A Portable Electronic Nose For Toxic Vapor Detection, Identification, and Quantification

B.R. Linnell, R.C. Young, T.P. Griffin, B.J. Meneghelli, B.V. Peterson, K.B. Brooks
Applied Chemistry Laboratory, NASA Kennedy Space Center, Florida 32899

Abstract

The Space Program and military use large quantities of hydrazine and monomethyl hydrazine as rocket propellant, which are very toxic and suspected human carcinogens. Current off-the-shelf portable instruments require 10 to 20 minutes of exposure to detect these compounds at the minimum required concentrations and are prone to false positives, making them unacceptable for many operations. In addition, post-mission analyses of grab bag air samples from the Shuttle have confirmed the occasional presence of on-board volatile organic contaminants, which also need to be monitored to ensure crew safety. A new prototype instrument based on electronic nose (e-nose) technology has demonstrated the ability to qualify (identify) and quantify many of these vapors at their minimum required concentrations, and may easily be adapted to detect many other toxic vapors. To do this, it was necessary to develop algorithms to classify unknown vapors, recognize when a vapor is not any of the vapors of interest, and estimate the concentrations of the contaminants. This paper describes the design of the portable e-nose instrument, test equipment setup, test protocols, pattern recognition algorithms, concentration estimation methods, and laboratory test results.

Keywords: Electronic nose, space program, hypergolic fuel, pattern recognition, classification, quantification.

Introduction

The Space Program and military use large quantities of hypergolic fuels such as hydrazine (Hz) and monomethyl hydrazine (MMH) as rocket propellant. These substances are very toxic and are suspected human carcinogens. The American Conference of Governmental Industrial Hygienists has set the threshold limit exposure value to 10 ppb[1]. Current off-the-shelf portable instruments not only require 10 to 20 minutes of exposure to detect 10 ppb concentrations, but they often react to other vapors (interferants) to give false positives, making these units unacceptable for many operations.

The ability to monitor air contaminants in a closed environment, such as the Shuttle, the International Space Station (ISS), and future human missions to Mars or the moon is important to ensure mission success and the safety of astronauts. Continuous air monitoring could provide notification of adverse events such as chemical spills or leaks. Post-mission analyses of grab air samples from the Shuttle have confirmed the occasional presence of on-board volatile organic contaminants (VOCs)[2]. The Spacecraft Maximum Allowable Concentration (SMAC)[3,4] was established as a guideline to maintain the air quality in spacecraft. To assure compliance to SMAC, continuous air monitoring must be performed, and specific compound identification and quantification is required. The VOCs listed in Table I are those used in this research, and are typical of the more than 60 compounds identified in Shuttle air samples so far[2].

Vapor	Abrv.	7-day SMAC (ppm)
Acetone	Ace	22
Isopropyl alcohol	IPA	60
Methylethyl Ketone	MEK	10
Toluene	Tol	16
Xylene	Xyl	50

Table I – Volatile Organic Compounds

A Portable Electronic Nose
For Toxic Vapor Detection, Identification, and Quantification

B.R. Linnell, R.C. Young, T.P. Griffin, B.J. Meneghelli, B.V. Peterson, K.B. Brooks
Applied Chemistry Laboratory, NASA Kennedy Space Center, Florida 32899

Abstract

The Space Program - ·
which are very
minutes of exp
positives, maki
samples from tl
also need to be
technology has
required concen
to develop algor
estimate the conc
test equipment se
laboratory test res

This is the entire original paper – the matlab code does not have to be included in what we send to Airsense.

Bruce

ropellant,
e 10 to 20
false
air
its, which
nose)
nimum
ecessary
est, and
rument,

Keywords: Electr

n.

We Have Friends In High Places

German company, AIRSENSE, that sold us the machinery used to gather data requested copies of final report

Space Flight Awareness

sfa.nasa.gov

Barbara Lee Duke

Introduction

The Space Progra ... military use large quantitie
monomethyl hydrazine (MMH) as rocket propellan
human carcinogens. The American Conference of (
threshold limit exposure value to 10 ppb[1]. Current
to 20 minutes of exposure to detect 10 ppb concentr
(interferants) to give false positives, making these u

d
ected

uire 10

The ability to monitor air contaminants in a closed e
Space Station (ISS), and future human missions to M
success and the safety of astronauts. Continuous air
events such as chemical spills or leaks. Post-mission
confirmed the occasional presence of on-board volatil
Maximum Allowable Concentration (SMAC)[3,4] was
in spacecraft. To assure compliance to SMAC, continuous air monitoring must be performed, and specific compound identification and quantification is required. The VOCs listed in Table I are those used in this research, and are typical of the more than 60 compounds identified in Shuttle air samples so far[2].

al

e
ave
raft
maintain the air quality

Vapor	Abrv.	7-day SMAC (ppm)
Acetone	Ace	22
Isopropyl alcohol	IPA	60
Methylethyl Ketone	MEK	10
Toluene	Tol	16
Xylene	Xyl	50

Table I – Volatile Organic Compounds

An electronic nose (e-nose) consists of an array of non-specific vapor sensors[5]. In general, the sensor array is designed such that each individual sensor responds to a broad range of chemicals, but with a unique sensitivity relative to the other sensors. Chemical identification is achieved by comparing the sensor response pattern of an unknown vapor to previously established patterns of known vapors. Different sensor types (metal oxide semiconductor, polymer composite, quartz microbalance, etc.) have different advantages and disadvantages – for example, some sensors are more sensitive to specific vapors, while others are less prone to drift due to changes in ambient conditions (e.g., temperature, RH, pressure). Many commercial e-noses have been trained to assign a quality value to flavors and food products, diagnosis certain diseases, and detect chemical spills, among other applications; however, few have been used to quantify the concentrations.

NASA at Kennedy Space Center (KSC) has assessed the sensitivity of several commercially available and pre-production e-noses. One very sensitive e-nose was found which is capable of identifying all of the hypergolic fuels and VOCs described above at the minimum required concentrations. It has been developed into a portable instrument capable of 8 hours of continuous operation, using a Palm Pilot for the user interface. This prototype has algorithms which allow it to classify unknown vapors with a high success rate, recognize when a vapor is not one of the vapors of interest, and accurately estimate the concentration of single vapors or mixtures of two vapors. The prototype unit has completed lab testing and is being field tested as a personnel safety monitor.

Methods

Generation of Calibrated Standard Vapors

Test vapors were generated using commercial vapor generators with permeation devices (PD) (Kintek Model 360, Austin TX), as shown in Figure 1. The PDs were maintained at a constant temperature and were purged continuously with dry nitrogen at 0.1 L/min. Flows were verified prior to tests using flow meters (SKC Accuflow, Eighty Four PA).

Vapors from the Kinteks were precisely blended with clean air from a temperature, relative humidity (RH) and flow rate controller (Miller-Nelson Model HCS-40, Monterey CA), which provided dilution up to a factor of 50. Activation of a solenoid valve allowed the vapor stream to mix with the clean air or not, thus generating either the test vapors or non-contaminated air. The resulting airstream (mixed or pure) was then drawn into the e-nose at 2 L/min.

Hz and MMH vapor concentrations were verified using an impinger filled with 0.1M H2SO4 to scrub a known volume of vapor. The Hz or MMH concentration in the resulting solution was determined by coulometric titration[6,7]. Organic vapor concentrations were determined by the weight loss of the PD during the operational period.

Earlier work has shown that stainless steel and other materials are incompatible with Hz or MMH vapors, especially at low concentrations[8], therefore the use of stainless steel tubing and fittings was kept to a minimum. All pneumatic lines were either Teflon or Bev-a-Line IV® tubing, which were shown to have minimal effect on these vapors, even in the low-ppb range.

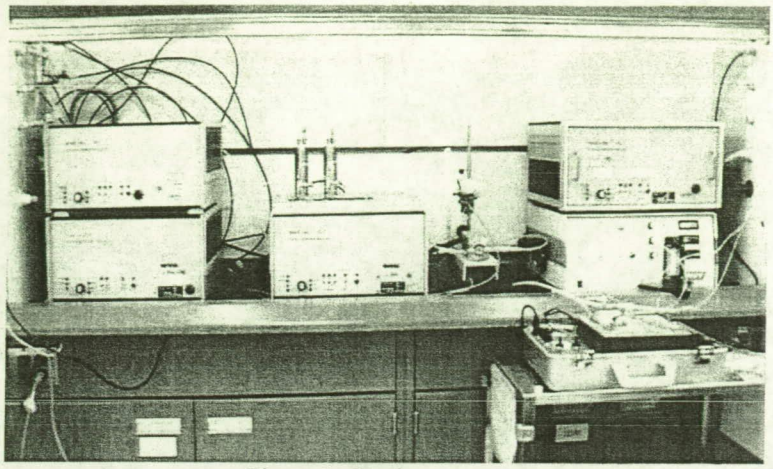


Figure 1. Basic test setup

E-nose Evaluation

A literature and market search of available e-noses was performed to identify instruments suitable for these applications. Both commercially available and pre-production e-noses were considered, which had to be lightweight and portable (see Table II). The initial evaluation was based upon the ability of the unit to detect organic and hypergolic fuel vapors at the minimum required levels. Instruments with sufficient sensitivity were then subjected to more rigorous testing, including an assessment of their analytic performance, evaluating a "trained" instrument's ability to identify test vapors using the vendor's software, and the development of algorithms to assess the information content of the raw data.

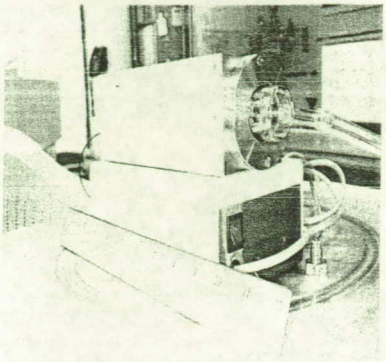
Instrument	Manufacturer	Array
i-Pen2	Air Sense	10 MOS
SamDetect	Daimler-Benz Aerospac	5 MOS
Cyranose	Cyrano Sciences	32 PC
KAMINA	Karlsruhe Research Center	38 MOS

Table II : E-nose instrumentation used in this study
MOS=metal oxide semiconductor, PC=polymer composite

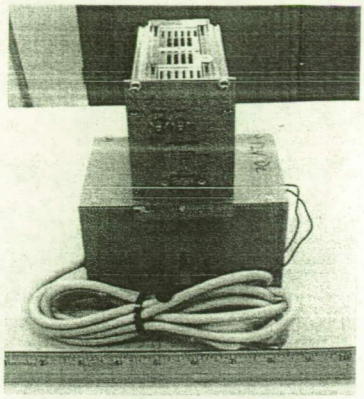
Several of the e-noses showed reasonable sensitivity to ppm levels of MMH, Hz, and the TOC's. However, of the instruments tested, only the KAMINA (Germany) and i-Pen (Germany), shown in Figure 2, were able to respond to 10 ppb levels of Hz and MMH with a signal to noise ratio greater than 3. Typical responses of the KAMINA and i-Pen to 10 ppb Hz and 10 ppb MMH are displayed in Figures 3 and 4 respectively. The i-Pen was selected for further development because the signal from the i-Pen is much less noisy, and the vendor was willing to provide the communications protocols for the development of an e-nose interface with a Palm Pilot.

Prototype Design and Fabrication

The basic OEM unit was fitted with an air pump (for drawing samples into the sensors), an 8-hour lithium-ion battery (for portability), an inlet filter (for establishing a baseline reading), and a Palm Pilot (to minimize size, weight, and startup time). The lithium-ion battery was selected to provide at least 8 hours of continuous operation for the lowest weight. The filter consists of glass wool soaked in a solution of 50% sulphuric acid and 50% water (by volume), suctioned to dampness, and dried at 60 degrees C.



KAMINA



i- Pen

Figure 2. Vendor units for screening test

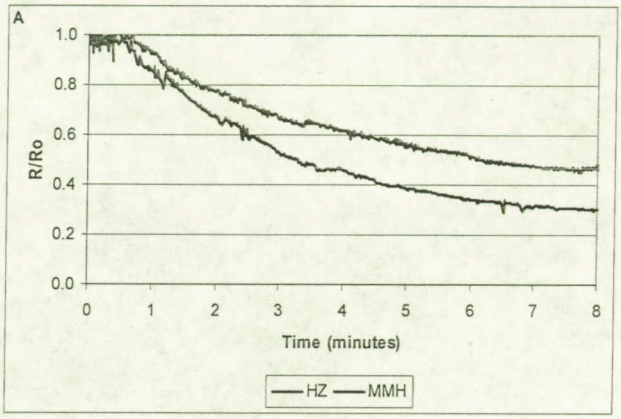


Figure 3. Response of the KAMINA to 10 ppb of Hz and MMH. The average of the 38 sensors are plotted as R/Ro where R is the sensor response at any point in time and Ro is the response of the sensor in clean air

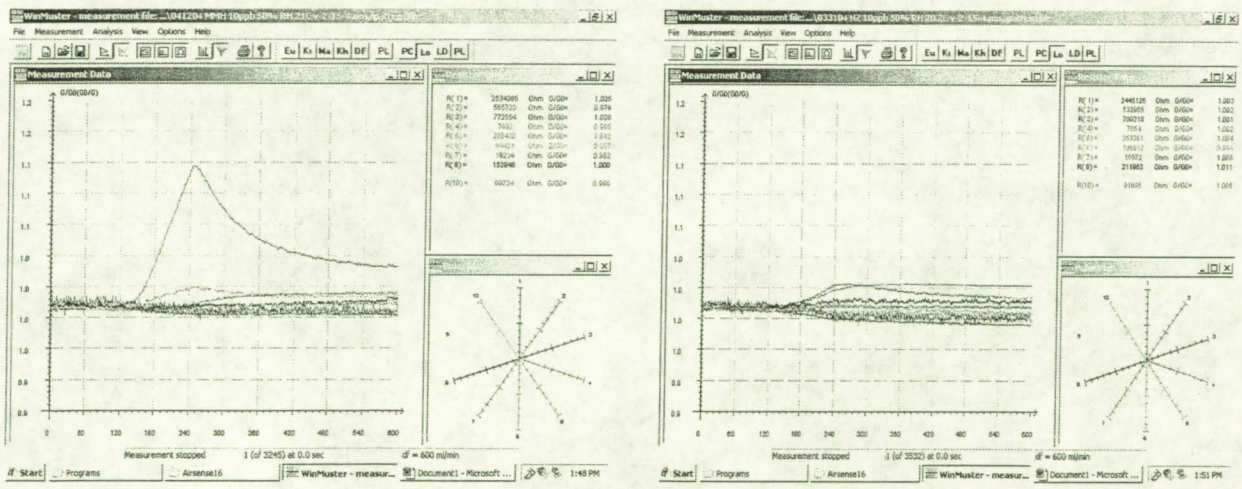


Figure 4. Response of the i-Pen to 10 ppb of MMH (left) and Hz (right). The 10-sensor response plot, the sensor values, and radial plot are shown.

Figure 5 shows a schematic diagram of the prototype unit, and Figure 6 shows some photos of the unit.

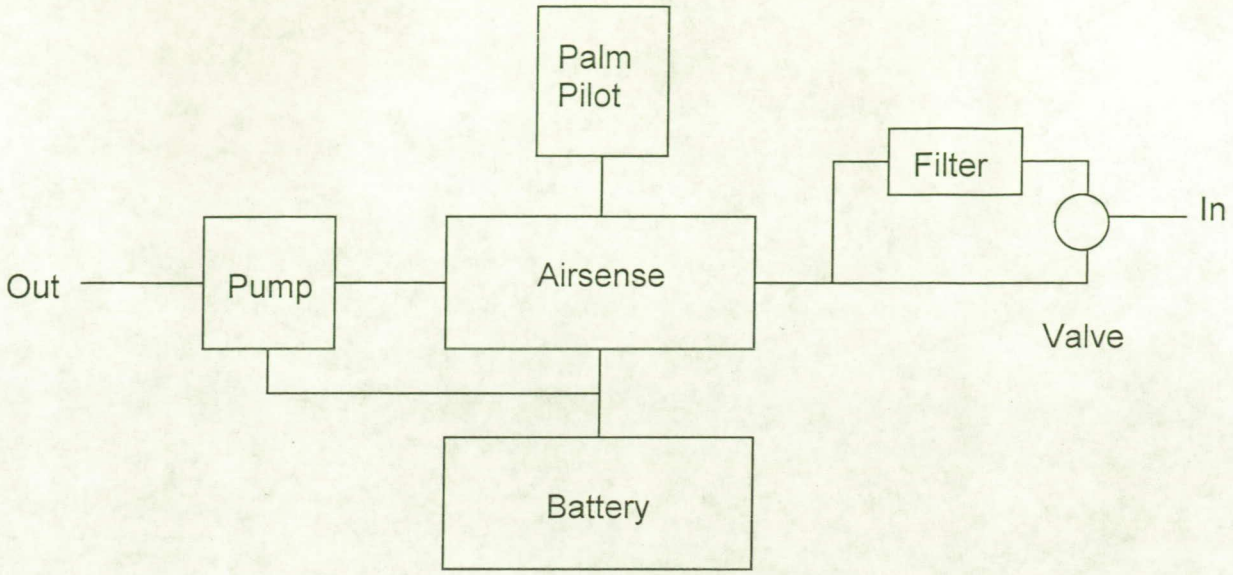


Figure 5. Prototype schematic

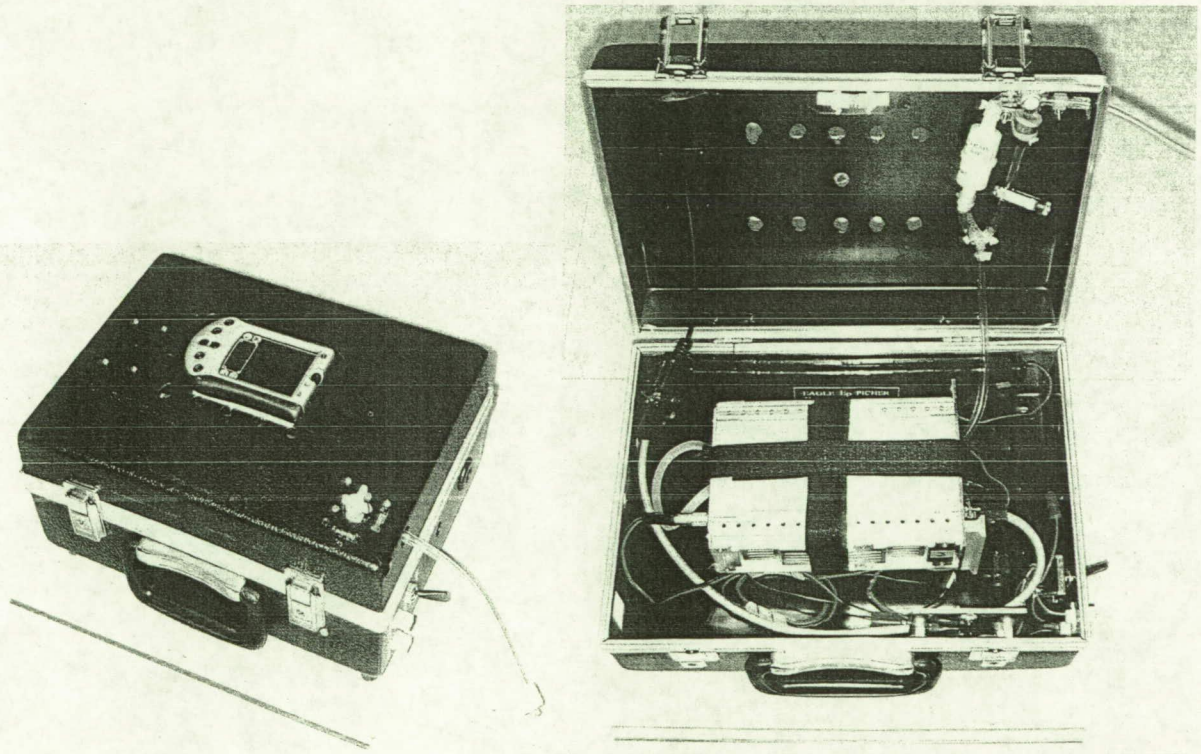


Figure 6. Prototype unit

Training and Validation Data

The i-Pen was trained using four concentrations of each vapor at three RHs (see Table III). In all cases, a lab computer was used to acquire the training data for the models. Due to time constraints, mixture vapors were only gathered for Ace+IPA, MEK+Tol, MEK+Xyl, and Tol+Xyl at the concentrations listed in Table III.

Vapor	Conc. Used (ppm)	RHs (%)
MMH	0.0072, 0.047, 0.095, 0.52	50, 70, 85
Hz	0.011, 0.048, 0.095, 0.46	50, 70, 85
Ace	4.7, 6.7, 12, 23	20, 50, 85
IPA	2.2, 3.2, 5.6, 11	20, 50, 85
MEK	1.9, 2.7, 4.8, 9.5	20, 50, 85
Tol	2.3, 3.3, 5.8, 11	20, 50, 85
Xyl	1.6, 2.3, 4.0, 7.9	20, 50, 85

Table III – Model Vapor Concentrations and Humidities

For the hypergolic fuels, validation data using Hz and MMH vapors at various concentrations and RHs (to be described in the Results section) were then gathered using the Palm Pilot, as it would be used in the field. A lab computer was used to acquire the VOC validation data, which also spanned a range of concentrations and RHs (also described in Results).

Feature Extraction

Raw e-nose data consists of a time-varying curve for each sensor (see Figure 7). In pattern classification, a "feature" is any direct or derived measurement of the system that helps differentiate between classes (in this case, vapors). Many different possible features can be extracted from e-nose data, but it has been shown that either the final value or maximum initial slope (shown in Figure 7) are among the best single features for discriminating between classes[9]. However, the maximum initial slope is very sensitive to noise, which can be significant at low vapor concentrations. While the final value is robust and simple to calculate, most sensors require a long time to stabilize (one to 20 minutes depending on the odor, intensity, and sensor). For this research, the sensor values at 90 seconds after the start of the sniff were used, which is much less than the time needed for the sensors to reach steady-state.

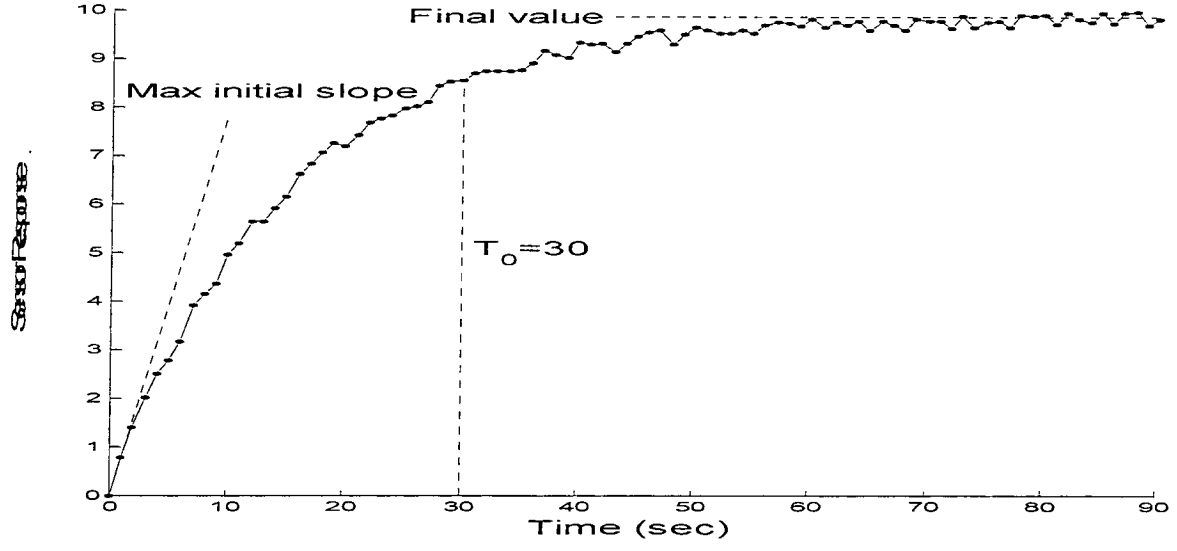


Figure 7 – E-nose Features

Single-Vapor Identification Algorithm

Software was developed at KSC to provide vapor identification (described below) and quantification (discussed subsequently). All the vendor-supplied software was found to be inadequate for identification, and unable to perform quantification.

Code was written for the Palm Pilot to implement a standard quadratic classifier[10]. Given class i with mean μ_i and covariance matrix $\sum_i$, an unknown sample x is assigned to that class with the smallest value of

$$(x-\mu_i)^T\sum_i^{-1}(x-\mu_i) + \ln(|\sum_i|)$$

where x^T indicates a vector transpose and $|\sum_i|$ indicates the determinant of $\sum_i$.

After a class has been selected, the square of the Mahalanobis distance[10] $(x-\mu_i)^T\sum_i^{-1}(x-\mu_i)$ from the example to the estimated class is calculated, and compared to a statiscially determined threshold. If the example is too far from the assumed class, the example is rejected as an "unknown" vapor (i.e., the vapor is not one of the known vapors in the model).

Single-Vapor Quantification Algorithm

To estimate the concentration of a single vapor, the model data was plotted to show sensor response as a function of concentration, shown in Figure 8. This data was then fitted (using the Levenberg-Marquardt nonlinear least-squares algorithm) to the formula $y=A(1-e^{Bx})$, to find the values for the parameters A and B which best relate the sensor response y to the vapor concentration x. This formula was determined to be appropriate because the sensor response should go to zero as the concentration goes to zero, and the sensor response should saturate at high enough concentrations. Other formulas with similar qualities were tested, but did not perform as well.

When presented with an unknown sample, the inverse of this formula was then used to convert each of the ten sensor responses into ten concentration estimations. Many different techniques were explored to convert the ten estimates into a single value, including taking the mean, the mean weighted by the quality of the curve fit, and multiple linear regression, but it turned out that using the estimate of one particular sensor was usually the best. Which sensor to use depended on the vapor, and was determined by analyzing the model data.

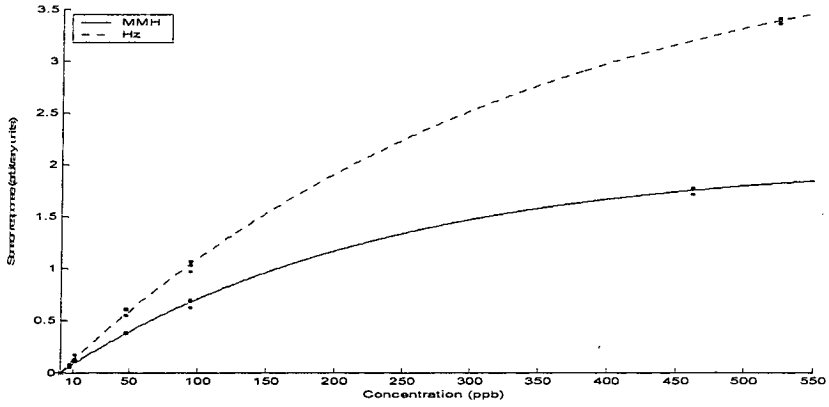


Figure 8. Concentration curve fitting (one sensor)

Note that with this approach, if the unknown vapor is misidentified, the concentration estimate will be invalid.

Two-Vapor Identification/Quantification Algorithm

In order to build a model of how the sensors respond to a mixture of two vapors, sensor responses were gathered for each vapor of interest, and for all pairs of vapors (in a 50/50 mix), across a range of known concentrations (see Figure 9 and Table VIII). For each pair of vapors, the response of each sensor is plotted vs. the concentration (creating a surface, also shown in Figure 9), and an equation of the form $z=A + (B*x^C + D*y^E)^F$ is fitted to the data (again using the Levenberg-Marquardt, or LM, algorithm) to find the least-squares values of the parameters A,B,C,D,E,F which relate the concentrations x and y to the sensor response z. This formula has been found by other researchers[11-12] to be the best model of sensor response to mixtures of vapors. Other formulas with similar properties were evaluated, but did not perform as well.

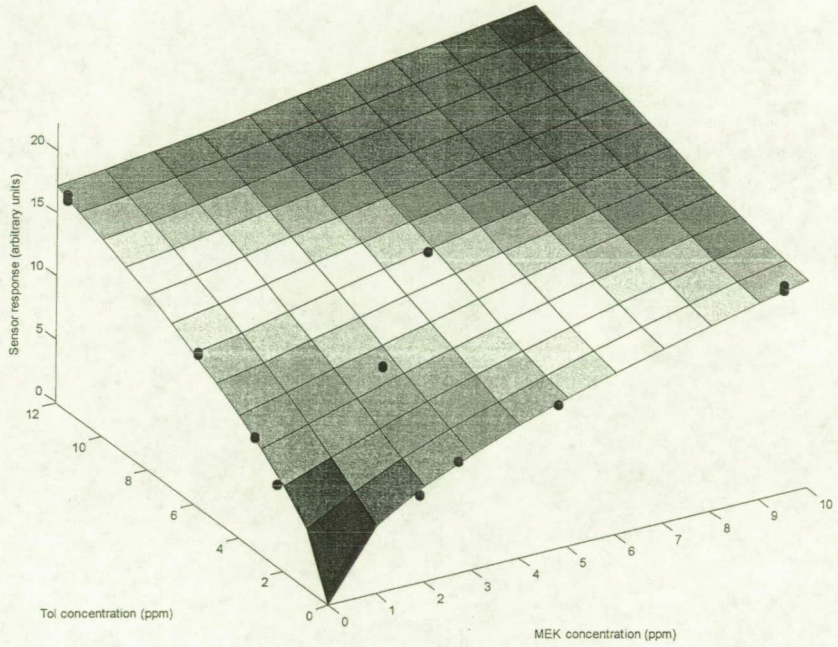


Figure 9 – Two-vapor concentration model surface for MEK+Tol, sensor #9

Given an unknown example, the first step is to calculate the difference between each sensor value of the unknown and the modeled surface for that sensor, for all possible vapor pairs (Ace+IPA, MEK+Tol, etc.). The lowest point on this error surface represents the most likely concentrations which produced the given sensor value, for that sensor and vapor pair. Next, for each vapor pair, the multiple error surfaces (one per sensor) are combined to create a cumulative error surface. Besides selecting which sensors to leave out based on lack of response or saturation, addition of normalized and un-normalized error surfaces were tested.

For each vapor pair's cumulative error surface, the minimum point on that surface must be found. To facilitate rapid algorithm development, finding the optimum point has been done so far by sampling values on a grid, then using the minimum value. Future enhancements include more sophisticated searches such as gradient descent, which has been determined to be optimal since the error surface has only one minimum.

The "quality" of each minimum for all vapor pairs is then determined. The best metric turned out to be the value of the surface at the minimum, although various normalizations of this value were tried as well. The vapor pair whose minimum has the best quality is then selected as the identified classes. The location of the minimum within that pair's error surface determines the estimated concentrations.

For example, given an unknown which is a mixture of 5.8 ppm Tol and 4.0 ppm Xyl, the cumulative error surfaces are shown in Figure 10 for Ace+IPA and Tol+Xyl. As can be seen, the minimum point on the Tol+Xyl surface is much lower than the minimum point on the Ace+IPA surface, so Tol+Xyl would be chosen as the most likely vapor pair. The minimum point on the Tol+Xyl error surface occurs at (5.81, 3.99), which would be taken to be the most likely concentrations of the vapors.

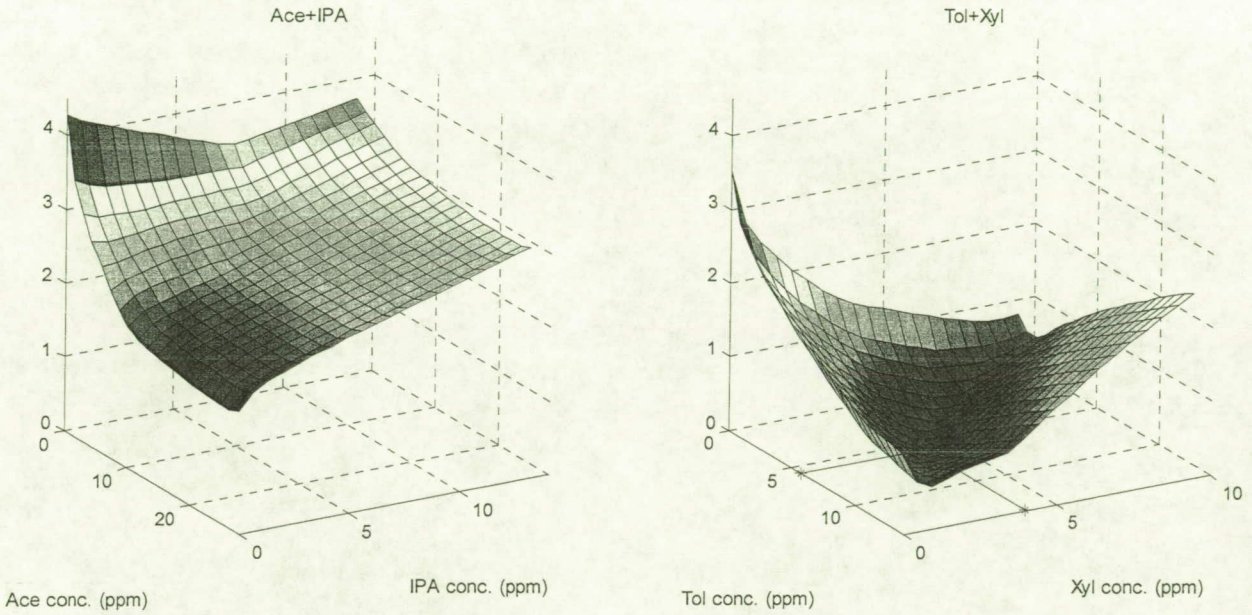


Figure 10 – Picking vapor pairs and concentration estimates from cumulative error surfaces

Results

Hypergolic Fuels, Single-Vapor, Estimated Performance

In order to determine how well the classes can be differentiated, some estimate of the classifier's future performance is required[13]. None of the software provided by the vendors could automatically calculate this value.

If all the data is used to build the model, and is then also used to estimate the success rate (known as "Resubstitution", abbreviated "Resub"), the estimate will generally be too optimistic. This problem is usually solved by using techniques such as "N-Fold Cross Validation", which sets aside part of the data, builds a model with the remaining data, and uses the first part to estimate the performance. N different portions are set aside, and the N estimates are then averaged (N=3-10 is typical). If N equals the number of examples, the result is the "Leave-One-Out" estimator (LOO), which is often called simply "cross validation". However, when there is a relatively large amount of data, all estimation methods will return approximately the same value[14].

The same methods were used to estimate the average %error in quantification; however when estimating the concentration it is assumed that the vapor is correctly identified. To avoid artificially inflated error

rates, two values are shown in Table IV. The first is the mean |%error| for all concentrations greater than 10 ppb, and the second is the mean |error| (in ppb) for the 10 ppb examples. This is because errors of just a few ppb have very large %errors at 10 ppb, which is at the limit of detection for the e-nose. These results are across all vapors, concentrations, and humidities.

Result	Resub	LOO
Classification success	99%	98%
Quantification error	3.5%, 2.5 ppb	4.1%, 2.5 ppb

Table IV – Hypergolic single vapor estimated performance

Hypergolic Fuels, Single-Vapor, Validation Performance

All validation tests were performed at 70% RH. Appendix A shows the results for each individual test. Table V shows the summary statistics for each vapor and concentration. The mean concentration errors shown are only for those examples that were correctly classified, and are again separated into 10 ppb and greater-than-10-ppb values. Note that the 250 ppb vapors were not part of the training data, but are accurately identified and quantified.

Vapor	Std. Conc. (ppb)	% correctly identified	Mean |%error| or |error|
Hz	7.2	50	0.4 ppb
Hz	95	100	5.5%
Hz	250	100	1.3%
Hz	524	100	2.8%
MMH	11	83	2.6 ppb
MMH	95	100	10%
MMH	250	100	9%
MMH	461	100	1.3%
Overall	-----	**92**	**5.0%, 1.8 ppb**

Table V – Hypergolic single vapor validation performance

Organic, Single-Vapor, Estimated Performance

The results are shown in Table VI, across all vapors, concentrations, and humidities. When estimating the concentration, it is assumed that the vapor has been correctly identified. For this and all subsequent VOC results, only mean |%error| is given since none of the the lowest concentrations were near the limit of detection.

Result	Resub	LOO
Classification success	100%	100%
Quantification error	5.3%	5.9%

Table VI – Organic single vapor estimated performance

Organic, Single-Vapor, Validation Performance

Appendix B shows the results for each individual example. As can be seen from Appendix B, a wide but not exhaustive combination of concentrations and humidities were tested. Table VII shows the summary statistics for each vapor and concentration. When estimating the concentration, it is assumed that the vapor has been correctly identified.

Vapor	Std. Conc. (ppm)	% correctly identified	Mean |%error|
Ace	4.7	100	12.7%
Ace	6.7	100	7.2%
Ace	12	100	4.0%
Ace	23	100	1.6%
IPA	2.2	100	36.3%
IPA	3.2	83	27.1%
IPA	5.6	100	25.4%
IPA	11	100	15.5%
MEK	1.9	100	36.8%
MEK	2.7	100	25.8%
MEK	4.8	100	32.6%
MEK	9.5	100	27.9%
Tol	2.3	100	4.3%
Tol	3.3	100	1.8%
Tol	5.8	100	4.6%
Tol	11.5	100	9.5%
Xyl	1.6	0	6.3%
Xyl	2.3	50	7.5%
Xyl	4.0	100	4.1%
Xyl	7.9	50	15.1%
Overall	-----	**91**	**14.8%**

Table VII – Organic single vapor validation performance

A question arose as to whether the poorer results of the validation data could be due to sensor "drift", a common problem with e-noses. While the validation data was started only two weeks after the model data was finished, the success rate in classification shown in Table VII is deceiving, because the Mahalanobis distance is not taken into account. When that is done, it turns out that every example in the validation set is considered different enough from the model to be classified as an "unknown". Therefore the validation set might not be similar enough to the model for the results in Table VII to be considered relevant.

Organic, Two-Vapor, Estimated Performance

All examples were taken at 50% RH. Table VIII shows the summary results, where "Both Classified" indicates the percentage of the unknowns for which both vapors were correctly identified. For single-vapor unknowns, "Both Classified" indicates that the second vapor's concentration was determined to be zero. In all cases, "Both Classified" means the unknown was completely identified correctly. Unlike the previous estimation results, these results are given by vapor and concentration to show trends better.

Because the LM algorithm took significantly longer to process the two-variable surface equation, Leave-One-Out testing was not performed. Resubstitution was determined to be an acceptable estimate here, because there were 52 examples used to create each surface, and the removal of any one of them would not significantly alter the resulting model parameters of the surface. As can be seen in the other estimated performance results above (Tables IV and VI), Resubstitution and Leave-One-Out give very similar results when there are a large number of examples.

Vapor #1	Conc. (ppm)	Vapor #2	Conc. (ppm)	Both Classified (%)	Mean |%error|
Ace	4.7			100	23.6
Ace	6.7			100	8.4
Ace	12.0			100	7.5
Ace	23.0			100	5.8
IPA	2.2			100	21.1
IPA	3.2			100	9.7
IPA	5.6			100	3.9
IPA	11.0			100	2.1
MEK	1.9			100	5.3
MEK	2.7			100	11.1
MEK	4.8			100	9.4
MEK	9.6			100	4.2
Tol	2.3			100	13.2
Tol	3.3			100	10.0
Tol	5.8			100	12.7
Tol	11.5			100	5.8
Xyl	1.6			100	6.4
Xyl	2.3			100	8.5
Xyl	4.0			100	12.4
Xyl	7.9			100	1.1
Ace	4.7	IPA	2.2	100	6.3
Ace	6.7	IPA	3.2	100	10.9
Ace	12.0	IPA	5.6	100	18.1
Ace	23.0	IPA	11.0	100	8.7
MEK	1.9	Tol	2.3	no data	no data
MEK	2.7	Tol	3.3	100	13.3
MEK	4.8	Tol	5.8	100	12.6
MEK	9.6	Tol	11.5	100	14.2
MEK	1.9	Xyl	1.6	100	15.0
MEK	2.7	Xyl	2.3	100	26.3
MEK	4.8	Xyl	4.0	100	12.4
MEK	9.6	Xyl	7.9	100	14.2
Tol	2.3	Xyl	1.6	100	9.8
Tol	3.3	Xyl	2.3	100	11.6
Tol	5.8	Xyl	4.0	100	4.9
Tol	11.5	Xyl	7.9	100	21.3
Overall	----	----	----	**100**	**10.2**

Table VIII – Organic Two Vapor Estimated Performance (Resubstitution)

Organic, Two-Vapor, Validated Performance

Validation data was gathered at concentrations that were not included in the model, for both single-vapor and two-vapor mixtures. All examples were taken at 50% RH. Results for each example are given in Appendix C, and summarized in Table IX, where "One Classified" indicates that one vapor was correctly identified and one was misidentified, and "None Classified" means that both vapors were misidentified. For single-vapor unknowns, "One Classified" means either that the first vapor was misidentified, or the

second vapor's estimated concentration was non-zero. The %error was calculated only for those non-zero vapors which were correctly identified.

Vapor #1	Conc. (ppm)	Vapor #2	Conc. (ppm)	Both Classified (%)	One Classified (%)	None Classified (%)	|%error|
Ace	13.2			0	100	0	55.8
Ace	17.5			0	100	0	40.5
IPA	6.3			100	0	0	15.3
IPA	8.3			100	0	0	4.8
MEK	5.4			0	100	0	49.5
MEK	7.2			0	100	0	42.9
Tol	6.5			0	100	0	49.5
Tol	8.7			0	100	0	45.3
Xyl	4.5			0	100	0	46.3
Xyl	6			0	100	0	33.4
Ace	13.2	IPA	6.3	100	0	0	42.6
Ace	17.5	IPA	8.3	100	0	0	35.7
MEK	5.4	Tol	6.5	0	100	0	5.3
MEK	7.2	Tol	8.7	0	100	0	9.7
MEK	5.4	Xyl	4.5	100	0	0	17.9
MEK	7.2	Xyl	6	100	0	0	9.6
Tol	6.5	Xyl	4.5	100	0	0	26.8
Tol	8.7	Xyl	6	100	0	0	21.5
Overall	----	----	----	**31**	**69**	**0**	**31.7**

Table IX – Organic Two Vapor Validation Performance

This data was taken three to four months after the original model data. As with the single-vapor VOC validation data, all of the validation examples used here (single and mixed) were significantly different enough from the model data to be considered "unknowns" when the Mahalanobis distance is included, and so these results may not be relevant.

Resistance to False Positives

Four samples were taken of each of the organic vapors at the minimum concentrations listed in Table III, and classified using the hypergolic fuel training samples as the model. As can be seen in Table X, the organic vapors are identified as not belonging to either MMH or Hz because the Mahalanobis distance is too large (vapors with distances greater than 5.07 are considered "unknown").

Vapor	Mahalanobis distances
MMH	1.4 - 2.9
Hz	0.8 - 3.8
Ace	328.2 - 365.7
IPA	41.7 - 50.0
MEK	157.0 - 164.1
Tol	178.6 - 193.6
Xyl	572.6 - 633.4

Table X – False positive results

Discussion

Single Vapors
In all cases, the estimated performance was slightly better than the validation performance. The estimated classification success rate was 98%-100%, and the validation success rate was 91%-92%. Excluding the 10 ppb hypergols, the |%error| was about 3%-6% except for the validated organics, where it was 15%. However, as discussed above, the relevance of the VOC validation data is very questionable.

Other studies[15-16] which have attempted to quantify single-vapor e-nose data have shown poor performance in identification and/or quantification. One reason for our good results is due to our method of using a very accurate standard statistical pattern recognition technique to first identify the vapor, whereas other studies used non-standard methods to identify the vapor. Another improvement comes from the choice of formula the concentration data is fitted to. Other studies used cubic splines or polynomials, which do not model the underlying trend of the data very well.

Mixture Vapors
For the mixtures, the validation results differed significantly from the estimated results. The estimated results show a 100% success in identifying both vapors, with an average %error of 10%, while the validation results show only a 31% chance of identifying both vapors, with an average %error of 32%. However, as discussed above, the VOC mixture validation data is significantly different than the model data, and is probably not relevant.

Previous studies[17-21] which have attempted to quantify e-nose data for vapor mixtures have had similar or better accuracy in quantification, but they all started with the assumption that the identity of the vapors in the mixture was already known. As far as is known, the algorithm presented here is unique in its ability to identify the mixtures as well as quantify them. In addition, research on this algorithm is not finished, and with further development the results presented here could be improved.

Conclusions

A prototype portable e-nose capable of detecting 10 ppm MMH and Hz has been developed at KSC NASA. It is capable of detecting, identifying, and quantifying vapors in only 90 seconds, with a 1 to 10 minute recovery time (depending on the concentration of the exposure). This unit classifies single vapors with 90-100% accuracy, and quantifies them with an average of about 5% error, except at the limit of detection (10 ppb), where the error is less than 3 ppb. This unit also shows excellent resistance to false positives, and may be trained to detect, identify, and quantify virtually any vapor of interest, within the detection limits of the sensors. It also shows great promise in being able to accurately identify and quantify mixtures of two vapors and possibly more, depending on future research.

References

(1) ACGIH, *Documentation of the Threshold Limit Values and Biological Exposure Indices,* 5th ed, American Conference of Governmental Industrial Hygienists, Cincinnati, Ohio, 2002.

(2) J. James, T. Linero, H. Leano, J. Boyd, P.Covington, Volatile Organic Contaminants Found in the Habitable Environment of the Space Shuttle: STS-26 to STS-55, Aviation, Space, and Environmental Medicine, September 1994, pp. 851-857.

(3) Spacecraft Maximum Allowable Concentrations for Airborne Contaminants, JSC20584, NASA Johnson Space Center, May 1990.

(4) Safety Requirements Document for International Space Station Program, SSP50021, NASA Johnson Space Center, December 12, 1995.

(5) H. Nagle, R. Gutierrez-Osuna, S. Schiffman, "The How and Why of Electronic Noses", IEEE Spectrum, September 1998, pp. 22-34.

(6) J. Wyatt, S. Rose-Pehrsson, T. Cecil, K. Crossman, N. Mehta, R. Young, "Coulometric Method for the Quantification of Low-Level Concentrations of Hydrazine and Monomethylhydrazine", American Industrial Hygiene Association Journal, June 1993, pp. 285-292

(7) "Determination of Concentrations of N2H4 or MMH Vapor in Nitrogen or Air by the Columnetric Titration Method", Applied Chemistry Laboratory, Kennedy Space Center, internal document.

(8) P. Taffe, S. Rose-Pehrsson, J. Wyatt, "Material Compatibility with Threshold Limit Value Levels of Monomethyl Hydrazine", NRL Memorandum Report 6291, October 1988.

(9) S. Roussel, G. Forsberg, V. Steinmetz, P. Grenier, V. Bellon-Maurel, "Optimization of Electronic Nose Measurements, Part I : Methodology of Output Feature Selection", Journal of Food Engineering, Vol. 37, pp. 207-222, 1998.

(10) R.O. Duda and P.E. Hart, "Pattern Classification and Scene Analysis", John Wiley&Sons, New York, 1973.

(11) L. Lundstrom, "Hydrogen-sensitive MOS-structures, Part I : Principles and Applications", Sensors and Actuators, Vol. 1, p. 403-426, 1981.

(12) P.K. Clifford and D.T. Tuma, "Characteristics of Semiconductor Gas Sensors. I : Steady-state Gas Response", Sensors and Actuators, Vol. 3, p. 233-254, 1982/1983.

(13) G. Toussaint, "Bibliography on Estimation of Misclassification", IEEE Transactions on Information Theory, Vol. 20, pp. 472-479, 1974.

(14) N. Glick, "Additive Estimators for Probabilities of Correct Classification", Pattern Recognition, Vol. 10, pp. 211-222, 1978.

(15) M.A. Ryan, H. Zhou, M.G. Buehler, K.S. Manatt, V.S. Mowrey, S.P. Jackson, et.al., "Monitoring Space Shuttle Air Quality Using the Jet Propulsion Laboratory Electronic Nose", IEEE Sensors Journal, Vol. 4, No. 3, p. 337-347, June 2004.

(16) L. Carmel, N. Sever, D. Lancet, D. Harel, "An eNose Algorithm for Identifying Chemicals and Determining Their Concentration", Sensors and Actuators B, 93, p. 77-83, 2003.

(17) H. Zhou, M.A. Ryan, M.L. Homer, "Nonlinear Least Squares Based Algorithm for Identifying and Quantifying Single and Mixed Air Contaminants with JPL's Electronic Nose", to appear in IEEE Sensors.

(18) W.P. Carey, S.S. Yee, "Calibration of Nonlinear Solid-State Sensor Arrays Using Multivariate Regression Techniques", Sensors and Actuators B, 9, p. 113-122, 1992.

(19) G. Huyberechts, P Szecowka, J Roggen, B.W. Licznerski, "Simultaneous Quantification of Carbon Monoxide and Methane in Humid Air Using a Sensor Array and an Artificial Neural Network", Sensors and Actuators B, 45, p. 123-130, 1997.

(20) G. Faglia, F. Bicelli, G Sberveglieri, P Maffezzoni, P. Gubian, "Identification and Quantification of Methane and Ethyl Alcohol in an Environment at Variable Humidity by an Hybrid Array", Sensors and Actuators B, 44, p. 517-520, 1997.

(21) B.W. Jervis, J. Desfieus, J. Jimenez, D. Martinez, "Quantification of Gas Concentrations in Mixtures of Known Gasses Using an Array of Different Tin-Oxide Sensors", IEE Proceedings-Science, Measurement, and Technology, Vol. 150, No. 3, p. 97-106, May 2003.

Acknowledgements

Funding for these projects was provided by NASA.

Appendix A – Individual Validation Results for Hypergolic Fuels, Single-Vapor

Vapor	RH (%)	Std. Conc. (ppb)	Identified as	Conc. Reading (ppb)	Error (ppb)	%error
Hz	70	7.2	Hz	7	-0.2	-2.7
Hz	70	7.2	Hz	8	0.8	11
Hz	70	7.2	MMH	10	N/A	N/A
Hz	70	7.2	Hz	7	-0.2	-2.7
Hz	70	7.2	MMH	9	N/A	N/A
Hz	70	7.2	MMH	8	N/A	N/A
Hz	70	95	Hz	95	0	0
Hz	70	95	Hz	101	6	6.3
Hz	70	95	Hz	98	3	3.2
Hz	70	95	Hz	102	7	7.4
Hz	70	95	Hz	102	7	7.4
Hz	70	95	Hz	103	8	8.4
Hz	70	250	Hz	249	-1	-0.4
Hz	70	250	Hz	251	1	0.4
Hz	70	250	Hz	251	1	0.4
Hz	70	250	Hz	245	-5	-2.0
Hz	70	250	Hz	246	-4	-1.6
Hz	70	250	Hz	243	-7	-2.8
Hz	70	524	Hz	532	8	1.5
Hz	70	524	Hz	528	4	0.8
Hz	70	524	Hz	510	-14	-2.7
Hz	70	524	Hz	514	-10	-1.9
Hz	70	524	Hz	488	-36	-6.9
Hz	70	524	Hz	507	-17	-3.2
MMH	70	11	MMH	9	-2	-18
MMH	70	11	MMH	10	-1	-9
MMH	70	11	MMH	8	-3	-27
MMH	70	11	MMH	7	-4	-36
MMH	70	11	Hz	4	N/A	N/A
MMH	70	11	MMH	8	-3	-27
MMH	70	95	MMH	84	-11	-12
MMH	70	95	MMH	84	-11	-12
MMH	70	95	MMH	85	-10	-10
MMH	70	95	MMH	87	-8	-8.4
MMH	70	95	MMH	85	-10	-10
MMH	70	95	MMH	88	-7	-7.4
MMH	70	250	MMH	227	-23	-9.2
MMH	70	250	MMH	228	-22	-8.8
MMH	70	250	MMH	229	-21	-8.4
MMH	70	250	MMH	225	-25	-10
MMH	70	250	MMH	226	-24	-9.6
MMH	70	250	MMH	230	-20	-8
MMH	70	461	MMH	465	4	0.9
MMH	70	461	MMH	452	-9	-1.9
MMH	70	461	MMH	466	5	1.1
MMH	70	461	MMH	472	11	2.4
MMH	70	461	MMH	456	-5	-1.1
MMH	70	461	MMH	464	3	0.6

Appendix B – Individual Validation Results for Organics, Single-Vapor

Vapor	RH (%)	Std. Conc. (ppm)	Identified as	Conc. Reading (ppb)	Error (ppb)	%error
Ace	20	6.7	Ace	5.9	-0.8	-11.8
Ace	20	6.7	Ace	5.9	-0.8	-12.6
Ace	20	6.7	Ace	5.9	-0.8	-12.6
Ace	20	23	Ace	22.6	-0.4	-1.8
Ace	20	23	Ace	22.9	-0.1	-0.5
Ace	20	23	Ace	23	0	0.2
Ace	50	4.7	Ace	4	-0.7	-14.4
Ace	50	4.7	Ace	4.1	-0.6	-12.5
Ace	50	4.7	Ace	4.3	-0.4	-8.5
Ace	50	12	Ace	11.4	-0.6	-4.7
Ace	50	12	Ace	11.5	-0.5	-4.1
Ace	50	12	Ace	11.6	-0.4	-3.3
Ace	85	6.7	Ace	6.5	-0.2	-3.7
Ace	85	6.7	Ace	6.7	0	-0.3
Ace	85	6.7	Ace	6.8	0.1	2.1
Ace	85	23	Ace	23	0	0.2
Ace	85	23	Ace	23.7	0.7	2.9
Ace	85	23	Ace	24	1	4.4
IPA	20	3.2	IPA	2.5	-0.7	-21.9
IPA	20	3.2	IPA	2.6	-0.6	-20.3
IPA	20	3.2	TCE	2.6	-0.6	-20.3
IPA	20	11	IPA	9.5	-1.5	-13.9
IPA	20	11	IPA	9.6	-1.4	-13
IPA	20	11	IPA	10	-1	-9.3
IPA	50	2.2	IPA	1.4	-0.8	-36.1
IPA	50	2.2	IPA	1.4	-0.8	-36.9
IPA	50	2.2	IPA	1.5	-0.7	-31.9
IPA	50	5.6	IPA	4.1	-1.5	-26.1
IPA	50	5.6	IPA	4.1	-1.5	-26.8
IPA	50	5.6	IPA	4.3	-1.3	-23.4
IPA	85	3.2	IPA	2	-1.2	-37.6
IPA	85	3.2	IPA	2.2	-1	-31.2
IPA	85	3.2	IPA	2.2	-1	-31.7
IPA	85	11	IPA	8.8	-2.2	-20.5
IPA	85	11	IPA	8.9	-2.1	-19.5
IPA	85	11	IPA	9.1	-1.9	-17
MEK	20	2.7	MEK	2.2	-0.5	-17.5
MEK	20	2.7	MEK	2.3	-0.4	-14.3
MEK	20	2.7	MEK	2.3	-0.4	-16
MEK	20	9.5	MEK	7	-2.5	-26.7
MEK	20	9.5	MEK	7.1	-2.4	-24.9
MEK	20	9.5	MEK	7.2	-2.3	-23.7
MEK	50	1.9	MEK	1	-0.9	-46.8
MEK	50	1.9	MEK	1.2	-0.7	-34.4
MEK	50	1.9	MEK	1.2	-0.7	-34.4
MEK	50	4.8	MEK	3.2	-1.6	-33.7

MEK	50	4.8	MEK	3.2	-1.6	-34.3
MEK	50	4.8	MEK	3.4	-1.4	-29.7
MEK	85	2.7	MEK	1.7	-1	-35.2
MEK	85	2.7	MEK	1.7	-1	-38.2
MEK	85	2.7	MEK	1.8	-0.9	-33.9
MEK	85	9.5	MEK	6.4	-3.1	-32.8
MEK	85	9.5	MEK	6.4	-3.1	-33.1
MEK	85	9.5	MEK	7	-2.5	-26.3
TCE	20	5	TCE	4.4	-0.6	-12.5
TCE	20	5	TCE	4.5	-0.5	-10.9
TCE	20	5	TCE	4.7	-0.3	-6.8
TCE	20	17	TCE	17.9	0.9	5.1
TCE	20	17	TCE	18.3	1.3	7.5
TCE	20	17	TCE	18.6	1.6	9.3
TCE	50	3.5	TCE	2.6	-0.9	-24.8
TCE	50	3.5	TCE	2.8	-0.7	-20.4
TCE	50	3.5	TCE	2.9	-0.6	-18
TCE	50	8.7	TCE	7.8	-0.9	-10.3
TCE	50	8.7	TCE	8.1	-0.6	-6.4
TCE	50	8.7	TCE	8.4	-0.3	-3.2
TCE	85	5	TCE	3.8	-1.2	-23.6
TCE	85	5	TCE	3.8	-1.2	-23.6
TCE	85	5	TCE	3.8	-1.2	-24.6
TCE	85	17	TCE	16.2	-0.8	-4.8
TCE	85	17	TCE	16.9	-0.1	-0.9
TCE	85	17	TCE	19.8	2.8	16.6
Tol	20	3.3	Tol	3.4	0.1	1.6
Tol	20	3.3	Tol	3.4	0.1	2.5
Tol	20	3.3	Tol	3.4	0.1	3
Tol	20	11.5	Tol	11.3	-0.2	-2
Tol	20	11.5	Tol	12	0.5	4
Tol	20	11.5	Tol	13	1.5	12.9
Tol	50	2.3	Tol	2.2	-0.1	-2.6
Tol	50	2.3	Tol	2.2	-0.1	-3.4
Tol	50	2.3	Tol	2.3	0	-1
Tol	50	5.8	Tol	5.4	-0.4	-7.1
Tol	50	5.8	Tol	5.5	-0.3	-4.7
Tol	50	5.8	Tol	5.7	-0.1	-1.9
Tol	85	3.3	Tol	3.3	0	0
Tol	85	3.3	Tol	3.3	0	1
Tol	85	3.3	Tol	3.4	0.1	2.9
Tol	85	11.5	Tol	9.9	-1.6	-13.9
Tol	85	11.5	Tol	9.9	-1.6	-14.2
Tol	85	11.5	Tol	10.3	-1.2	-10.1
Xyl	20	2.3	Xyl	2.5	0.2	9.4
Xyl	20	2.3	Xyl	2.5	0.2	10.6
Xyl	20	2.3	Xyl	2.6	0.3	11.8
Xyl	20	7.9	Xyl	9.4	1.5	19.3
Xyl	20	7.9	Xyl	9.6	1.7	21.7
Xyl	20	7.9	Xyl	10.1	2.2	28.3

Xyl	50	1.6	MEK	1.6	0	-1.4
Xyl	50	1.6	MEK	1.8	0.2	9.5
Xyl	50	1.6	MEK	1.8	0.2	9.8
Xyl	50	4	Xyl	3.8	-0.2	-3.9
Xyl	50	4	Xyl	3.8	-0.2	-4
Xyl	50	4	Xyl	3.8	-0.2	-4.3
Xyl	85	2.3	Tol	2.4	0.1	4.4
Xyl	85	2.3	Tol	2.4	0.1	4.5
Xyl	85	2.3	Tol	2.4	0.1	4.6
Xyl	85	7.9	Tol	8.4	0.5	6.3
Xyl	85	7.9	Tol	8.4	0.5	6.4
Xyl	85	7.9	Tol	8.6	0.7	8.6

Appendix C – Individual Validation Results for Organics, Two-Vapor

Vapor 1	Conc (ppm)	Vapor 2	Conc. (ppm)	Est Vapor 1	Est conc 1	Error	%error	Est Vapor 2	Est Conc 2	Error	%error
Ace	17.5			Ace	10	7.5	42.8	IPA	2.8	2.8	---
Ace	17.5			Ace	10.8	6.7	38.1	IPA	2.8	2.8	---
Ace	13.2			Ace	5.8	7.4	55.8	IPA	2.4	2.4	---
Ace	13.2			Ace	5.8	7.4	55.8	IPA	2.4	2.4	---
IPA	8.3			IPA	7.5	0.8	9.6				
IPA	8.3			IPA	7.5	0.8	9.6				
IPA	6.3			IPA	4.4	1.9	30.5				
IPA	6.3			IPA	4.4	1.9	30.5				
MEK	7.2			MEK	4.1	3.1	42.9	Xyl	2.6	2.6	---
MEK	7.2			MEK	4.1	3.1	42.9	Xyl	2.6	2.6	---
MEK	7.2			MEK	4.1	3.1	42.9	Xyl	2.6	2.6	---
MEK	7.2			MEK	4.1	3.1	42.9	Xyl	2.6	2.6	---
MEK	5.4			MEK	2.7	2.7	49.5	Xyl	1.7	1.7	---
MEK	5.4			MEK	2.7	2.7	49.5	Xyl	1.7	1.7	---
Tol	8.7			MEK	3.4	5.3	60.7	Tol	3.3	3.3	---
Tol	8.7			Tol	6.2	2.5	28.6	Xyl	1.4	1.4	---
Tol	6.5			Tol	3.3	3.2	49.5	Xyl	0.6	0.6	---
Tol	6.5			Tol	3.3	3.2	49.5	Xyl	0.6	0.6	---
Xyl	6			Xyl	4	2	33.4	Tol	2.5	2.5	---
Xyl	6			Xyl	4	2	33.4	Tol	2.5	2.5	---
Xyl	4.5			Xyl	2.6	1.9	43.2	Tol	1.6	1.6	---
Xyl	4.5			Xyl	2.3	2.2	49.5	Tol	2.1	2.1	---
Ace	17.5	IPA	8.3	Ace	6.7	10.8	61.9	IPA	7.5	0.8	9.6
Ace	17.5	IPA	8.3	Ace	6.7	10.8	61.9	IPA	7.5	0.8	9.6
Ace	13.2	IPA	6.3	Ace	3.3	9.9	74.7	IPA	6.7	0.5	7.4
Ace	13.2	IPA	6.3	Ace	3.3	9.9	74.7	IPA	7.1	0.9	13.7
MEK	7.2	Tol	8.7	MEK	6.5	0.7	9.7	Xyl	4	4.7	54.1
MEK	7.2	Tol	8.7	MEK	6.5	0.7	9.7	Xyl	4	4.7	54.1
MEK	5.4	Tol	6.5	MEK	5.1	0.3	5.3	Xyl	3.1	3.4	52.1
MEK	5.4	Tol	6.5	MEK	5.1	0.3	5.3	Xyl	3.1	3.4	52.1
MEK	7.2	Xyl	6	MEK	6.5	0.7	9.7	Xyl	5.4	0.6	9.5
MEK	7.2	Xyl	6	MEK	6.5	0.7	9.7	Xyl	5.4	0.6	9.5
MEK	5.4	Xyl	4.5	MEK	4.8	0.6	11.6	Xyl	3.4	1.1	24.2
MEK	5.4	Xyl	4.5	MEK	4.8	0.6	11.6	Xyl	3.4	1.1	24.2
Tol	8.7	Xyl	6	Tol	7.8	0.8	9.6	Xyl	3.7	2.3	38.1
Tol	8.7	Xyl	6	Tol	7.8	0.8	9.6	Xyl	4.3	1.7	28.6
Tol	6.5	Xyl	4.5	Tol	6.6	0.1	1.1	Xyl	2.3	2.2	49.5
Tol	6.5	Xyl	4.5	Tol	7	0.5	7.4	Xyl	2.3	2.2	49.5

Appendix D – MATLAB Code

ConcLoo.M : calculate single-vapor Resub and LOO concentration errors (Tables IV, VI)

```
%use best single sensor, with separate model and validation data sets
Init;
InitGraf;

RESUB=FALSE;      %normal, LOO operation
RESUB=TRUE;       %resub operation

MODEL='Poly';     %1 very bad performance
MODEL='Hyper';    %3 prone to discontinuities w/in concentration range
MODEL='Best';     %  DO NOT USE! no correlation btwn rho,SSE and performance!
MODEL='Tan';      %4 second best
MODEL='Exp';      %2 overall best (for air5o3)

RH=char('Low','Mid','Hi');
CONC=char('Low','Mid','Hi','VHi','SHi','UHi');

%=============================================================================
DIR='\enose\air3f5\';   VERSION=4;                        % 200 ppb max
DIR='\enose\air3f5\';   VERSION=3;                        % 500 ppb max, no 200
DIR='\enose\air3f5\';   VERSION=2;                        % 500 ppb max, w/200
DIR='\enose\air3f5\';   VERSION=1;                        %1000 ppb max, w/200
DIR='\enose\air3f5\';   VERSION=5;                        % 100 ppb max
REMOVE_CLASS=[];                                          %use all classes
BEST_SENSOR=[9 9];                                        %same for all RH

DIR='\enose\air3f6\';   VERSION=1;                        % 500 ppb max, filtered
DIR='\enose\air3f7\';   VERSION=1;                        % 500 ppb max, unfiltered
REMOVE_CLASS=[];                                          %use all classes
BEST_SENSOR=[9 9];                                        %same for all RH

DIR='\enose\air5o3\';   VERSION=1;
REMOVE_CLASS=3;                                           %TCE for air5o3
BEST_SENSOR=[4 4 7; 7 5 5; 5 6 2; 9 6 5; 9 9 5; 8 9 5];   %each RH          WRT VALIDATAION
DATA
BEST_SENSOR=[4 5 8 5 9 5];                                %same for all RH   WRT VALIDATAION
DATA
BEST_SENSOR=[4 9 8 4 9 9];                                %same for all RH   WRT MODEL DATA

%=============================================================================
%allXXX = all the data

[alldata,allclass,allrh,allconc,alltrue,TRUE_CONC]=LoadFile(DIR,VERSION);
% data    class    1-3    1-4  conc values

DIMS=size(alldata,2);
CLASSES=max(allclass);
CONCS=size(TRUE_CONC,2);
CONC_MAX=TRUE_CONC(:,end);       %per class
CONC_MIN=TRUE_CONC(:,1);         %per class

disp(['Model = ',MODEL]);

if (prod(size(BEST_SENSOR)) == 1)
  BEST_SENSOR=BEST_SENSOR * ones(CLASSES,3);      %3=RH's
  end;

if (any(size(BEST_SENSOR) == 1))
  BEST_SENSOR=Vec2Ary(BEST_SENSOR,3)';            %3=RH's
  end;

BEST_SENSOR
%=============================================================================
%remove a class

if (~isempty(REMOVE_CLASS))
```

```
    if (length(REMOVE_CLASS) > 1)
      error('Cannot handle removing more than one class at a time!');
    else
      disp(['Removing class #',num2str(REMOVE_CLASS)]);
      end;

    indx=find(allclass == REMOVE_CLASS);
    alldata(indx,:)=[];
    allclass(indx)=[];
    allrh(indx)=[];
    allconc(indx)=[];
    alltrue(indx)=[];
    TRUE_CONC(REMOVE_CLASS,:)=[];
    for i=REMOVE_CLASS+1 : CLASSES
      allclass(allclass == i)=i-1;
      end;

    CONC_MAX(REMOVE_CLASS)=[];
    CONC_MIN(REMOVE_CLASS)=[];
    BEST_SENSOR(REMOVE_CLASS,:)=[];
    CLASSES=CLASSES - 1;
    end;
%=============================================================================
delta=[];
delclass=[];
errall=zeros(size(alldata,1),1);
perrall=zeros(size(alldata,1),1);
disp(' ');

for targetRH=1 : 3
  subplot(2,2,targetRH);
  hold on;
  CM=max(CONC_MAX);
  plot([0 CM],[0 CM],'k-');
  plot([0 CM],1.1*[0 CM],'r:');
  plot([0 CM],1.2*[0 CM],'b:');
  plot([0 CM],[0 CM]/1.1,'r:');
  plot([0 CM],[0 CM]/1.2,'b:');

  indx=find(allrh == targetRH);
  alldata1=alldata(indx,:);                    %allXXX1 = all the data for 1 RH
  allclass1=allclass(indx);
  allconc1=allconc(indx);
  alltrue1=alltrue(indx);
  EX1=size(alldata1,1);

  perr=zeros(size(alldata1,1),1);
  abserr=zeros(size(alldata1,1),1);

  for ex=1 : EX1
%split into train and test sets
    tdata=alldata1(ex,:);        %tXXX = test data, 1 RH
    tclass=allclass1(ex);
    tconc=allconc1(ex);
    ttrue=alltrue1(ex);
    data=alldata1;
    class=allclass1;
    conc=allconc1;
    true=alltrue1;
    if (~RESUB)
      data(ex,:)=[];               %XXX = train data, 1 RH
      class(ex)=[];
      conc(ex)=[];
      true(ex)=[];
      end;
%alldata,allclass,allconc,alltrue,allrh       = all data (all RH's)
%alldata1,allclass1,allconc1,alltrue1         = all data (for this RH)
%data,class,conc,true                         = model/training data
%tdata,tclass,tconc,ttrue                     = test data (one ex)

%model the training set
```

```
    [P,whichmodel,rhoall]=MakeModel(data,class,true,MODEL);
%     P is {SENSORS,CLASSES} of model co-efficients
%     whichmodel is (SENSORS,CLASSES) of 1=polynomial model, 2=exponential model
%     rhoall is (SENSORS,CLASSES) of curve fit quality

%get concentration estimations (10 per example)
    estconc=TestModel(tdata,tclass,P,whichmodel,CONC_MIN,CONC_MAX);

%reduce them to a single estimate by picking best sensor for this class and RH
    estconc=estconc(BEST_SENSOR(tclass,targetRH));

%display the results
    plot(ttrue,estconc,[colors(tclass) '.']);

  %update error tracking variables
    delta=[delta (estconc-ttrue)];                          %this could be replaced by abserr,
with work
    delclass=[delclass; tclass];                            %not used

    perr(ex)=abs(estconc - ttrue) / ttrue;
    abserr(ex)=abs(estconc - ttrue);
    end; %all ex's

  warning off MATLAB:divideByZero;
  for i=1 : CLASSES
    for j=1 : CONCS
      classconcmeanRH(i,j)=mean(perr((allclass1==i) & (allconc1==j)));
      end;
    end;
  warning on MATLAB:divideByZero;

%save %errors in list w/same order as allclass, allconc, allrh
  indx=find(allrh == targetRH);
  errall(indx)=abserr;
  perrall(indx)=perr;

%display table info (per RH)
  disp('------------------------------------------------');
  disp(' ');
  disp(['Mean |%err| by vapor and concentration, RH=',RH(targetRH,:),' : ']);
  disp(' ');
  str1='Gas |';
  str2='--- |';
  for i=1 : CONCS
    str1=[str1 ' ' CONC(i,:) '  '];
    str2=[str2 ' ---  '];
    end;
  str1=[str1 '|  Avg'];
  str2=[str2 '|  ---'];
  disp(str1);
  disp(str2);
  for i=1 : CLASSES
    temp=classconcmeanRH(i,:);
    temp(isnan(temp))=[];
    disp([' ',num2str(i),'  |',sprintf(' %3d  ',round(100*classconcmeanRH(i,:))),'|
',sprintf('%3d',round(100*mean(temp)))]);
    end;
  temp=classconcmeanRH(:);
  temp(isnan(temp))=[];
  disp(str2);
  disp(['Avg |',sprintf(' %3d  ',round(100*mean(classconcmeanRH))),'|
',sprintf('%3d',round(100*mean(temp)))]);
  disp(' ');

%finish graph
  xlabel('True concentration (ppm/ppb)');
  ylabel('Estimated concentration (ppm/ppb)');
  if (targetRH == 1) title([DIR(8:end-1),' : low RH (exp.model)']); end;
  if (targetRH == 1) legend('Exact','+/- 10%','+/- 20%',2); end;
  if (targetRH == 2) title('Mid RH'); end;
  if (targetRH == 3) title('Hi RH'); end;
```

```
  end; %all RH's

%-------------------------------------------------------------------------------
%display table info (across all RH's)

warning off MATLAB:divideByZero;
for i=1 : CLASSES
  for j=1 : CONCS
    mean_perr(i,j)=mean(perrall((allclass==i) & (allconc==j)));
    end;
  end;
warning on MATLAB:divideByZero;

disp('-------------------------------------------------');
disp(' ');
disp('Mean |%err| across all RH : ');
disp(' ');
str1='Gas |';
str2='--- |';
for i=1 : CONCS
  str1=[str1 ' ' CONC(i,:) '  '];
  str2=[str2 ' ---  '];
  end;
str1=[str1 '|  Avg'];
str2=[str2 '|  ---'];
disp(str1);
disp(str2);
for i=1 : CLASSES
  temp=mean_perr(i,:);
  temp(isnan(temp))=[];
  disp([' ',num2str(i),'  |',sprintf(' %3d  ',round(100*mean_perr(i,:))),'|
',sprintf('%3d',round(100*mean(temp)))]);
  end;
temp=mean_perr(:);
temp(isnan(temp))=[];
disp(str2);
disp(['Avg |',sprintf(' %3d  ',round(100*mean(mean_perr))),'|
',sprintf('%3d',round(100*mean(temp)))]);

disp(' ');
disp('-------------------------------------------------');
disp(' ');
disp('Across vapor, RH, conc : ');
disp(['  Overall mean |%err| = ',sprintf('%3.1f',100*mean(mean(mean_perr)))]);
disp(['  Overall mean |%err| = ',sprintf('%3.1f',100*mean(mean(mean_perr(:,2:end)))),' (excluding
lowest concentration)']);

errlow=errall(allconc == 1);
disp(' ');
disp(['  Lowest concentration : mean estimation error = ',sprintf('%4.2f',mean(errlow)),'
ppb/ppm']);
list=sort(errlow);
i=round(0.95*length(list));
disp(['  Lowest concentration : 95% of the estimation errors are less than
',sprintf('%4.2f',list(i)),' ppb/ppm']);

disp(' ');
disp(['  Fraction of estimates within  5% = ',sprintf('%4.2f',mean(perrall<=0.05)),' (all)  =
',sprintf('%4.2f',mean(perrall(allconc>1)<=0.05)),' (not low)']);
disp(['  Fraction of estimates within 10% = ',sprintf('%4.2f',mean(perrall<=0.10)),' (all)  =
',sprintf('%4.2f',mean(perrall(allconc>1)<=0.10)),' (not low)']);
disp(['  Fraction of estimates within 20% = ',sprintf('%4.2f',mean(perrall<=0.20)),' (all)  =
',sprintf('%4.2f',mean(perrall(allconc>1)<=0.20)),' (not low)']);

disp(' ');
disp('See graphs');
%-----------------------------------
%display last graph : distribution of delta's

subplot(2,2,4);
MAX=5 * ceil(max(abs(delta)) / 5);
```

```
STEP=MAX / 5;
STEP=5 * round(STEP/5);
if (STEP < 1) STEP=1; end;
hist(delta,-MAX:STEP:MAX);
xlabel('Error in estimate (ppm/ppb)');
title('Estimation error');
```

Milestone4.M : calculate organic single-vapor validation results (Table VII)

```
%use best single sensor (exponential model only)
%modify to show results in Tim's format
Init;
InitGraf;

NO_TCE=FALSE;
NO_TCE=TRUE;

BEST_SENSOR=[4 4 4; 5 5 5; 8 8 8; 5 5 5; 9 9 9; 5 5 5];   %same for all RH, WRT VALIDATION DATA
BEST_SENSOR=[4 4 4; 9 9 9; 8 8 8; 4 4 4; 9 9 9; 9 9 9];   %same for all RH, WRT MODEL DATA

%=============================================================================
MODEL='Exp';

VAPOR=char('Ace','IPA','TCE','MEK','Tol','Xyl');
RH=char('20','50','85');
MN=[50 35 20 10];

DIR='\enose\air5o3\';   VERSION=1;
%=============================================================================
%allXXX = all the data

[alldata,allclass,allrh,allconc,alltrue,TRUE_CONC]=LoadFile('\enose\air5o3\',1);
% data    class    1-3    1-4  conc values

DIMS=size(alldata,2);
CLASSES=max(allclass);
CONCS=size(TRUE_CONC,2);
CONC_MAX=TRUE_CONC(:,end);       %per class
CONC_MIN=TRUE_CONC(:,1);         %per class

%=============================================================================

[vdata,vclass,vrh,vconc,vtrue,VTRUE_CONC]=LoadValid;
% data class  1-3  1-4  conc     values

%=============================================================================

[ulist,klist]=DC2UK(alldata,allclass);
for i=1 : size(vdata,1)
  vguess(i)=PickQuad(ulist,klist,vdata(i,:));
  end;
vguess=vguess';

%=============================================================================

disp('Vapor RH  True  ID    Est     Err     %err');
disp('----- --  ----  ---   ----    ----    ----');

for i=1 : CLASSES
  for j=1 : CONCS
    classconcperr{i,j}=[];
    classconcerr{i,j}=[];
    end;
  end;
delta=[];
dclass=[];

for targetRH=1 : 3
  subplot(2,2,targetRH);
  hold on;
```

```
  CM=max(CONC_MAX);
  plot([0 CM],[10 10],'r:');
  plot([0 CM],[20 20],'b:');
  plot([0 CM],[-10 -10],'r:');
  plot([0 CM],[-20 -20],'b:');

  indx=find(allrh == targetRH);
  alldata1=alldata(indx,:);                    %allXXX1 = all the data for 1 RH
  allclass1=allclass(indx);
  allconc1=allconc(indx);
  alltrue1=alltrue(indx);

%model the training set
  [P,polymodel,rhoall]=MakeModel(alldata1,allclass1,alltrue1,MODEL);

  indx=find(vrh == targetRH);
  vdata1=vdata(indx,:);
  vclass1=vclass(indx);
  vconc1=vconc(indx);
  vtrue1=vtrue(indx);
  vguess1=vguess(indx);

%get concentration estimations
  estconc=TestModel(vdata1,vclass1,P,polymodel,CONC_MIN,CONC_MAX);

%reduce them to a single estimate
  cguess1=[];                                              %best sensor for C,RH
  for ex=1 : size(estconc,1)
    cguess1(ex)=estconc(ex,BEST_SENSOR(vclass1(ex),targetRH));
    end;
  estconc=cguess1';

  for ex=1 : size(estconc,1)
    disp([' ',VAPOR(vclass1(ex),:),'  ',RH(targetRH,:),'% ',...
          sprintf('%4.1f',vtrue1(ex)),'   ',...
          VAPOR(vguess1(ex),:),'   ',...
          sprintf('%4.1f',estconc(ex)),'   ',sprintf('%5.1f',estconc(ex)-vtrue1(ex)),'   ',...
          sprintf('%5.1f',100*(estconc(ex)-vtrue1(ex))/vtrue1(ex))]);
    end;
```

```
  warning off MATLAB:divideByZero;
  delta=[delta (estconc-vtrue1)'];
  dclass=[dclass; vclass1];
  perr=abs(estconc-vtrue1) ./ vtrue1;
  err=abs(estconc-vtrue1);
  for i=1 : CLASSES
    for j=1 : CONCS
      classconcperr{i,j}=[classconcperr{i,j} perr((vclass1==i) & (vconc1==j))];
      classconcerr{i,j}=[classconcerr{i,j} err((vclass1==i) & (vconc1==j))];
      end;
    end;
  warning on MATLAB:divideByZero;

  if (targetRH == 1) classconcperr1=classconcperr; end;
  if (targetRH == 2) classconcperr2=classconcperr; end;
  if (targetRH == 3) classconcperr3=classconcperr; end;
  end; %all RH's

%-------------------------------------------------------------------------------

disp(' ');
disp('Vapor Conc %success Error');
total=0;
count=0;
for i=1 : CLASSES
  for j=1 : CONCS
    temp=classconcperr{i,j};
    temp2=classconcerr{i,j};
    vg=vguess((vclass==i) & (vconc==j));
    vc=vclass((vclass==i) & (vconc==j));
    ps=round(100*mean(vg == vc));
```

```
    if (j == 1)
      disp([' ',VAPOR(i,:),'   ',sprintf('%3.1f',TRUE_CONC(i,j)),'      ',sprintf('%3d',ps),'
',sprintf('%4.1f',mean(temp2(:))),' ppm']);
    else
      disp([' ',VAPOR(i,:),'   ',sprintf('%3.1f',TRUE_CONC(i,j)),'      ',sprintf('%3d',ps),'
',sprintf('%4.1f',100*mean(temp(:))),'%']);
      if (i ~= 3)
        total=total + 100*mean(temp(:));
        count=count + 1;
        end;
      end;
    end;
  end;

mean_perr=total / count
```

MixModel6.M : calculate two-vapor organic estimation results (Table VIII)

```
%run one- and two-vapor examples thru the models (resub)
%run all 4 sets of test pairs at once
%unlike earlier programs, this has no class=0 values!
Init;
InitGraf;

NORM=FALSE;
NORM=TRUE;

DELETE=[1 3 5];
DELETE=[];
DELETE=[1 3 4 5 10];
DELETE=[1 3 4 10];
DELETE=[1 3 5 10];     %good
DELETE=[1 2 3 5 10];   %slightly better

STEPS=50;
STEPS=20;

PAIRS=[1 2; 4 5; 4 6; 5 6];

%=============================================================================
DIR='\enose\air5o3\';   VERSION=1;
%=============================================================================
%allXXX = all the data

[alldata,allclass,allrh,allconc,alltrue,TRUE_CONC]=LoadFile(DIR,VERSION,0);     %1 --> raw data
% data    class    1-3    1-4  conc values

DIMS=size(alldata,2);
CLASSES=max(allclass)
CONCS=size(TRUE_CONC,2)
CONC_MIN=TRUE_CONC(:,1);          %for each class
CONC_MAX=TRUE_CONC(:,end);        %for each class

[ROWS,COLS]=BestRC(4);
USE=setdiff(1:DIMS,DELETE);

%=============================================================================

[vdata,vclass,vrh,vconc,vtrue,VTRUE_CONC]=LoadAir5o3D(0);                    %1-->return raw
data

%vclass, vconc, vtrue all have 2 columns : [vapor#1 vapor#2]

%=============================================================================
STEPS
DELETE
NORM

classtrue=[];
conctrue=[];
```

```
classguesses=[];
concguesses=[];

tic;
for targetRH=2 : 2
  disp(' ');
  disp(['RH=',num2str(targetRH)]);

  load(['mix4rh',num2str(targetRH),'.mat']);     %P, whichmodel, rhoall, maxr0

  classestried=0;
  classesright=0;
  totalerr=0;
  meanex=0;

for pair=1 : 4
  TESTPAIRS=PAIRS(pair,:);

  indx=find(vrh == targetRH);                    %vXXX1 = all diagonal data for this RH
  vdata1=vdata(indx,:);
  vclass1=vclass(indx,:);
  vtrue1=vtrue(indx,:);

  indx=find(allrh == targetRH);                  %allXXX1 = all the data for 1 RH
  alldata1=alldata(indx,:);
  allclass1=allclass(indx,:);
  alltrue1=alltrue(indx,:);

  indx=find(allclass1 == TESTPAIRS(1));          %XXX1a = 1 RH, first vapor of pair
  data1a=alldata1(indx,:);
  class1a=allclass1(indx);
  true1a=alltrue1(indx);

  indx=find(allclass1 == TESTPAIRS(2));          %XXX1b = 1 RH, second vapor of pair

  data1b=alldata1(indx,:);
  class1b=allclass1(indx);
  true1b=alltrue1(indx);

  indx=find((vclass1(:,1) == TESTPAIRS(1)) & (vclass1(:,2) == TESTPAIRS(2)));
  dataP=vdata1(indx,:);                          %XXXP = 1 RH, diagonal data for this pair
  classP=vclass1(indx,:);
  trueP=vtrue1(indx,:);

  data0=[data1a; data1b; dataP];
  class0=[[class1a TESTPAIRS(2)*ones(length(class1a),1)];...
          [TESTPAIRS(1)*ones(length(class1b),1) class1b];...
          classP];
  true0=[[true1a zeros(length(true1a),1)];...
         [zeros(length(true1b),1) true1b];...
         trueP];

  dataP=data0;                                   %make test examples ALL examples for this pair
  classP=class0;
  trueP=true0;

  dataP(:,DELETE)=[];

  [EX,DIMS]=size(dataP);
%-------------------------------------------------------------------------------
  bothright=0;
  oneright=0;
  bothwrong=0;
  perr=[];

  for ex=1 : EX
    mixmeas=dataP(ex,:);
    class1=classP(ex,1);
    class2=classP(ex,2);
    true1=trueP(ex,1);
    true2=trueP(ex,2);
```

```
%mixmeas       = response from mixture of gasses
%class1,class2 = class #'s of gasses
%true1,true2   = true conc of gases (ppm)

%-evaluate on a grid,pick smallest value-------------------------------------

    for p=1 : size(PAIRS,1)
      x=linspace(0,1.2*CONC_MAX(PAIRS(p,1)),STEPS);
      y=linspace(0,1.2*CONC_MAX(PAIRS(p,2)),STEPS);
      [X,Y]=meshgrid(x,y);

      combosum=zeros(size(X));
      for k0=1 : DIMS
        k=USE(k0);
        mixmeas(k0)=whichmodel(k,p) * mixmeas(k0);

        coeff=P{k,p};
        model=coeff(6) + (coeff(1)*(X.^coeff(2)) + coeff(3)*(Y.^coeff(4))) .^ coeff(5);

        if (NORM)
          combosum=combosum + abs(model - mixmeas(k0)) / max(max(model)); %ERROR FUNCTION AND
COMBINATION FUNCTION
if (class1 == 4) & (class2 == 6)
  combosum1{k0}=abs(model - mixmeas(k0)) / max(max(model));
  end;
        else
          combosum=combosum + abs(model - mixmeas(k0));                  %ERROR FUNCTION AND
COMBINATION FUNCTION
          end;
        end; %all sensors

if (class1 == 4) & (class2 == 6)
  combosum0=combosum;
  end;

      minval=min(min(combosum));
      [r,c]=find(combosum == minval);                  %FIND MIN ERROR FUNCTION
      if (length(r) > 1)
        r'
        num_mins=length(r)
        error('Method #1 : more than one minimum!');
        end;
      cest1=x(c);
      cest2=y(r);

      concguess(p,:)=[cest1 cest2];
      quality(p)=minval;
      end; %all pairs

%-pick something--------------------------------------------------------------

    indx=find(quality == min(quality));
    if (length(indx) > 1) error('Multiple bests!'); end;

    clguess1=PAIRS(indx,1);
    coguess1=concguess(indx,1);
    clguess2=PAIRS(indx,2);
    coguess2=concguess(indx,2);

    if (coguess1 == 0) & (clguess2 == class2) clguess1=class1; end;
    if (coguess1 == 0) & (clguess2 == class1) clguess1=class2; end;
    if (coguess2 == 0) & (clguess1 == class1) clguess2=class2; end;
    if (coguess2 == 0) & (clguess1 == class2) clguess2=class1; end;

    if (clguess1 > clguess2)
%     error('Classes reversed #2!');
      temp=clguess1;
      clguess1=clguess2;
      clguess2=temp;
      temp=coguess1;
      coguess1=coguess2;
```

```
      coguess2=temp;
      end;

if (class1 == 4) & (class2 == 6)
  [class1 class2 true1 true2 mixmeas]
  end;
%if (class1 == 4) & (class2 == 6) & (true1 > 0) & (true2 > 0)
if (class1 == 4) & (class2 == 6) & (true1 < 4) & (true2 < 4) & (coguess1 > 4) & (coguess2 > 4) &
(0)
  mixmeas
  [class1 class2 true1 true2 coguess1 coguess2 (coguess1-true1)/true1 (coguess2-true2)/true2]
  if (1)
    InitGraf;
    for k0=1 : DIMS
      subplot(2,3,k0);
      surf(X,Y,combosum1{k0});
      axis([0 20 0 10 0 .8]);
      end;
    subplot(2,3,6);
    surf(X,Y,combosum0);
    return;
    end;
  end;

    if (clguess1 == class1) & (clguess2 == class2)
      bothright=bothright + 1;
      classesright=classesright + 2;
      if (true1 > 0)
        perr=[perr (coguess1-true1)/true1];
        end;
      if (true2 > 0)
        perr=[perr (coguess2-true2)/true2];
        end;
    elseif ((clguess1 == class1) | (clguess2 == class2) | (clguess1 == class2) | (clguess2 ==
class1))
      oneright=oneright + 1;
      classesright=classesright + 1;

      if (clguess1 == class1)
        if (true1 > 0)
          perr=[perr (coguess1-true1)/true1];
          end;
      elseif (clguess1 == class2)
        if (true2 > 0)
          perr=[perr (coguess1-true2)/true2];
          end;
        end;

      if (clguess2 == class2)
        if (true2 > 0)
          perr=[perr (coguess2-true2)/true2];
          end;
      elseif (clguess2 == class1)
        if (true1 > 0)
          perr=[perr (coguess2-true1)/true1];
          end;
        end;
    else
      bothwrong=bothwrong + 1;
      end;
    classestried=classestried + 2;

    classtrue=[classtrue; [class1 class2]];
    conctrue=[conctrue; [true1 true2]];
    classguesses=[classguesses; [clguess1 clguess2]];
    concguesses=[concguesses; [coguess1 coguess2]];
    end; %all validation examples

  disp(' ');
  disp('Both    One     None   |%error|');
  disp([sprintf('%3d',round(100*bothright/EX)),'%    ',...
```

```
        sprintf('%3d',round(100*oneright/EX)),'%    ',...
        sprintf('%3d',round(100*bothwrong/EX)),'%    ',...
        sprintf('%3d',round(100*mean(abs(perr)))),'%']);

  totalerr=totalerr + mean(abs(perr));
  meanex=meanex + bothright/EX;

  end; %all pairs

% disp(['Classification = ',num2str(round(100*classesright/classestried)),...
%       '%    |error|=',num2str(round(100*totalerr/4)),'%']);
  disp(['Classification = ',num2str(round(100*meanex/4)),...
        '%    |error|=',num2str(round(100*totalerr/4)),'%']);
  end; %all RH's
```

MixReport3.M : calculate two-vapor organic estimation results (Table VIII)

```
%display final report info
%run mixmodel6 first!

NAMES=char('Ace','IPA','TCE','MEK','Tol','Xyl');

warning off MATLAB:divideByZero;

CATS=size(concguesses,1)/4;  %number of pair/conc categories

for i=1 : CATS
  indx=4*(i-1)+1 : 4*i;

  classPi=classtrue(indx,:);
  truePi=conctrue(indx,:);
  guess=classguesses(indx,:);
  est=concguesses(indx,:);

  flag2=(classPi == guess);
  flag=flag2(:,1) & flag2(:,2);
  success2=mean(flag);

  perr=[];
  err=[];
  for j=1 : length(flag)
    if (flag(j))
      perr=[perr; abs(truePi(j,:)-est(j,:)) ./ truePi(j,:)];
      err=[err; abs(truePi(j,:)-est(j,:))];
      end;
    end;
  perr=perr(:)';
  perr(isnan(perr))=[];
  err=err(:)';

  disp([NAMES(classPi(1,1),:),'@',sprintf('%4.1f',truePi(1,1)),'ppm + ',...
        NAMES(classPi(1,2),:),'@',sprintf('%4.1f',truePi(1,2)),'ppm : ',...
        sprintf('%3d',100*success2),'%   ',sprintf('%4.1f',100*mean(perr)),'%']);

ps(i)=success2;
pe(i)=mean(perr);
er(i)=mean(err);
  end;

ps_all=mean(ps)
perr_all=mean(pe)
err_all=mean(er)

minconc=(mod((0:CATS),4) == 0);
minconc(21)=[];

pe1=pe(~minconc);
perr_big=mean(pe1)

er1=er(minconc);
```

```
err_sml=mean(er1)

warning on MATLAB:divideByZero;
```

MixModel7.M : calculate two-vapor organic validation results (Table IX)

```
%run the last set of validation data (singles and diagonals) thru the model
%run all 4 sets of test pairs at once
%unlike earlier programs, this has no class=0 values!
Init;
InitGraf;

NORM=FALSE;
NORM=TRUE;

DELETE=[1 3 5];
DELETE=[];
DELETE=[1 3 4 5 10];
DELETE=[1 3 4 10];
DELETE=[1 3 5 10];    %good
DELETE=[1 2 3 5 10];  %slightly better

SHOWEACH=FALSE;
SHOWEACH=TRUE;

STEPS=50;
STEPS=20;

PAIRS=[1 2; 4 5; 4 6; 5 6];

%=============================================================================
VAPOR=char('Ace','IPA','TCE','MEK','Tol','Xyl');
RH=char('20','50','85');

DIR='\enose\air5mixV\';   VERSION=1;
%=============================================================================
%allXXX = all the data

[alldata,allclass,allrh,allconc,alltrue,TRUE_CONC]=LoadFile(DIR,VERSION,0);    %1 --> raw data
% data    class    1-3    1-4  conc values

DIMS=size(alldata,2);
CLASSES=max(allclass)
CONCS=size(TRUE_CONC,2)
CONC_MIN=TRUE_CONC(:,1);          %for each class
CONC_MAX=TRUE_CONC(:,end);        %for each class

[ROWS,COLS]=BestRC(4);
USE=setdiff(1:DIMS,DELETE);

%=============================================================================

[vdata,vclass,vrh,vconc,vtrue,VTRUE_CONC]=LoadAir5o3mixV(0);                    %1-->return raw
data

%vclass, vconc, vtrue all have 2 columns : [vapor#1 vapor#2]

%=============================================================================
STEPS
DELETE
NORM

classtrue=[];
conctrue=[];
classguesses=[];
concguesses=[];

tic;
for targetRH=2 : 2
  disp(' ');
```

```
    disp(['RH=',num2str(targetRH)]);

    load(['mix4rh',num2str(targetRH),'.mat']);      %P, whichmodel, rhoall, maxr0

    classestried=0;
    classesright=0;
    totalerr=0;
    meanex=0;

  for pair=1 : 4
    TESTPAIRS=PAIRS(pair,:);

    indx=find(vrh == targetRH);                     %vXXX1 = all diagonal data for this RH
    vdata1=vdata(indx,:);
    vclass1=vclass(indx,:);
    vtrue1=vtrue(indx,:);

    indx=find(allrh == targetRH);                   %allXXX1 = all the data for 1 RH
    alldata1=alldata(indx,:);
    allclass1=allclass(indx,:);
    alltrue1=alltrue(indx,:);

    indx=find(allclass1 == TESTPAIRS(1));           %XXX1a = 1 RH, first vapor of pair
    data1a=alldata1(indx,:);
    class1a=allclass1(indx);
    true1a=alltrue1(indx);

    indx=find(allclass1 == TESTPAIRS(2));           %XXX1b = 1 RH, second vapor of pair
    data1b=alldata1(indx,:);
    class1b=allclass1(indx);
    true1b=alltrue1(indx);

    indx=find((vclass1(:,1) == TESTPAIRS(1)) & (vclass1(:,2) == TESTPAIRS(2)));
    dataP=vdata1(indx,:);                           %XXXP = 1 RH, diagonal data for this pair
    classP=vclass1(indx,:);
    trueP=vtrue1(indx,:);

    data0=[data1a; data1b; dataP];
    class0=[[class1a TESTPAIRS(2)*ones(length(class1a),1)];...
            [TESTPAIRS(1)*ones(length(class1b),1) class1b];...
            classP];
    true0=[[true1a zeros(length(true1a),1)];...
           [zeros(length(true1b),1) true1b];...
           trueP];

    dataP=data0;                                    %make test examples ALL examples for this pair
    classP=class0;
    trueP=true0;

    dataP(:,DELETE)=[];

    [EX,DIMS]=size(dataP);
%-------------------------------------------------------------------------------
    bothright=0;
    oneright=0;
    bothwrong=0;
    perr=[];

    for ex=1 : EX
      mixmeas=dataP(ex,:);
      class1=classP(ex,1);
      class2=classP(ex,2);
      true1=trueP(ex,1);
      true2=trueP(ex,2);
%mixmeas       = response from mixture of gasses
%class1,class2 = class #'s of gasses
%true1,true2   = true conc of gases (ppm)

%-evaluate on a grid,pick smallest value-------------------------------------

      for p=1 : size(PAIRS,1)
```

```
        x=linspace(0,1.2*CONC_MAX(PAIRS(p,1)),STEPS);
        y=linspace(0,1.2*CONC_MAX(PAIRS(p,2)),STEPS);
        [X,Y]=meshgrid(x,y);

        combosum=zeros(size(X));
        for k0=1 : DIMS
          k=USE(k0);
          mixmeas(k0)=whichmodel(k,p) * mixmeas(k0);

          coeff=P{k,p};
          model=coeff(6) + (coeff(1)*(X.^coeff(2)) + coeff(3)*(Y.^coeff(4))) .^ coeff(5);

          if (NORM)
            combosum=combosum + abs(model - mixmeas(k0)) / max(max(model)); %ERROR FUNCTION AND
COMBINATION FUNCTION
          else
            combosum=combosum + abs(model - mixmeas(k0));    %ERROR FUNCTION AND COMBINATION
FUNCTION
            end;
          end; %all sensors

        minval=min(min(combosum));
        [r,c]=find(combosum == minval);                    %FIND MIN ERROR FUNCTION
        if (length(r) > 1)
          r'
          num_mins=length(r)
          error('Method #1 : more than one minimum!');
          end;
        cest1=x(c);
        cest2=y(r);

        concguess(p,:)=[cest1 cest2];
        quality(p)=minval;
        end; %all pairs

%-pick something-------------------------------------------------------------

      indx=find(quality == min(quality));
      if (length(indx) > 1) error('Multiple bests!'); end;

      clguess1=PAIRS(indx,1);
      coguess1=concguess(indx,1);
      clguess2=PAIRS(indx,2);
      coguess2=concguess(indx,2);

%if (class1 == 4) & (true2 == 0)
%  true=[class1 class2 true1 true2]
%  guess=[clguess1 clguess2 coguess1 coguess2]
%  end;

      if (coguess1 == 0) & (clguess2 == class2) clguess1=class1; end;
      if (coguess1 == 0) & (clguess2 == class1) clguess1=class2; end;
      if (coguess2 == 0) & (clguess1 == class1) clguess2=class2; end;
      if (coguess2 == 0) & (clguess1 == class2) clguess2=class1; end;

      if (clguess1 > clguess2)
%       error('Classes reversed #2!');
        temp=clguess1;
        clguess1=clguess2;
        clguess2=temp;
        temp=coguess1;
        coguess1=coguess2;
        coguess2=temp;
        end;

      if (SHOWEACH)
        disp(['True class=[',num2str([class1 class2]),']   true conc=[',sprintf('%4.1f ',[true1
true2]),...
              '] --> class=[',num2str([clguess1 clguess2]),']   conc=[',sprintf('%4.1f ',[coguess1
coguess2]),']']);
```

```
%disp([' ',VAPOR(class1(ex),:),'  ',RH(targetRH,:),'% ',...
%      sprintf('%4.1f',vtrue1(ex)),'   ',...
%      VAPOR(vguess1(ex),:),'   ',...
%      sprintf('%4.1f',estconc(ex)),'   ',sprintf('%5.1f',estconc(ex)-vtrue1(ex)),'   ',...
%      sprintf('%5.1f',100*(estconc(ex)-vtrue1(ex))/vtrue1(ex))]);
      end;

    if (clguess1 == class1) & (clguess2 == class2)
      bothright=bothright + 1;
      classesright=classesright + 2;
      if (true1 > 0)
        perr=[perr (coguess1-true1)/true1];
        end;
      if (true2 > 0)
        perr=[perr (coguess2-true2)/true2];
        end;
    elseif ((clguess1 == class1) | (clguess2 == class2) | (clguess1 == class2) | (clguess2 ==
class1))
      oneright=oneright + 1;
      classesright=classesright + 1;
      if (clguess1 == class1)
        if (true1 > 0)
          perr=[perr (coguess1-true1)/true1];
          end;

      elseif (clguess1 == class2)
        if (true2 > 0)
          perr=[perr (coguess1-true2)/true2];
          end;
        end;

      if (clguess2 == class2)
        if (true2 > 0)
          perr=[perr (coguess2-true2)/true2];
          end;
      elseif (clguess2 == class1)
        if (true1 > 0)
          perr=[perr (coguess2-true1)/true1];
          end;
        end;
    else
      bothwrong=bothwrong + 1;
      end;
    classestried=classestried + 2;

    classtrue=[classtrue; [class1 class2]];
    conctrue=[conctrue; [true1 true2]];
    classguesses=[classguesses; [clguess1 clguess2]];
    concguesses=[concguesses; [coguess1 coguess2]];
    end; %all validation examples

  disp(' ');
  disp('Both    One     None   |%error|');
  disp([sprintf('%3d',round(100*bothright/EX)),'%    ',...
        sprintf('%3d',round(100*oneright/EX)),'%    ',...
        sprintf('%3d',round(100*bothwrong/EX)),'%    ',...
        sprintf('%3d',round(100*mean(abs(perr)))),'%']);

  totalerr=totalerr + mean(abs(perr));
  meanex=meanex + bothright/EX;

  end; %all pairs

% disp(['Classification = ',num2str(round(100*classesright/classestried)),...
%       '%    |error|=',num2str(round(100*totalerr/4)),'%']);
  disp(['Classification = ',num2str(round(100*meanex/4)),...
        '%    |error|=',num2str(round(100*totalerr/4)),'%']);
  end; %all RH's
```

MixReport4.M : calculate two-vapor organic estimation results (Table IX)

```
%display final report info
%run mixmodel7 first!

NAMES=char('Ace','IPA','TCE','MEK','Tol','Xyl','   ');

%     ACE+IPA                                    MEK+Tol
INDX=[ 3  4;  1  2;  7  8;  5  6; 11 12;  9 10; 17 18; 13 16; 21 22; 19 20; 25 26; 23 24;...
      31 32; 27 30; 35 36; 33 34; 39 40; 37 38; 43 44; 41 42; 47 48; 45 46; 51 52; 49 50];
%     MEK+Xyl                                    Tol+Xyl

warning off MATLAB:divideByZero;

EXALL=size(classtrue,1);
CATS=size(INDX,1);  %number of pair/conc categories

perrall=[];
errall=[];
bothall=0;
oneall=0;
noneall=0;

for i=1 : CATS
  indx=INDX(i,1) : INDX(i,2);
  cltrue=classtrue(indx,:);
  if (any(cltrue(1,:) ~= mean(cltrue))) error('INDX wrong!'); end;
  cotrue=conctrue(indx,:);
  if (any(cotrue(1,:) ~= mean(cotrue))) error('INDX wrong!'); end;
  clguess=classguesses(indx,:);
  coguess=concguesses(indx,:);

  EX=size(cotrue,1);

  perr=[];
  err=[];
  both=0;
  one=0;
  none=0;
  for j=1 : EX
    co=0;
    if (cotrue(j,1) >  0) & (cotrue(j,2) >  0) & (coguess(j,1) >  0) & (coguess(j,2) >  0) co=1;
end;
    if (cotrue(j,1) >  0) & (cotrue(j,2) == 0) & (coguess(j,1) >  0) & (coguess(j,2) == 0) co=2;
end;
    if (cotrue(j,1) >  0) & (cotrue(j,2) == 0) & (coguess(j,1) >  0) & (coguess(j,2) >  0) co=3;
end;
    if (cotrue(j,1) == 0) & (cotrue(j,2) >  0) & (coguess(j,1) == 0) & (coguess(j,2) >  0) co=4;
end;
    if (cotrue(j,1) == 0) & (cotrue(j,2) >  0) & (coguess(j,1) >  0) & (coguess(j,2) >  0) co=5;
end;
    if (cotrue(j,1) >  0) & (cotrue(j,2) >  0) & (coguess(j,1) >  0) & (coguess(j,2) == 0) co=6;
end;
    if (cotrue(j,1) >  0) & (cotrue(j,2) >  0) & (coguess(j,1) == 0) & (coguess(j,2) >  0) co=7;
end;
    if (cotrue(j,1) >  0) & (cotrue(j,2) == 0) & (coguess(j,1) == 0) & (coguess(j,2) >  0) co=8;
end;
    if (cotrue(j,1) == 0) & (cotrue(j,2) >  0) & (coguess(j,1) >  0) & (coguess(j,2) == 0) co=9;
end;
    if (co == 0) error('co==0!'); end;

    cl=0;
    if (cltrue(j,1) == clguess(j,1)) & (cltrue(j,2) == clguess(j,2)) cl=1; end;
    if (cltrue(j,1) ~= clguess(j,1)) & (cltrue(j,2) ~= clguess(j,2)) cl=2; end;
    if (cltrue(j,1) == clguess(j,2)) & (cltrue(j,2) ~= clguess(j,1)) cl=3; end;
    if (cltrue(j,1) ~= clguess(j,2)) & (cltrue(j,2) == clguess(j,1)) cl=4; end;
    if (cltrue(j,1) == clguess(j,1)) & (cltrue(j,2) ~= clguess(j,2)) cl=5; end;
    if (cltrue(j,1) ~= clguess(j,1)) & (cltrue(j,2) == clguess(j,2)) cl=6; end;
    if (cl == 0) error('cl==0!'); end;

    done=0;
```

```
 if ((co == 1) & (cl == 1))
   both=both + 1;
   perr=[perr abs(cotrue(j,1)-coguess(j,1)) ./ cotrue(j,1)];
   perr=[perr abs(cotrue(j,2)-coguess(j,2)) ./ cotrue(j,2)];
   err=[err abs(cotrue(j,1)-coguess(j,1))];
   err=[err abs(cotrue(j,2)-coguess(j,2))];
   done=1;
   end;

 if ((co == 1) & (cl == 5))
   one=one + 1;
   perr=[perr abs(cotrue(j,1)-coguess(j,1)) ./ cotrue(j,1)];
   err=[err abs(cotrue(j,1)-coguess(j,1))];
   done=1;
   end;


 if ((co == 3) & (cl == 1)) | ((co == 3) & (cl == 5))
   one=one + 1;
   perr=[perr abs(cotrue(j,1)-coguess(j,1)) ./ cotrue(j,1)];
   err=[err abs(cotrue(j,1)-coguess(j,1))];
   done=1;
   end;

 if ((co == 3) & (cl == 3))
   one=one + 1;
   perr=[perr abs(cotrue(j,1)-coguess(j,2)) ./ cotrue(j,1)];
   err=[err abs(cotrue(j,1)-coguess(j,2))];
   done=1;
   end;

 if ((co == 4) & (cl == 1))
   both=both + 1;
   perr=[perr abs(cotrue(j,2)-coguess(j,2)) ./ cotrue(j,2)];
   perr=[perr 0];
   err=[err abs(cotrue(j,2)-coguess(j,2))];
   err=[err 0];
   done=1;
   end;

 if ((co == 5) & (cl == 1)) | ((co == 5) & (cl == 6))
   one=one + 1;
   perr=[perr abs(cotrue(j,2)-coguess(j,2)) ./ cotrue(j,2)];
   err=[err abs(cotrue(j,2)-coguess(j,2))];
   done=1;
   end;

 if ((co == 5) & (cl == 4))
   one=one + 1;
   perr=[perr abs(cotrue(j,2)-coguess(j,1)) ./ cotrue(j,2)];
   err=[err abs(cotrue(j,2)-coguess(j,1))];
   done=1;
   end;

 if (~done) co, cl, error('Unknown combo of co and cl!'); end;
 end; %all ex

disp([NAMES(cltrue(1,1),:),'@',sprintf('%4.1f',cotrue(1,1)),'ppm + ',...
      NAMES(cltrue(1,2),:),'@',sprintf('%4.1f',cotrue(1,2)),'ppm : ',...
      sprintf('%3d',100*both/EX),'/',sprintf('%3d',100*one/EX),'/',sprintf('%3d',100*none/EX),'
',...
      sprintf('%5.1f',100*mean(err)),'   ',...
      sprintf('%4.1f',100*mean(perr)),'%']);

perrall=[perrall perr];
errall=[errall err];
bothall=bothall + both;
oneall=oneall + one;
noneall=noneall + none;
end;
```

```
ps_both=bothall / EXALL
ps_one=oneall / EXALL
ps_none=noneall / EXALL
perr_all=mean(perrall)
err_all=mean(errall)

return;

minconc=(mod((0:CATS),4) == 0);
minconc(21)=[];

pe1=pe(~minconc);
perr_big=mean(pe1)

er1=er(minconc);
err_sml=mean(er1)

warning on MATLAB:divideByZero;
```