.85	18.56	29.98	6.987	2.600	38.760

### B. Vellum Template



NOTES: PERFORMANCE OBTAINED AND CORRECTED IN ACCORDANCE WITH SAE J1723

## C. Final Draft in Color



## D. MATLAB Code

```
% Compressor Performance Map Experimental Data Main Function-Calling Code
% Rev: B
% 12/26/2010
% Written By Jeff Freeman
% jeff.freeman01@gmail.com
%
% This program works in association with CPM_read_data.m, CPM_compute.m,
% and CPM.m to process compressor dyno test experimental data.
%
% In this code, the user can choose to either make a compressor map,
% calculate efficiency at one or manypoints, or both.
% To only make a map, let:      plot_on = 1 and interp_on = 1.
% To only interpolate, let:    plot_on = 0 and interp_on = 0.
% To make both, let:          plot_on = 1 and interp_on = 1.
%
% fname is a string which defines the .txt file that contains the raw
%     experimental data.
%
% smooth is a number sets the balance between smoothness and accuracy.
%     Larger numbers are smoother and smaller numbers are more accurate.
%
% legstring is a cell of strings that define what speed each speed line
%     represents.
%
% clabel_on defines whether or not the plot will have contour labels.
%
% XI is a two-column matrix of points to "read", where each row is a new
%     point. The first column is mass flow rate, and the second column is
%     pressure ratio.

clear
clc
close all

%% INSERT EXPERIMENTAL DATA

fname = '1212D_Input.txt';
r = CPM_read_data(fname);

%% Define Operating Parameters

smooth = 5; % See gridfit.m for details

% PLOTTING OPTIONS
plot_on = 1; % Do you want a plot (1) or not (0)?

legstring = {'20k', '25k', '30k', '35k', '40k', '45k', '50k'};
clabel_on = 1; % Do you want efficiency contour labels (1) or not (0)?

% INTERPOLATION OPTIONS
interp_on = 1; % Do you want to read the plot (1) or not (0)?

%XI = Data points (W,PR) at which to interpolate
```

```

wi = linspace(0,140,100);
pri = linspace(1,2.8,100);

[WI, PRI] = meshgrid(wi,pri);

XI(:,1) = WI(:)';
XI(:,2) = PRI(:)';

%% Calculate Performance Parameters from Experimental Data

for i = 1:length(r)
    for j = 1:length(r(i).s)
        [W(i,j), PR(i,j), eff(i,j)] = ...
            CPM_compute(r(i).s(j));
    end
end

%% Call the interpolating program and plot results if desired

[ZI] = CPM(W, PR, eff, smooth, ...
    plot_on, legstring, clabel_on, ...
    interp_on, XI, wi, pri);



---



function r = CPM_read_data(fname)
% read_data.m is used to read the input data files for the CPM Program.
%
% fname, is a .txt file that matches the following characteristics:
%
%     -The first line is the test number
%
%     -The second line is empty
%
%     -The third column reads: r_s      T_BM      T_11      T_12      T_13
%           T_21      T_22      T_23      P_BM      P1      P2      P_baro      D1      D2      A_BM
%
%     -The 15 columns, labeled above, contain raw experimental data
%       such that each row is a new speed, setting pair.
%
%     -The numbers in r_s are two digit integers, where the first digit
%       denotes the speed index and the second digit denotes the
%       setting index.
%
% This function utilizes Gerald Recktenwald's "readColData.m", which can be
% found at:
% http://web.cecs.pdx.edu/~gerry/MATLAB/plotting/loadingPlotData.html
%
% Written by: Jeff Freeman

[labels,r_s,data] = readColData(fname,15,2);

for i = 1:length(r_s)
    r_s_str(i,:) = int2str(r_s(i));
end

rlist = str2num(r_s_str(:,1));

```

```

slist = str2num(r_s_str(:,2));

for i = 1:length(rlist)
    r(rlist(i)).s(slist(i)).T_BM = data(i,1);
    r(rlist(i)).s(slist(i)).T_11 = data(i,2);
    r(rlist(i)).s(slist(i)).T_12 = data(i,3);
    r(rlist(i)).s(slist(i)).T_13 = data(i,4);
    r(rlist(i)).s(slist(i)).T_21 = data(i,5);
    r(rlist(i)).s(slist(i)).T_22 = data(i,6);
    r(rlist(i)).s(slist(i)).T_23 = data(i,7);
    r(rlist(i)).s(slist(i)).P_BM = data(i,8);
    r(rlist(i)).s(slist(i)).P1 = data(i,9);
    r(rlist(i)).s(slist(i)).P2 = data(i,10);
    r(rlist(i)).s(slist(i)).P_baro = data(i,11);
    r(rlist(i)).s(slist(i)).D1 = data(i,12);
    r(rlist(i)).s(slist(i)).D2 = data(i,13);
    r(rlist(i)).s(slist(i)).A_BM = data(i,14);
end

```

---

```

function [W, PR, eff] = CPM_compute(data)
% Compressor Performance Parameter Calculator
% Rev: A
% 12/23/2010
% Written by: Jeff Freeman
% jeff.freeman01@gmail.com
%
% This program calculates SAE Corrected Mass Flow Rate (W) [lbm/min],
% Pressure Ratio (PR) and Isentropic Efficiency (eff) [%] of a compressor
% from dyno test cell data. The calculations conform with SAE J1723. The
% data necessary for calculation includes one speed-setting pair as defined
% by CPM_read_data.m

```

#### %% Decompress data

```

T_BM = data.T_BM;% = Bell Mouth Temperature [F]
T_11 = data.T_11;% = Compressor Inlet Temperature #1 [F]
T_12 = data.T_12;% = Compressor Inlet Temperature #2 [F]
T_13 = data.T_13;% = Compressor Inlet Temperature #3 [F]
T_21 = data.T_21;% = Compressor Discharge Temperature #1 [F]
T_22 = data.T_22;% = Compressor Discharge Temperature #2 [F]
T_23 = data.T_23;% = Compressor Discharge Temperature #3 [F]
P_BM = data.P_BM;% = Bell Mouth Pressure [in. H2O (gauge)]
P1 = data.P1;% = Compressor Inlet Pressure [in. H2O (gauge)]
P2 = data.P2;% = Compressor Discharge Pressure [psia]
P_baro = data.P_baro;% = Barometric Pressure [in. Hg abs]
D1 = data.D1;% = Inner Diameter of Inlet Section [in]
D2 = data.D2;% = Inner Diameter of Discharge Section [in]
A_BM = data.A_BM;% = Calibrated Flow Nozzle (Bell Mouth) Area [in^2]

```

#### %% Calculate W (W\_sae\_corr)

```

T_1 = (T_11 + T_12 + T_13)/3 + 459.67; % Average Inlet Temp. [R]

P1_abs = P_baro - P1/13.608; % P1 [in. Hg abs] *note: recorded P1 is gauge

```

```

% Observed Mass Flow [lbm/min]
W_obs = 2.06 * A_BM * sqrt(P_baro*17.35*P_BM/(T_BM+459.67));

% Total Inlet Pressure [in. Hg abs]
Po1 = P1_abs * ((W_obs^2 * T_1) / (P1_abs^2 * D1^4 * 646) + 1);

W = W_obs * sqrt(T_1/537) * 29.236/Po1; % W_sae_corr
%W = W_obs * (T_BM+459.67)/536.67 * 29.238/P_baro; % W_sae_corr

%% Calculate PR (Pressure Ratio)

T_2 = (T_21 + T_22 + T_23)/3 + 459.67; % Average Discharge Temp. [R]

P2_Hg = P2 * 2.036; % Convert from psia to in. Hg abs

% Total Discharge Pressure [in. Hg abs]
Po2 = P2_Hg * ((W_obs^2 * T_2) / (P2_Hg^2 * D2^4 * 646) + 1);

PR = Po2/Po1;

%% Calculate eff (Isentropic Efficiency)

% Observed volumetric flow rate inlet [ft^3/min]
Q1 = W_obs * 0.75354 * (T_BM+459.67) / P_baro;

% Total Inlet Temp. [R]
To1 = T_1 + (Q1/(pi * (D1/24)^2))^2 / 43258068;

% Discharge Density [lbm/ft^3]
rho2 = P2*144/(53.3*T_2);

% Discharge speed of sound [ft/s]
c2 = sqrt(1.4*1716.59*T_2);

% Discharge volumetric flow rate [ft^3/min]
Q2 = W_obs/rho2;

% Discharge velocity [ft/s]
v2 = Q2/(60 * pi * (D2/24)^2);

% Discharge Mach number
M2 = v2/c2;

% Total Discharge Temp. [R]
To2 = T_2 * (1 + 0.2*M2^2);
%To2 = T_2 + (Q2/(pi * (D2/24)^2))^2 / 43084924;

dT = To2 - To1; % Total Temp Differential [R]

eff = (To1 * PR^0.286 - To1) / dT * 100; % Isentropic Efficiency

end

```

```

function [ZI] = CPM(W, PR, eff, smooth, ...% Base Inputs
    plot_on, legstring, clabel_on, ...    % Plotting Inputs
    interp_on, XI, wi, pri)               % Interpolating (reading) Inputs
% Compressor Map Plotting
% Rev: B
% 12/26/2010
% Written By Jeff Freeman
% jeff.freeman01@gmail.com
%
% This code plots a compressor map according to SAE J1723 and reads the
% efficiency for any mass flow rate - pressure ratio pair.
%
% Inputs are (For more information, see CPM_Call.m):
%   Base Inputs
%       W          == Corrected Mass Flow Rate
%       PR          == Pressure Ratio
%       eff         == Isentropic Efficiency
%       smooth      == Integer defining how much the program smooths data
%
%   Plotting Inputs
%       plot_on     == Defines whether to plot (1) or not (0)
%       legstring   == Cell of strings defining the legend entries
%       clabel_on   == Defines whether to plot contour labels (1) or
%                       not (0)
%
%   Interpolating (reading) Inputs
%       XI          == [n,2] array of [W,PR] points at which to
%                       interpolate data. If XI is built from a
%                       meshgrid of linspace (or similar), include the
%                       original linspace, wi and pri.
%       wi          == Array of Mass Flow Rates used to build XI(:,1)
%       pri         == Array of Mass Flow Rates used to build XI(:,2)
%
% The inputs W, PR, and eff need to be m x n matrices where m is the number
% of speed lines and n is the number of data points taken at each speed
% line. All speed lines must have the same number of data points associated
% with it. This version is compatible with up to 12 speed lines.
%
%
% To run this program, you need to have available the following codes
% written by John D'Errico:
%     gridfit.m
%     poly2tri.m
%     insc.m
%     simplicialcomplex.m
%     buildbccdata.m
%
% See also: CPM_Call, CPM_read_data, CPM_compute, gridfit, poly2tri, insc

%% Parse inputs for bookkeeping

% Check to make sure we are doing something
if ~interp_on && ~plot_on
    error 'Need to be either plotting, interpolating, or both.'
end

```

```

% Check for existence of necessary variables (W, PR, eff)
if exist('W','var') == 0 || ...
    exist('PR','var') == 0 || exist('eff','var') == 0
    error 'Not enough inputs are defined.'
end

% Define defaults of legstring and smooth if not provided
if exist('legstring','var') == 0
    legstring = {'20k','25k','30k','35k','40k','45k','50k'};
end

if exist('smooth','var') == 0
    smooth = 5;
end

% Determine number of speed lines
num = size(W); num = num(1);

% Check to see if default legstring is valid for size of input data
if length(legstring) ~= num
    error 'legstring size is not compatible with number of speed lines'
end

%% Find Surge and Choke Lines
% Surge Line
for i = 1:num
    W_surge(i) = W(i,1);
    PR_surge(i) = PR(i,1);
end

% Choke Line
for i = 1:num
    W_choke(i) = W(i,end);
    PR_choke(i) = PR(i,end);
end

%% Smooth the data for a pretty plot

% Reshape the data matrices into 1D vectors
W_vec = W(:);
PR_vec = PR(:);
eff_vec = eff(:);

% smoothing using gridfit. A bigger number will have a finer mesh.
Wnodes = 100;
PRnodes = 100;

[eff_grid, W_grid, PR_grid] = ...
    gridfit(W_vec,PR_vec,eff_vec,Wnodes,PRnodes...
        , 'smoothness', smooth);
% The last input for this function determines the ratio how "smooth" the

```

```

% map looks compared to how accurately it fits te data. Small numbers are
% very accurate while large numbers are very smooth.

% trace the external polygon (ccw direction)
x_edge = [W(1,:), W_choke, fliplr(W(end,:)), fliplr(W_surge)];
y_edge = [PR(1,:), PR_choke, fliplr(PR(end,:)), fliplr(PR_surge)];

% eliminate duplicate points at the corners
i = 1;
while i < length(x_edge)
    i = i+1;
    if x_edge(i) == x_edge(i-1)
        x_edge(i) = [];
        y_edge(i) = [];
    end
end

% triangulate using poly2tri to build a simplicial complex for use as a
% boundary check
sc = poly2tri(x_edge,y_edge);

% If I am making multiple calls to insc, I want to pre-allocate bccdata.
sc.bccdata = buildbccdata(sc);

%% Perform Bilinear Interpolation
if interp_on
    % User has input interpolation points and wants them to be output and
    % plotted.
    warnmark = 1;
    for i = 1:size(XI,1)

        % filter out parts of the grid that lie outside of the domain
        if insc(sc,XI(i,:),0.001) ~= 1
            ZI(i,1) = NaN;
            if warnmark
                warning('One or more desired interpolation points are outside of domain')
                warnmark = 0;
            end
            continue
        end

        % Run interp2
        ZI(i,1) = interp2(W_grid,PR_grid,eff_grid,XI(i,1),XI(i,2));

    end
else ZI = NaN;
end
% User just wants the basic compressor map.

%% Plot Compressor Map

```

```

if plot_on

    % filter out parts of the grid that lie outside of the plotting domain
    for i = 1:Wnodes
        for j = 1:PRnodes
            if insc(sc,[W_grid(i,j),PR_grid(i,j)],.001) ~= 1
                W_grid(i,j) = NaN;
                PR_grid(i,j) = NaN;
                eff_grid(i,j) = NaN;
            end
        end
    end

    %Define contour locations
    E_max = floor(max(max(eff)));
    E_min = floor(min(min(eff)));

    % If E_min is very low, we want it to count in increments of 10 until
    % we get close to E_max. Need to find the starting point.
    E_min1 = E_min/10;      % divide by 10 so we're dealing with x.x
    E_min2 = ceil(E_min1); % round up to nearest integer
    E_min3 = E_min2*10;     % *10 to get back up to percentage

    V1 = [E_max, (E_max-1)]; % count by 1
    V2 = E_max-3;           % count by 2
    V3 = E_max-6;           % count by 3
    V4 = E_max-12;          % count by 6
    V5 = fliplr((E_min3):10:(E_max-20)); % count by 10

    if V4-max(V5) > 11
        V4(2) = V4(1) - 6;
    end

    V = [V1, V2, V3, V4, V5];
    V = fliplr(V);

    % Initiate Plot
    figure
    hold on
    xlabel('Corrected Mass Flow Rate [lbm/min]'), ylabel('Pressure Ratio')
    axis([0, 140, 1.0, 2.8])
    grid on

    % Define cycle of line properties
    S = {'k-o','k-x','k-*','k-s','k-d','k-v','k-^','k-p','k-h'...
        , 'k-<','k->','k-+'};

    % Plot W vs. PR speed lines
    for i = 1:num
        plot(W(i,:),PR(i,:),S{i},'linewidth',2.0);
    end

    % Plot Choke and Surge Lines
    h = plot(W_surge,PR_surge,'k--',W_choke,PR_choke,'k:');

```

```

set(h,'LineWidth',2.0)

% Plot and Label Efficiency Contours
[c,hc] = contour(W_grid,PR_grid,eff_grid,V);
set(hc,'linewidth',2.0,'linecolor','k')
colormap('white')
if clabel_on
    clabel(c,hc,'background','white',...
        'fontweight','bold',...
        'fontsize',8,...
        'rotation',0,...
        'labelspacing',300)
end

if interp_on && exist('wi','var') && exist('pri','var')
    % Plot and Label Interpolated Efficiency Contours
    j = 1;
    for i = 1 : length(V)-1
        vi(j) = V(i);
        vi(j+1) = round((V(i+1)+V(i))/2);
        j = j+2;
    end

    zi = reshape(ZI,length(wi),length(pri));
    [csi,hi] = contour(wi,pri,zi,vi);
    colormap('jet')
    if clabel_on
        clabel(csi,hi,...
            'labelspacing', 300)
    end
end

% Compile Legend from legstring input
if interp_on
    legstring1 = {legstring{:},'Surge','Choke','Isen. Efficiency'...
        ,'Interpolated Data'};
else
    legstring1 = {legstring{:},'Surge','Choke','Isen. Efficiency'};
end

% Insert Legend
legend(legstring1...
    ,'location','BestOutside');

hold off

end % This "end" ends the plotting sequence

```

---

```

function sc = poly2tri(x,y)
% poly2tri: converts a polygon in (x,y) into a triangulation (a
% simplicialcomplex)
% usage: sc = poly2tri(x,y)

```

```

%
% Note: uses ear clipping to triangulate the polygon. The
% polygon does not need to be convex, but it must be a
% simply connected polygon that does not cross itself.
%
% arguments: (input)
%   x,y - vectors of points that define the polygon.
%         x and y must be the same length vectors (either
%         row or column vectors are allowed.)
%
%
% arguments: (output)
%   sc - simplicial complex structure - as defined by the
%        simplicialcomplex function.
%
%
% Example usage:
% An arbitrary (non-convex) polygon
%
%   x = [0 1 1 .6 .2 .5 0 0];
%   y = [0 0 1 .1 .5 .75 1 0];
%   sc = poly2tri(x,y);
%
% Example usage:
% Triangulation from points on the perimeter of a circle
%
%   theta = linspace(0,2*pi,20);
%   x = cos(theta);
%   y = sin(theta);
%   sc = poly2tri(x,y);
%
%
% See also: simplicialcomplex, delaunays, alphashape
%
%
% Author: John D'Errico
% E-mail: woodchips@rochester.rr.com
% Release: 1.0
% Release date: 10/13/08

% make both x and y into column vectors
x=x(:);
y=y(:);

n = length(x);
if n~=length(y)
    error 'x and y must be vectors of the same length'
end

% combine x and y into a set of domain vertices
xy = [x,y];

% Was the last point wrapped around? If so, we
% can drop it. if not, then connect the two ends

```

```

% of the polygon.
if all(xy(1,:) == xy(end,:))
    xy(end,:) = [];
    n = n-1;
end

% we must have at least three points in the polygon
if n<=2
    error 'Insufficient points to triangulate the polygon'
end

% list of edges of the polygon. There will be n edges.
edges = (1:n)';
edges = [edges,edges+1];
% make that last edge wrap around
edges(end,2) = 1;

% Form a simplicial complex structure for eventual
% return. It will be a triangulation by the time we
% are done.
sc = simplicialcomplex(xy,edges);

% form triangles from each pair of consecutive edges
eartri = [edges,edges([2:end),1],2)];

% compute the included angle between the pairs
% of edges of each triangle in this list
ang12 = atan2(xy(eartri(:,1),2) - xy(eartri(:,2),2), ...
    xy(eartri(:,1),1) - xy(eartri(:,2),1));
ang23 = atan2(xy(eartri(:,3),2) - xy(eartri(:,2),2), ...
    xy(eartri(:,3),1) - xy(eartri(:,2),1));
ang = ang12 - ang23;

% if an angle is negative, we may either be traversing
% the polygon in a clockwise direction, or this may be a
% convex polygon. We need to figure out what is happening.
m = (ang < 0);
ang(m) = ang(m) + 2*pi;

% was the curve traversed clockwise or counter-clockwise?
% If the sum of the angles is now (n+2)*pi, then we
% have traversed the polygon in a clockwise sequence.
totalangle = sum(ang);
if abs(totalangle - (n+2)*pi)<(1e4*eps)
    % swap to a counterclockwise order
    ang = 2*pi - ang;
    eartri = eartri(:,[3 2 1]);
    totalangle = sum(ang);
end

% The sum of angles must be (n-2)*pi
% the polygon was traversed in a counterclockwise sequence. If
% the sum was zero, then the polygon must cross itself.
if abs((totalangle) - pi*(n-2)) > (10000*eps)
    % the polygon was not a proper one.

```

```

    error 'Holy polygon, Batman! Its an improper one - does it cross itself?'
end

% It is time to start clipping the polygon into triangles

% preallocate the triangulation array as
% a list of triangles.
tri = nan(n-2,3);
nt = 0;
ne = size(eartri,1);
while ne>3
    % pick that triangle with smallest (positive) included angle
    apos = ang;
    apos((ang<0)|(ang>pi)) = inf;
    [apos,angtags] = sort(apos);

    % will any of the other edges end up crossing the new
    % edge if we clip off the first triangle in this list?
    % If it does, then we need to pick a new triangle that
    % does not make that happen. We can always find one.
    % We do this by testing if any other vertex would have
    % fallen inside the prospective new triangle.
    npos = sum(~isinf(apos));
    failflag = true;
    for itri = 1:npos
        % search through the prospective triangles, starting
        % with the smallest included angle and working up.
        potentialtriangle = eartri(angtags(itri),:);

        allothervertices = setdiff(unique(eartri(:)),potentialtriangle(:));
        P1 = xy(potentialtriangle(1,:),:);
        M = [0 0 1;xy(potentialtriangle(2,:),:)-P1,1; ...
            xy(potentialtriangle(3,:),:)-P1,1]';
        % compute barycentric coordinate for all of them
        nothers = length(allothervertices);
        bcc = M\[xy(allothervertices,:)-repmat(P1,nothers,1),ones(nothers,1)]';
        if all(any(bcc<0,1) | any(bcc>1,1))
            % this triangle is acceptable
            failflag = false;
            break
        end
    end % for itri = 1:ntri
    if failflag
        % if we drop into here, then there was a problem with the polygon
        error('Failure in cleaving this polygon into triangles.')
    end

    % clip off the chosen triangle, add it to tri
    stri = potentialtriangle;
    nt = nt+1;
    tri(nt,:) = stri;

    eartri(angtags(itri),:) = [];
    ang(angtags(itri)) = [];
end

```

```

% modify the pair of triangles that shared
% an edge with the selected triangle.
k3 = find(eartri(:,3) == stri(2));
eartri(k3,3) = stri(3);
k1 = find(eartri(:,1) == stri(2));
eartri(k1,1) = stri(1);

% now update the included angles for triangles k1 and k3
k = [k1,k3];
ang12 = atan2(xy(eartri(k,1),2) - xy(eartri(k,2),2), ...
    xy(eartri(k,1),1) - xy(eartri(k,2),1));
ang23 = atan2(xy(eartri(k,3),2) - xy(eartri(k,2),2), ...
    xy(eartri(k,3),1) - xy(eartri(k,2),1));

angk = ang12 - ang23;
m = (angk < -pi);
angk(m) = angk(m) + 2*pi;
m = (angk > pi);
angk(m) = angk(m) - 2*pi;

ang(k) = angk;

ne = ne - 1;

end % while ne>3
tri(nt+1,:) = eartri(1,:);

% stuff the triangulation into sc.tessellation
sc.tessellation = tri;

```

---

```

function SC = simplicialcomplex(domainpoints,tessellation,range)
% simplicialcomplex: creator tool for simplicial complex struct
% usage: SC = simplicialcomplex(domainpoints,tessellation,range)
%
% arguments (input):
%   domainpoints - (n x p) array of data points, each row is a "point"
%                   in a p-dimensional space
%
%   tessellation - Array containing the tessellation
%                   (as returned by delaunayn or convhull)
%                   Each row defines a simplex in the tessellation, each
%                   element refers to one row of the array in domainpoints.
%
%   A simplex need not be a full volume simplex. It
%   may be a lower order simplex. Thus in p dimensions,
%   a p-simplex with p+1 vertices will be a full volume
%   simplex. More simply, in 3-d, a tetrahedron (4 vertices)
%   is a 3-simplex. A 2-simplex is just a triangle. So the
%   boundary surface of a tessellation is equally admissable
%   in 3-dimensions it will be composed of triangles. This
%   might then be referred to as a 2-manifold in the 3-d
%   domain space.
%

```

```

%
% range - [OPTIONAL] (nxm) array of points in an m-dimensional
%         space. Think of this as the image of the data in
%         the array "points" through some mapping function
%         into another space. range allows us to do interpolation,
%         reverse interpolation (where appropriate), iso-surfaces,
%         iso-slices, and many other useful things.
%
%         If supplied, the range array must have the same number
%         of rows as does the domainpoints array. They may have
%         different numbers of columns, although then reverse
%         interpolation will usually be impossible.
%
% arguments (output):
% SC      - struct containing a simplicial complex
%
%         SC will contain fields named:
%         'domain', 'tessellation', 'range'
%
%         Also allowed are the (OPTIONAL) fields:
%         'description', 'userdata'
%         But these fields can only be set by assigning them, i.e.,
%         SC.description = 'The quick brown fox'
%         SC.userdata = '10-22-06'
%
% See also: delaunays, tessellatehypercube, tessellatelattice, alphashape
%
% Author: John D'Errico
% e-mail: woodchips@rochester.rr.com
% Release: 1.0
% Release date: 10/22/06

% dimension of the space the data lives in?
[n,dim] = size(domainpoints);

if (nargin<2) || isempty(tessellation)
    tessellation = zeros(0,0);
end
[nt,m] = size(tessellation);

% create a structure underlying the object
SC.domain = domainpoints;
SC.tessellation = tessellation;

% create other fields, leave empty
if (nargin<3) || isempty(range)
    SC.range = zeros(n,0);
else
    if (n~=size(range,1))
        error('Domain and range points arrays must have the same number of rows')
    end
    SC.range = range;
end

```

```

function bccdata = buildbccdata(sc)
% buildbccdata: builds the information used by insc to compute barycentric coordinates
% usage: bccdata = buildbccdata(sc)
%
% When multiple successive calls will be made to insc or
% interpvc, use a prior call to buildbccdata to accelerate
% those calls. This is the only reason I've left this
% function accessible to the user. After you call the
% buildbccdata, you must NOT change the vertices or the
% tessellation in the complex, or the bccdata field will
% be incorrect.
%
%   sc.bccdata = buildbccdata(sc);
%
% arguments: (input)
%   sc - a simplicialcomplex struct, to be later used in either
%         insc or interpvc. sc must not contain a manifold. sc
%         must be a full volume complex.
%
% arguments: (output)
%   bccdata - structure containing information used by insc

% unpack things
vert = sc.domain;
[nv,dim] = size(vert);
% a full volume simplex in n-d has one more
% vertex than the number of dimensions. Used
% a lot in this code.
dim1 = dim+1;

% mean subtract the vertices for an accuracy
% gain when the complex is far away from the
% origin.
bccdata.centerpoint = mean(vert,1);

% could use bsxfun here
vert = vert - repmat(bccdata.centerpoint,nv,1);

tess = sc.tessellation;
[nt,cols]=size(tess);
if cols~=dim1
    error('This is a manifold complex')
end

% What version is this? I need to know if we need to
% set some spparms options to get around a solver
% problem.
matlabversion = ver('matlab');
matlabversion = str2double(matlabversion.Version);
if matlabversion >= 7
    % at least release 7
    % store the spparms settings to restore
    % them when we are done
    sppold = spparms;
    spparms('umfpack',0)

```

```

    spparms('bandden',1)
end

% block the problem?
blocksize = min(512,nt);
nL = blocksize;
L = 1:nL;
offset = 0;
bccdata.mat = NaN(dim1,nt*dim1);
failedlist = false(1,nt);
while ~isempty(L)
    % work on a subset of the full problem
    tessL = tess(L,:);

    % stuff into bccdata
    bcc = bccblock(tessL)';
    m = offset + (1:(length(L)*dim1*dim1));
    bccdata.mat(m) = bcc(:);
    % maintain a list of any failures
    if isinf(bcc(1))
        failedlist(L) = true;
    end

    % grab the next block
    offset = dim1*dim1*L(end);
    L = L + blocksize;
    L(L>nt) = [];
end

% did any of the sparse block inverses fail?
% If any did fail, then redo those blocks, one
% piece at a time. A singular matrix will not be
% a problem in itself, since then NO points will
% ever be found by insc that lie inside that
% simplex. The simplex must have been degenerate.
% This is acceptable, since even if a point is
% "inside" a degenerate simplex, it will also be
% on an edge of another simplex too, and thus
% inside that simplex.
k = find(failedlist);
if ~isempty(k)
    for L = k
        tessL = tess(L,:);
        m = dim1*(L-1) + (1:dim1);

        % stuff into bccdata. Some of these will
        % produce garbage, since the previous inv
        % failed on the enclosing block. It is ok
        % though, as argued above.
        bccdata.mat(m,:) = bccblock(tessL);
    end
end

% we generated the transpose of the .mat field,
% so do the swap now.

```

```

bccdata.mat = bccdata.mat';

% restore spparms if we changed it
if matlabversion >= 7
    spparms(sppold)
end

% all done.

% =====
%     nested function - bccblock
% =====
function bccmat = bccblock(subtess)
    nst = size(subtess,1);
    if nst > 1
        % sparse block inverses
        N = dim1*nst;
        m = vert(subtess',:);
        m = [m,ones(N,1)];

        % populate the sparse block diagonal matrix
        % directly. This should be faster than a
        % mat2cell followed by a call to blkdiag.
        rowindex = repmat((1:N)',1,dim1);
        columnindex = repmat(1:N,dim1,1);
        columnindex = reshape(columnindex,[dim1 dim1,nst]);
        columnindex = reshape(permute(columnindex,[2 1 3]),dim1,N)';

        % sparse inverse
        minv = inv(sparse(rowindex,columnindex,m,N,N));

        % recover the blocks as a flat array
        bccmat = minv(sub2ind([N,N],columnindex,rowindex));
        bccmat = reshape(full(bccmat),N,dim1);
    else
        % its a single simplex. don't do it in
        % sparse block diagonal form, as that
        % would be less efficient.
        bccmat = inv([vert(subtess,:')';ones(1,dim1)]);
    end % if nst > 1
end % nested function end
end % mainline end

```

---

```

function sc = poly2tri(x,y)
% poly2tri: converts a polygon in (x,y) into a triangulation (a
% simplicialcomplex)
% usage: sc = poly2tri(x,y)
%
% Note: uses ear clipping to triangulate the polygon. The
% polygon does not need to be convex, but it must be a
% simply connected polygon that does not cross itself.
%
% arguments: (input)

```

```

% x,y - vectors of points that define the polygon.
%       x and y must be the same length vectors (either
%       row or column vectors are allowed.)
%
%
% arguments: (output)
%   sc - simplicial complex structure - as defined by the
%       simplicialcomplex function.
%
%
% Example usage:
% An arbitrary (non-convex) polygon
%
%   x = [0 1 1 .6 .2 .5 0 0];
%   y = [0 0 1 .1 .5 .75 1 0];
%   sc = poly2tri(x,y);
%
% Example usage:
% Triangulation from points on the perimeter of a circle
%
%   theta = linspace(0,2*pi,20);
%   x = cos(theta);
%   y = sin(theta);
%   sc = poly2tri(x,y);
%
%
% See also: simplicialcomplex, delaunays, alphashape
%
%
% Author: John D'Errico
% E-mail: woodchips@rochester.rr.com
% Release: 1.0
% Release date: 10/13/08

% make both x and y into column vectors
x=x(:);
y=y(:);

n = length(x);
if n~=length(y)
    error 'x and y must be vectors of the same length'
end

% combine x and y into a set of domain vertices
xy = [x,y];

% Was the last point wrapped around? If so, we
% can drop it. if not, then connect the two ends
% of the polygon.
if all(xy(1,:) == xy(end,:))
    xy(end,:) = [];
    n = n-1;
end

```

```

% we must have at least three points in the polygon
if n<=2
    error 'Insufficient points to triangulate the polygon'
end

% list of edges of the polygon. There will be n edges.
edges = (1:n)';
edges = [edges,edges+1];
% make that last edge wrap around
edges(end,2) = 1;

% Form a simplicial complex structure for eventual
% return. It will be a triangulation by the time we
% are done.
sc = simplicialcomplex(xy,edges);

% form triangles from each pair of consecutive edges
eartri = [edges,edges([2:end),1],2)];

% compute the included angle between the pairs
% of edges of each triangle in this list
ang12 = atan2(xy(eartri(:,1),2) - xy(eartri(:,2),2), ...
    xy(eartri(:,1),1) - xy(eartri(:,2),1));
ang23 = atan2(xy(eartri(:,3),2) - xy(eartri(:,2),2), ...
    xy(eartri(:,3),1) - xy(eartri(:,2),1));
ang = ang12 - ang23;

% if an angle is negative, we may either be traversing
% the polygon in a clockwise direction, or this may be a
% convex polygon. We need to figure out what is happening.
m = (ang < 0);
ang(m) = ang(m) + 2*pi;

% was the curve traversed clockwise or counter-clockwise?
% If the sum of the angles is now (n+2)*pi, then we
% have traversed the polygon in a clockwise sequence.
totalangle = sum(ang);
if abs(totalangle - (n+2)*pi)<(1e4*eps)
    % swap to a counterclockwise order
    ang = 2*pi - ang;
    eartri = eartri(:,[3 2 1]);
    totalangle = sum(ang);
end

% The sum of angles must be (n-2)*pi
% the polygon was traversed in a counterclockwise sequence. If
% the sum was zero, then the polygon must cross itself.
if abs((totalangle) - pi*(n-2)) > (10000*eps)
    % the polygon was not a proper one.
    error 'Holy polygon, Batman! Its an improper one - does it cross itself?'
end

% It is time to start clipping the polygon into triangles

% preallocate the triangulation array as

```

```

% a list of triangles.
tri = nan(n-2,3);
nt = 0;
ne = size(eartri,1);
while ne>3
    % pick that triangle with smallest (positive) included angle
    apos = ang;
    apos((ang<0)|(ang>pi)) = inf;
    [apos,angtags] = sort(apos);

    % will any of the other edges end up crossing the new
    % edge if we clip off the first triangle in this list?
    % If it does, then we need to pick a new triangle that
    % does not make that happen. We can always find one.
    % We do this by testing if any other vertex would have
    % fallen inside the prospective new triangle.
    npos = sum(~isinf(apos));
    failflag = true;
    for itri = 1:npos
        % search through the prospective triangles, starting
        % with the smallest included angle and working up.
        potentialtriangle = eartri(angtags(itri),:);

        allothervertices = setdiff(unique(eartri(:)),potentialtriangle(:));
        P1 = xy(potentialtriangle(1,:),:);
        M = [0 0 1;xy(potentialtriangle(2,:),:)-P1,1; ...
            xy(potentialtriangle(3,:),:)-P1,1]';
        % compute barycentric coordinate for all of them
        nothers = length(allothervertices);
        bcc = M\[xy(allothervertices,:)- repmat(P1,nothers,1),ones(nothers,1)]';
        if all(any(bcc<0,1) | any(bcc>1,1))
            % this triangle is acceptable
            failflag = false;
            break
        end
    end % for itri = 1:ntri
    if failflag
        % if we drop into here, then there was a problem with the polygon
        error('Failure in cleaving this polygon into triangles.')
    end

    % clip off the chosen triangle, add it to tri
    stri = potentialtriangle;
    nt = nt+1;
    tri(nt,:) = stri;

    eartri(angtags(itri),:) = [];
    ang(angtags(itri)) = [];

    % modify the pair of triangles that shared
    % an edge with the selected triangle.
    k3 = find(eartri(:,3) == stri(2));
    eartri(k3,3) = stri(3);
    k1 = find(eartri(:,1) == stri(2));
    eartri(k1,1) = stri(1);

```

```

% now update the included angles for triangles k1 and k3
k = [k1,k3];
ang12 = atan2(xy(eartri(k,1),2) - xy(eartri(k,2),2), ...
    xy(eartri(k,1),1) - xy(eartri(k,2),1));
ang23 = atan2(xy(eartri(k,3),2) - xy(eartri(k,2),2), ...
    xy(eartri(k,3),1) - xy(eartri(k,2),1));

angk = ang12 - ang23;
m = (angk < -pi);
angk(m) = angk(m) + 2*pi;
m = (angk > pi);
angk(m) = angk(m) - 2*pi;

ang(k) = angk;

ne = ne - 1;

end % while ne>3
tri(nt+1,:) = eartri(1,:);

% stuff the triangulation into sc.tessellation
sc.tessellation = tri;

```