11-20-2008

An Analysis of Fourier Transform Infrared Spectroscopy Data to Predict Herpes Simplex Virus 1 Infection

Patrick D. Champion

AN ANALYSIS OF FOURIER TRANSFORM INFRARED SPECTROSCOPY DATA TO PREDICT HERPES SIMPLEX VIRUS 1 INFECTION

by

PATRICK D CHAMPION

Under the Direction of Dr. Yu-Sheng Hsu

ABSTRACT

The purpose of this analysis is to evaluate the usefulness of Fourier Transform Infrared (FTIR) spectroscopy in the detection of Herpes Simplex Virus 1 (hsv1) infection at an early stage. The raw absorption values were standardized to eliminate inter-sampling error. Wilcoxon-Mann-Whitney (WMW) statistic's Z score was calculated to select significant spectral regions. Partial least squares modeling was performed because of multicollinearity. Kolmogorov-Smirnov statistic showed models for healthy tissues from different time groups were not from same distribution. The additional 24 hour dataset was evaluated using the following methods. Variables were selected by WMW Z score. Difference of Composites statistic, D_C, was created as a disease indicator and evaluated using area under the ROC curve, specificities, and confidence intervals using bootstrap algorithm. The specificity of D_C was high, however the confidence intervals were large. Future studies are required with larger sample sizes to test this statistic's usefulness.

INDEX WORDS: FTIR, Collinearity, Disease predictor, Wilcoxon-Mann-Whitney, Partial least squares, Kolmogorov-Smirnov, ROC Curve, Specificity, Bootstrap

AN ANALYSIS OF FOURIER TRANSFORM INFRARED SPECTROSCOPY DATA TO PREDICT HERPES SIMPLEX VIRUS 1 INFECTION

by

PATRICK D CHAMPION

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

in the College of Arts and Sciences

Georgia State University

2008

AN ANALYSIS OF FOURIER TRANSFORM INFRARED SPECTROSCOPY DATA TO PREDICT HERPES SIMPLEX VIRUS 1 INFECTION

by

PATRICK D CHAMPION

Committee Chair: Dr. Yu-Sheng Hsu

Committee: Dr. Gary Hastings
Dr. Jun Han

Electronic Version Approved:

Office of Graduate Studies
College of Arts and Sciences
Georgia State University
December 2008

DEDICATION

I would like to dedicate this thesis to my wife Youn-Kyung, who encouraged me, prodded me when needed, and believed in me when I didn't think I could go on in the pursuit of my masters. Thank you so much. I would also like to dedicate this paper to Jesus Christ, without whom I would likely not be alive. He has saved me and blessed me in so many ways and has shown His hand at numerous time in my life. I am forever joyfully indebted to Him.

ACKNOWLEDGEMENTS

I would like to acknowledge with deep gratitude Dr. Yu-Sheng Hsu who graciously guided me through my thesis. I have been deeply impressed by how much he cares for all of his students' success. I am grateful to him for providing the experience of working with real world data which can be far more challenging to analyze than what is seen in text books. I am also so thankful for his patience with my split commitments to school and a workplace that very often keeps me at work late into the evenings.

I would also like to acknowledge both Dr. Gary Hastings and Dr. Jun Han who graciously accepted the offer to be on my thesis review committee. Thank you for your assistance. I would also like to express a special thanks to Dr. Gary Hastings who provided me with such intriguing data and who graciously answered my questions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Distinguishing between healthy and diseased tissue is very important for the purpose of making a diagnosis of many diseases. A spectroscopic tool, called LightTouch™ made by Spectrx, Inc, has been shown useful in the detection of cervical cancer by measuring the reflectance and fluorescence spectrum of cervical tissue using the visible spectrum between 410nm and 710nm (Wang 2007).

Fourier Transform Infrared spectroscopy (FTIR) is another promising technology. It has been used by chemical engineers to perform polymer forensics and by the EPA to measure 100 out of the 189 pollutants on the governments Hazardous Airborne Pollutants list. FTIR devices have been commercially available since the 1980's. They can be as small as two reams of paper or as large as a desktop. FTIR works on many organic and inorganic substances and can measure solid, liquid and gaseous substances. FTIR works by splitting a HeNe laser beam in two. One beam then bounces off of a stationary mirror and the other off of a rapidly moving mirror. The beams are recombined, forming a time based interference pattern that is then directed at a specimen. The resulting reflected or transmitted infrared light is then measured and a Fourier transform is performed to create a absorption spectrum.

This study used a Fourier Transform Infrared (FTIR) spectroscopic device, which measured absorption in the infrared spectrum from 800 cm^{-1} to 1500 cm^{-1}. Please note that the measurement, 800 cm^{-1} means 1/800 cm = 12.5 μm. This places the studied frequencies in the mid-infrared frequency region.

In this study, the device was used to scan cellular suspensions of Vero cells to obtain infrared absorption data that was used to test for the presence or absence of Vero cells infected

with Herpes Simplex Virus 1 (hsv1). Vero cells are cells derived from the epithelial kidney cells of the African green monkey. These cells can be grown in culture banks indefinitely without deterioration in the cell's characteristics. They have been in continuous culture since 1962 and used for research into the development of vaccines against many diseases because of both their stability and sensitivity to many human diseases (Sheets 2000). The purpose of this thesis is to identify an indicator statistic to predict disease status between hsv1-infected and healthy (mock) Vero cell cultures using the data produced by FTIR measurements of the sampled cultures.

Analyzing the data produced by the spectroscopic measurements presented several challenges. One major issue was that the variables produced were highly correlated. In the FTIR spectroscopic measurements, an absorption value was recorded for roughly each integer increment in wavelength, ranging from 800 cm^{-1} to 1500 cm^{-1}. The peaks in absorption, however, were not isolated to single frequencies–instead, the absorption data consisted of peaks and valleys extending for large ranges of frequencies. As a result, the number of correlated variables produced were very large. The second major issue was that the sample size was small compared to the number of variables.

To overcome these problems, several approaches were pursued including Partial least squares method, Kolmogorov-Smirnov tests, sensitivity/specificity calculations for ROC curves, and Bootstrap methods for calculating confidence intervals. This thesis presents results using these methods and insights gleaned from these results.

2. METHODOLOGY

2.1 Examination and Standardization of the Initial Raw Data

In the study data, there were 728 infrared absorption variables for the corresponding 728 frequencies that were measured per location on a cellular suspension. Vero cells had been cultured. Some cells would be withdrawn from the culture and divided into two subcultures. One subculture would be infected with hsv1, the hsv1 sample, and one would not be infected, the mock sample. The two cultures would then be allowed to grow for a specified number of hours, 2, 4, 6, and 24 hours. The first dataset included cells that were cultured for 2, 4, 6, and 24 hours. The second set of data included 8 samples that were each cultured for 24 hours. Each of the two subcultures, hsv1 and mock, would then be placed into separate cellular suspensions and scanned.

The following table shows the total number of samples in each group by disease status within the first dataset.

Table 1. Total Number of Samples in Each Group by Disease Status

Group	Diseased (hsv1)	Non-diseased (mock)
2 hour	65	54
4 hour	45	82
6 hour	83	72
24 hour	111	180

For each sample, the spectroscopic measurements were taken and stored in a file – one file per sample. Each file was saved in comma-separated-value (csv) ASCII format where each line consisted of a spectrum frequency followed by the corresponding absorption value. Files were organized into directories where a directory contained all samples for a particular disease status and hour measurement. Using the Python language, the files were merged into one file, the

master dataset, so that each row of the master dataset contained the disease status (mock or hsv1), the group (2, 4, 6, and 24 hours), the file number within the group, the filename, the spectrum value, and the absorption value. The Python program for loading one directory at a time into the dataset file is shown in Appendix A. The DOS batch script that changed to each directory in the directory structure that contained the data and loaded the files from that directory is shown in Appendix B.

Next, to get an initial visual idea of the nature of the data and secondly to confirm the need for standardization of the data, Python and SAS were used. A Python script filtered the master dataset data by group and Vero cell type (hsv1/mock) into csv files. Python code then generated SAS programs for reading and plotting the data from a SAS template file. The SAS programs were run and the Python script renamed the generated output GIF file to meaningful file names. The Python script is shown in Appendix C and the SAS template file is shown in Appendix D. Examination of the data revealed that differences between the mean absorption of one sample (file) and another was far greater than the variation in absorption values within a file. Sometimes this inter-sample variation was more than 10 times the intra-sample variation. Therefore, the need for data standardization became obvious.

Standardization was accomplished by the following procedure. All samples were examined to find the highest minimum spectrum value that the sample was measured. That frequency was 799.492 cm^{-1}. Similarly, the lowest maximum frequency was found to be 1500.615 cm^{-1}. All frequency measurements outside of this range were excluded so that samples could be compared at the same frequencies. For plotting convenience, the range was slightly narrowed to include only frequencies between 800 and 1500. The Python script shown in Appendix A also does this filtering now. Within the above range, all measurements were found to have been taken

at the same set of frequencies after the exclusion of a sixth decimal point in some of the data. In other words, some spectrum values were record to six decimal places and some to 5 decimal places depending on the day the measurement was taken. Once the impact of this extra digit of precision was removed, all samples were found to have been measured at the same frequencies. This was accomplished by the Python script shown in Appendix E. Next, the mean value and standard deviation were calculated for each sample's absorption values. A standardized value was the calculated for each frequency by the formula standardized absorption = (absorption – standard deviation) / mean. The Python script that accomplished this is shown in Appendix F. The standardized absorption values were then plotted using code similar to that used for plotting the raw data. The Python script and SAS template script that did this are shown in Appendixes G and H respectively.

2.2 Variable Reduction

The next step was a variable reduction process. Each sample had 728 spectrum variables at which an absorption value were measured (starting at 799.492 cm^{-1}). Many of these spectrum variables might not differ overall in value between mock and hsv1 tissues. Our goal was to eliminate those spectrum values where the differences between hsv1 and mock absorption values were not highly significant. To do this we used the Wilcoxon-Mann-Whitney test and excluded any frequency (i.e. variable) where $Z < 8.0$. The high value of 8.0 was chosen instead of 4.0 because the plateaus in the Z values occur above 8 in all plots and choosing only 4.0 would eliminate very little of the less significant data.. This was done for the 2, 4, 6, and 24 hour groups.

The Wilcoxon rank sum test was initially developed by Wilcoxon in 1945. An equivalent test, the Mann-Whitney U test, was developed by Mann and Whitney in 1947. The two tests are

collectively called the Wilcoxon-Mann-Whitney test. The test is a nonparametric rank based test to assess whether two independent random samples belong to the same population by assessing whether their population distributions are the same or not. This is achieved by the following simple algorithm (Sprent 2001). For sample X of size m and sample Y of size n, combine the two samples into one sample of size m+n and order, i.e. rank, the values from 1 to m+n. Then separate out the samples and sum the rank values given to the X sample elements and call it S_m. Sum the rank values given to the Y sample elements and call it S_n. If S_m is much larger than S_n, then there is evidence that they belong to different distributions. StatXact will calculate the exact p-value for these two number. SAS also provides Wilcoxon-Mann-Whitney calculation p-values as well as a Z value via the NPAR1WAY procedure. The Z value provided, and which we used, is the Z value one would get if one took the calculated Wilcoxon-Mann-Whitney p-value and treated it as if it were a p-value from a normal distribution and then took the associated Z value. The python code for this is shown in Appendix I. The SAS template code used by the python code is shown in Appendix J. The python code that filtered through the SAS listing file to get the Z values and the corresponding frequencies and dump them to a csv file is shown in Appendix K.

The number of frequency variables was then further reduced by taking the average of the absorption values for the five adjacent spectrum variables in each sample and creating replacing the five variables with the median frequency among the five and assigning it this average absorption value. This acted as a kernel smoothing function that smoothed out localized noise. These new spectrum variables were used for all the disease prediction modeling below. The code that did this is included in all the the associated Partial least squares and Kolmogorov-Smirnov code referred to in the below sections.

2.3 Prediction of Disease Status

Logistic regression would be used usually with binary outcome data. However, logistic regression becomes unstable and misleading if either of the following conditions exist. Too many variables exist compared to sample size. When initially investigating Logistic regression against the data, SAS would report a failure to meet convergence criteria until the variable count was reduced to 1 variable for roughly every 10.5 samples. For 24 hour data this would require a Z cutoff of 13.85 and an overly fitted model. There exists collinearity between several variables. Correlations among the predictors make it seem that no one variable is important when all the other collinear variables are in the model (Agresti 2002). Hence logistic regression could not be utilized directly.

Instead, we used the Partial least squares (PLS) regression in SAS to derive a model that explained 95% or more of the variation in the independent variables. A second reason for using PLS was that PLS does not require a normally distributed response variable. It is safest when dealing with biological data to assume that the response variable is non-normal.

PLS was developed by Herman Wold and published as an econometric technique (Wold 1966) . It is now used by chemical engineers and many others . PLS works by finding linear combinations of predictors, termed latent factors, to predict what are called manifest factors. Both the latent factors and predictors are linear combinations of the dependent and independent variables in the data.

The PLS method computes the factors by extracting one factor at at time. Let $X=X_0$ be a centered and scaled matrix of predictors and $Y=Y_0$ be the centered and scaled matrix of response variables. The PLS method starts with a linear combination $\mathbf{t} = X_0\mathbf{w}$of the predictors, where **t** is

called a score vector and **w** is its associated weight vector. The PLS method predicts both X_0 and Y_0 by regression on **t**.

$$\hat{X}_0 = \boldsymbol{tp}', \text{ where } \boldsymbol{p}' = (\boldsymbol{t}'\boldsymbol{t})^{-1}\,\boldsymbol{t}'X_0$$
$$\hat{Y}_0 = \boldsymbol{tc}', \text{ where } \boldsymbol{c}' = (\boldsymbol{t}'\boldsymbol{t})^{-1}\,\boldsymbol{t}'Y_0$$

The vectors **p** and **c** are called the X and Y loadings, respectively. The specific linear combination $\mathbf{t} = X_0\mathbf{w}$ is the one that has a maximum covariance **t'u** with some response linear combination $\mathbf{u} = Y_0\mathbf{q}$. Another characterization is that the X and the Y weights **w** and **q** are proportional to the first eigenvectors of $X_0'Y_0Y_0'X_0$ and $Y_0'X_0X_0'Y_0$ respectively. This gives the first PLS factor that is extracted. The second factor is then extracted by replacing X_0 and Y_0 with the X and Y residuals from the first factor. $X_1 = X_0 - \hat{X}_0$ and $Y_1 = Y_0 - \hat{Y}_0$ Each successive desired factor is extracted in the same way. The more factors that are created, the more variance is explained in the dependent and independent variables.

We used the 24 hour hsv1 versus mock data to generate a model with PLS that explained 95% of the variation in the independent variables, the absorption variables. Since we were told that 24 data is considered to be a gold standard for hsv1 spectrum analysis, we used this model against the 2, 4, and 6 hour data and evaluated how well the model performed for those data.

To be able to evaluate the usefulness of the model for diagnosing a patient's hsv1 status, a Receiver Operating Characteristic (ROC) curve is needed. SAS generates ROC curves under the Logistic regression procedure. Therefore, the orthogonal factors from the PLS model that accounted 95% of the independent variable variation were used to have PLS generate a disease predictor model that was stored in the variable, ypred. This predictor model variable was then used as the model in a Logistic regression. From the Logistic regression in SAS we then obtained the ROC curve and and a table of the curves specificity and sensitivity values. The Python and

SAS code that accomplished the 5 variable reduction, PLS modeling, and use of Logistic regression to generate a ROC curve is shown in Appendixes L and M.

2.4 Test for Distribution Equivalence Within Mock and Hsv1 Groups

For a diagnostic tool to be valid especially with Vero cell which can be cultured indefinitely, there should be a common underlying distribution for all the healthy Vero cells and there should be a change in the distributions of the diseased Vero cells as the disease progresses with time. To test this hypothesis on the experiment data, PLS was run against the 24 mock sample and mock sample from a different hour. For each location on a sample, PLS generated a single number which would act as a predictor of the time group that the location belonged to. This was done for 2 hours versus 24 hours, 4 hours versus 24 hours, and 6 hours versus 24 hours. Then the Kolmogorov-Smirnov (KS) test was used to compare the distribution of the predicted time values for each comparison pair. If the 2 hour mock cells behaved the same as the 24 hour mock cells, then their predicted hour values - which were based on each location's absorption values – should be similar in distribution and return a KS p-value that is not significant (i.e. p-value > 0.05).

The Kolmogorov-Smirnov test was proposed by Kolmogorov(1933) and Smirnov(1936). It is a nonparametric test of equality that compares the empirical distribution function of two samples. It is sensitive to differences in both location and shape of the empirical cumulative functions of the two samples. The test statistic is defined as follows. For a sample $\{x_j\}$ where j=1,2,...,n the empirical distribution function (EDF) is defined as the function F(x) where:

$$F(x) = \frac{1}{n}(number\ of\ x_j \le x) = \frac{1}{n}\sum_{j=1}^{n} I(x_j \le x)$$ where I() is an indicator function.

The Kolmogorov-Smirnov statistic measures the maximum deviation of the EDF within the classes from the pooled EDF by computing the statistic as follows.

$$KS = \max_j \sqrt{\frac{1}{n}\sum_i n_i (F_i(x_j) - F(x_j))^2} \quad \text{where } j = 1, 2, \ldots, n$$

The Python program and SAS template that calculated the PLS and KS test are shown in Appendixes N and O.

2.5 New Directions: Loading Eight Mock and Hsv1 Groups at 24 Hours Exposure

Since the results of the Kolmogorov-Smirnov test showed that the 2, 4, 6, and 24 hour mock populations all differed, and since there were still apparent differences between the hsv1 and mock samples at a given hour, it was determined that we needed to look for consistent contrasts between the mock and hsv1 population at a given hour. The second set of sample data was used for this analysis. This data consisted of 8 samples of hsv1 and mock data for the 24 hour period.

The new data had been collected in the same manner as the previous data. Therefore, we standardized the data using the same code for standardization that was used earlier except for minor changes relating to directory names and similar things so that it could be manipulated in a SAS dataset. The data loading programs are shown in Appendix P.

With this new data, it was discovered that each sample set was actually a set of measurements of different locations on the same Vero cell culture. Hence there were only 8 hsv1 samples and 8 mock samples. For example, there were 79 spectrum measurements of the single hsv1 culture that was examined on March 24, 2008. Therefore it made more sense to calculate the mean values of absorption at each frequency to generate a single mean curve to represent that culture. These mean curves were calculated and plotted using the code in Appendix Q.

2.6 Calculation of the Wilcoxon-Mann-Whitney Curve

To find at what spectrum the hsv1 sample differed from the mock sample, we calculated Wilcoxon-Mann-Whitney Z scores for each spectrum frequency using the 8 mean hsv1 values and 8 mean mock values. The code for this is in the second half of the Curves.sas program shown in Appendix Q.

2.7 Generation of a Difference of Composites Statistic D_C

From the results we then chose the frequencies that had the largest significance – those frequencies at which the absolute value of the Z value was greater than or at least near 4.0. These resulting frequencies consisted of two frequency ranges. Then for each of the 8 hsv1 and 8 mock mean curves, we created a statistic. The statistic, let's call it the difference of composites, D_C, takes the sum of the mean values within the region where approximately Z > 3, and subtracts from it a sum of the frequency region where approximately Z < -3. Below is a formula that represents this.

$$D_C = \sum_{i=1}^{m} \bar{x}_i - \sum_{j=1}^{n} \bar{x}_j \quad \text{where i and j are frequencies such that} \quad Z_i > 3 \quad \text{and} \quad Z_j < 3$$

The SAS code that accomplished this is GetDataForBootStrap.sas shown in Appendix R.

2.8 Evaluation of D_C Statistics by ROC AUC value, Specificity, and Confidence Intervals

In the final stage of our analysis, we evaluated the performance of this new D_C statistic. In the last part of the program, GetDatForBootStrap.sas, we calculated the mean and variance of our 8 D_C values. We then used those numbers to generate an Area Under Curve (AUC) value for the Receiver Operating Characteristic (ROC) Curve.

We used the following formula to calculate the AUC. $$AUC = P\left(Z < \frac{\overline{D}_{C_{Hsv1}} - \overline{D}_{C_{Mock}}}{S^2_{Dc_{Hsv1}} - S^2_{Dc_{Mock}}}\right)$$

To calculate the ROC specificity values given 0.95, 0.9, and 0.8 sensitivities we use the following formula. $$Specificity = P\left(Z < \frac{\overline{D}_{C_{Hsv1}} - \overline{D}_{C_{Mock}} - Z_\alpha \cdot S_{Dc_{Hsv1}}}{S_{Dc_{Mock}}}\right) \text{where } \alpha = 0.95,\ 0.9,\ 0.8$$

Calculating the confidence intervals for the AUC and the specificities, requires the same formulas but applied in a parametric bootstrap to generate 10,000 or 100,000 values which we then take the 95% quartiles of. This is accomplished in the Python program, Bootstrap3.py, shown in Appendix S.

3. RESULTS

3.1 Examination and Standardization of the Initial Raw Data

The initial examination of the data involved the plotting of the raw data for hsv1 versus mock. As shown in Figures 1 through 4, the inter-sample variation exceeds the intra-sample variation. In some cases, the intra-sample variation is quite small and almost appears to not exist, whereas in other samples within the plot, the intra-sample variation is quite pronounced even within the same Vero cell type (hsv1 or mock). However, on all four plots, one notices that all the curves do appear to have similar peaks and valleys, albeit with different intensities.

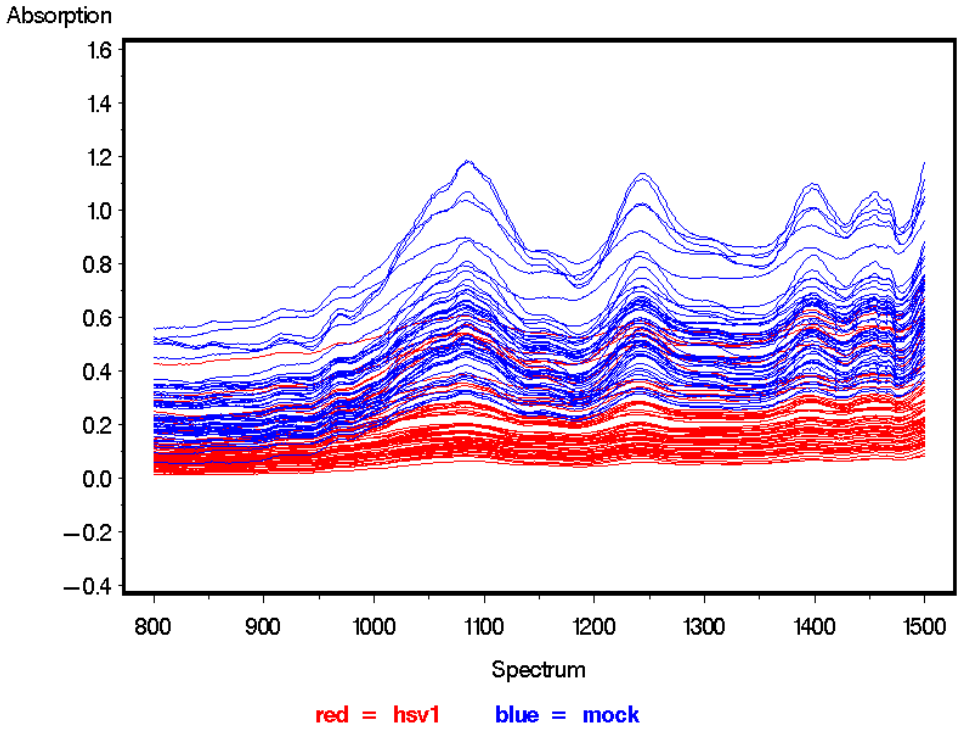

Figure 1. Raw Hsv1 Versus Mock Data at 2 Hours

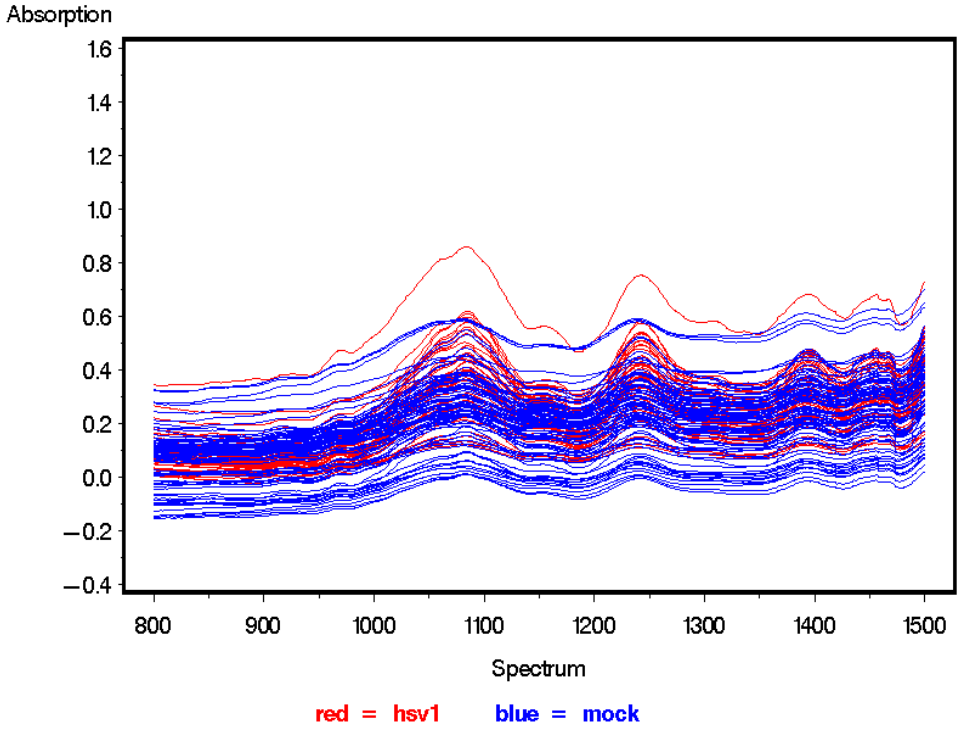

Figure 2. Raw Hsv1 Versus Mock Data at 4 Hours

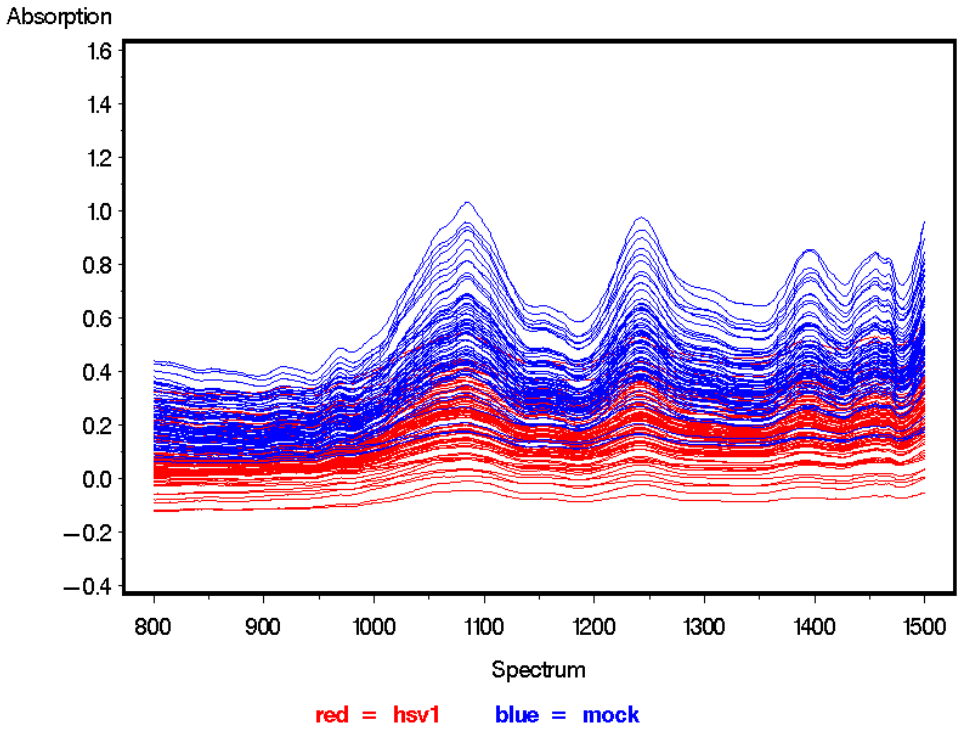

Figure 3. Raw Hsv1 Versus Mock Data at 6 Hours

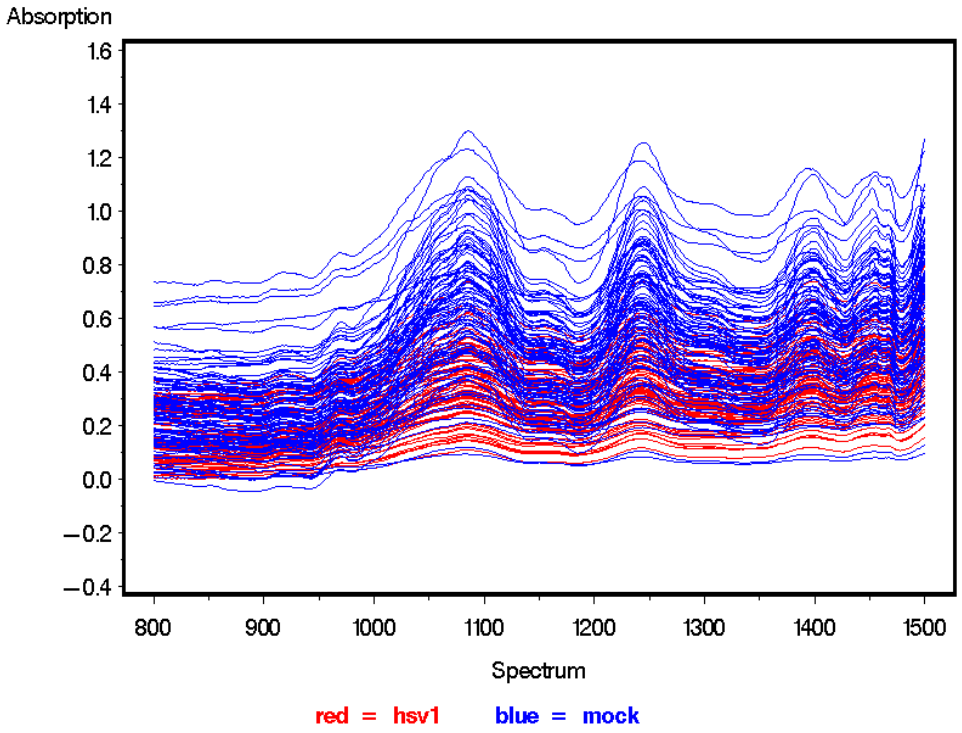

Figure 4. Raw Hsv1 Versus Mock Data at 24 Hours

In the following four plots, Figures 5 through 8, of standardized data, we can see these results. Inter-sample shifting disappears. Each sample now scales to similar minimum and maximums thus eliminating the variation in the spectroscope's ability to detect variations in absorption perhaps caused by differences in thicknesses of the sample or other factors. Now, the areas where mock and hsv1 overlap are far more apparent as are the areas where there is significant divergence between the two. Also eliminated are the misleading data points which appeared to show negative absorption values, i.e. the location on the sample appeared to have generated its own light.

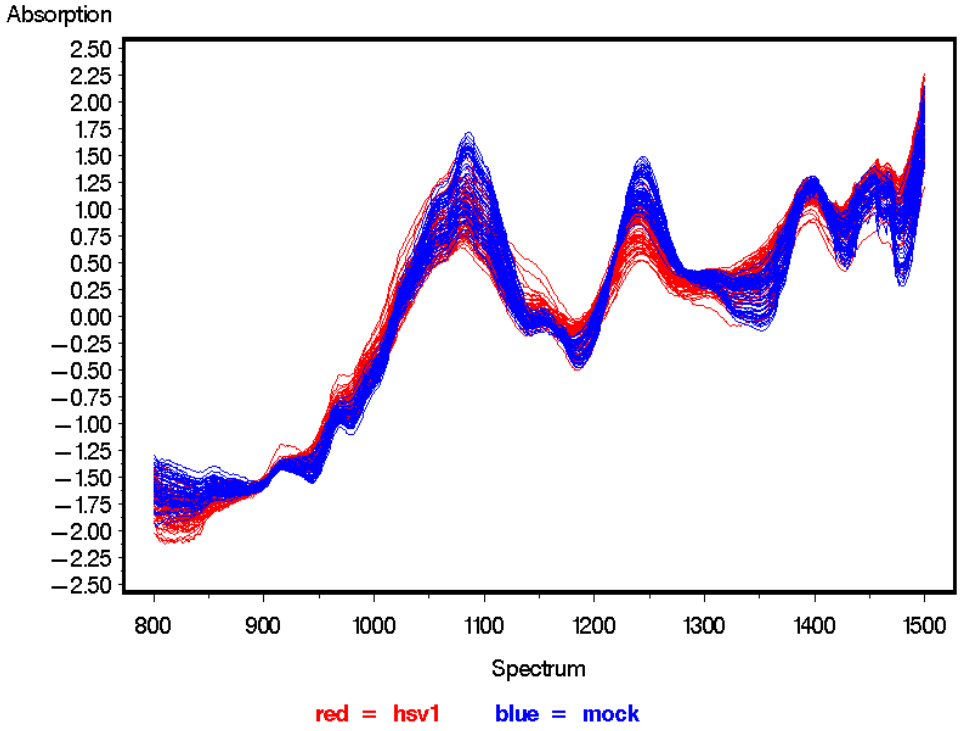

Figure 5. Standardized Hsv1 Versus Mock Data at 2 Hours

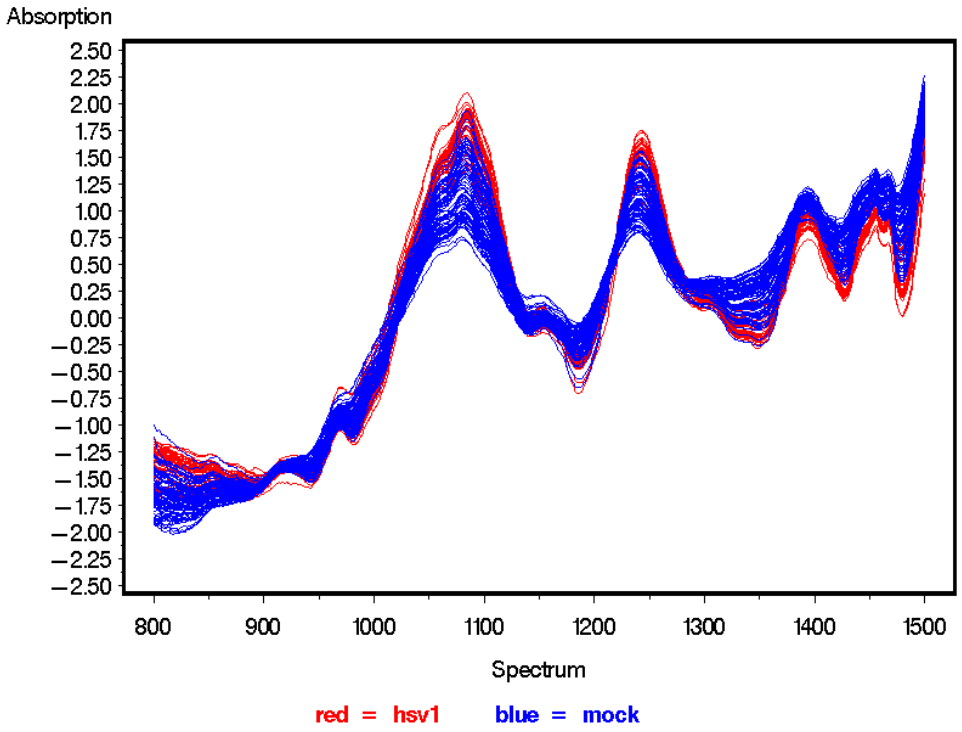

Figure 6. Standardized Hsv1 Versus Mock Data at 4 Hours

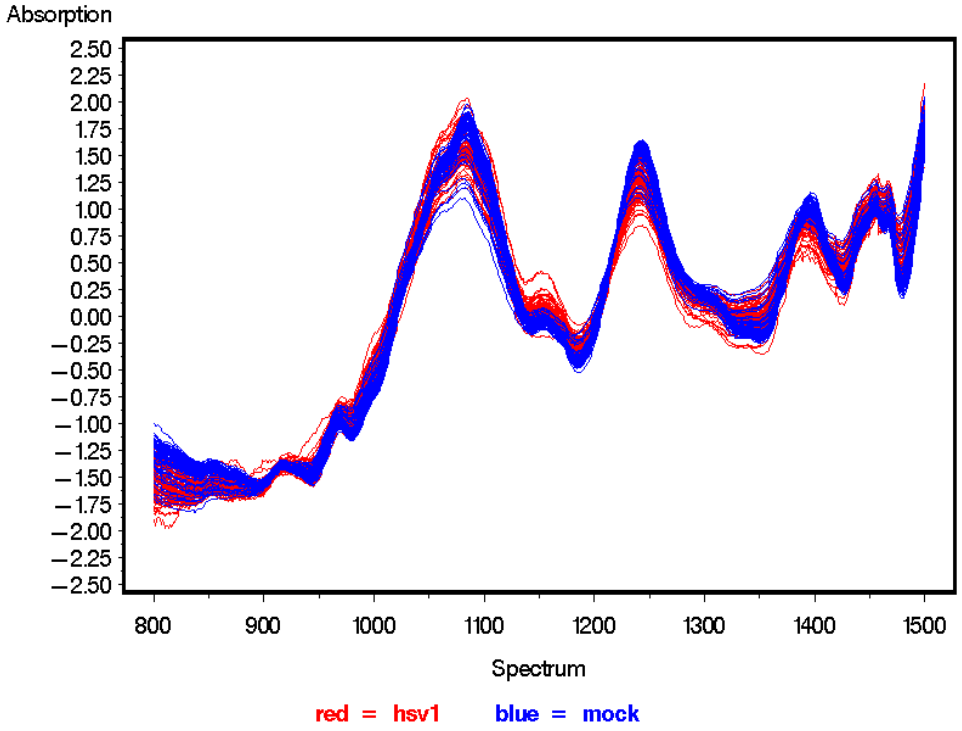

Figure 7. Standardized Hsv1 Versus Mock Data at 6 Hours

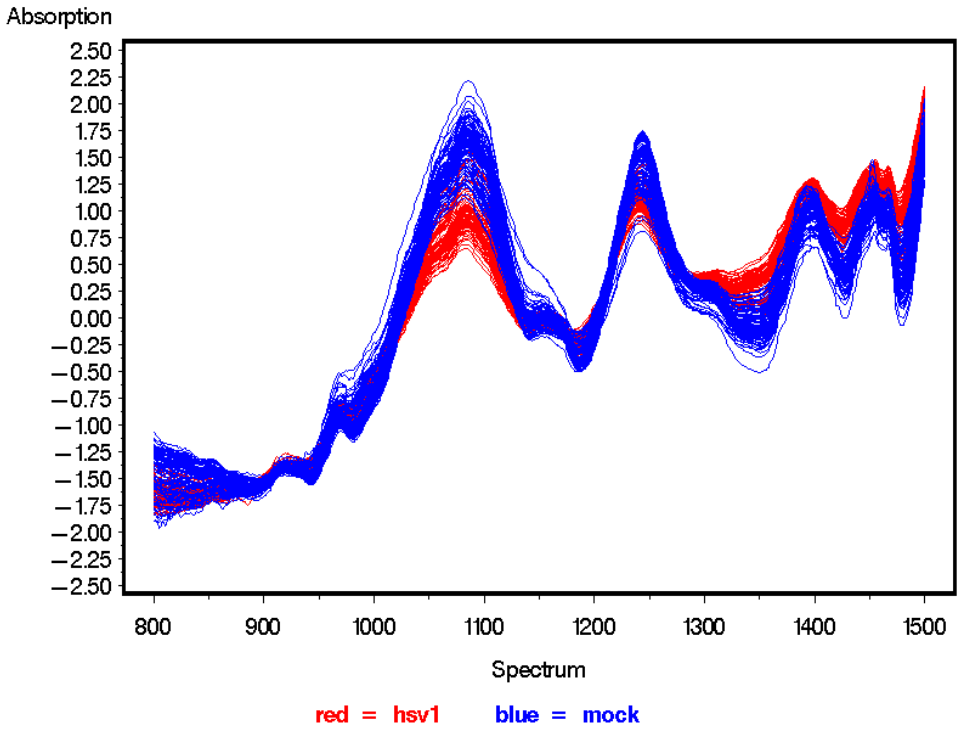

Figure 8. Standardized Hsv1 Versus Mock Data at 24 Hours

3.2 Variable Reduction

Next, we used the Wilcoxon-Mann-Whitney statistic's Z value to identify the spectrum variables that showed significant variation between hsv1 and mock samples. We derived the following plots using OpenOffice.org's Calc (an Excel clone) from the csv files generated by the Python and SAS code for generating the WMW curves.

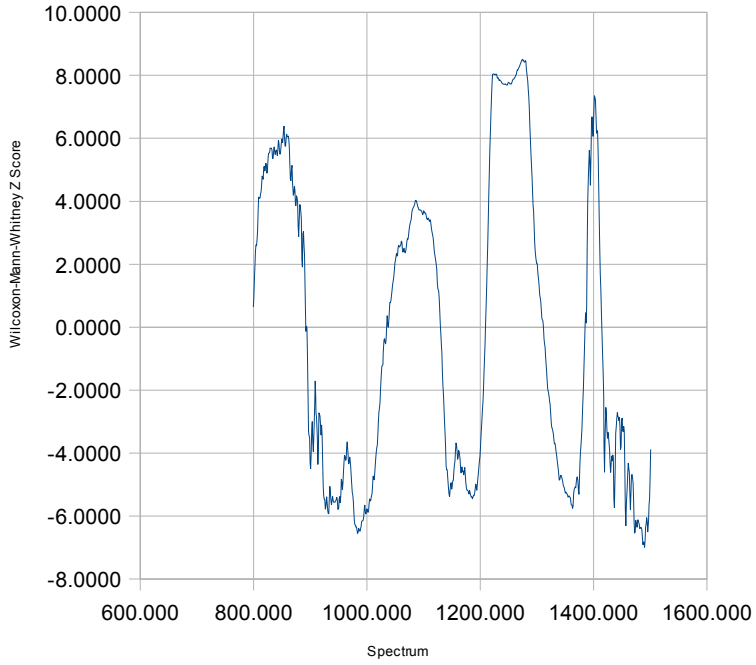

Figure 9. Wilcoxon-Mann-Whitney Statistic's Z Score for Hsv1 Versus Mock Data at 2 Hours

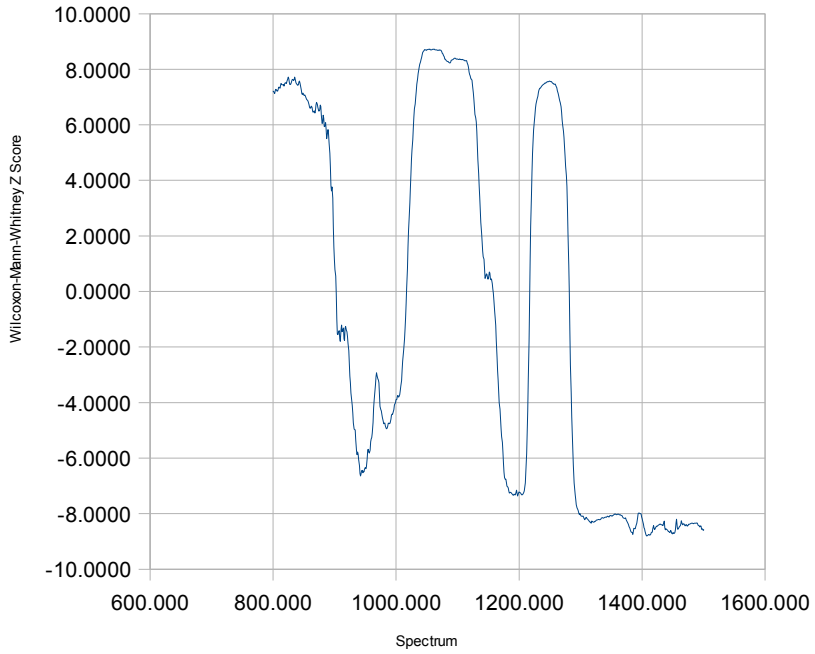

Figure 10. Wilcoxon-Mann-Whitney Statistic's Z Score for Hsv1 Versus Mock Data at 4 Hours

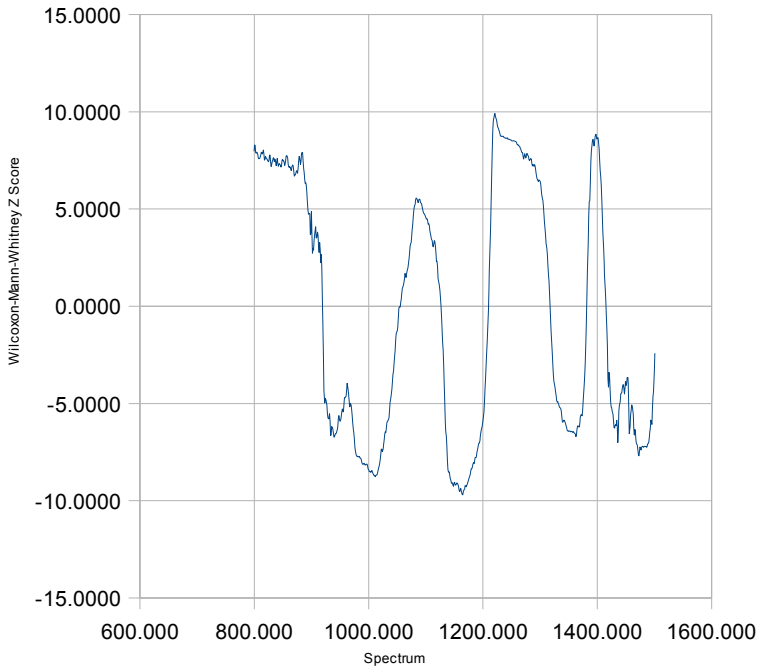

Figure 11. Wilcoxon-Mann-Whitney Statistic's Z Score for Hsv1 Versus Mock Data at 6 Hours

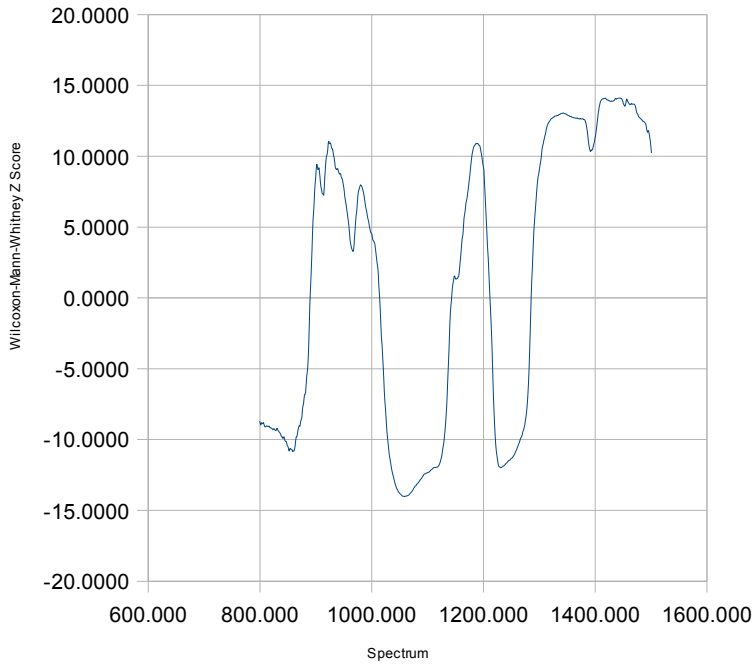

Figure 12. Wilcoxon-Mann-Whitney Statistic's Z Score for Hsv1 Versus Mock Data at 24 Hours

3.3 Prediction of Disease Status

When we used Partial least squares (PLS) regression to derive a 24 hour model that explained 95% or more of the variation in the independent variables, SAS generated 6 latent factors which explained 96% of the variance according to the following breakdown.

Table 2. Results of PLS Analysis

	number of independent variables	factors and total X variation accounted for	
24 hour hsv1 vs mock	103	1	73.23%
		2	81.77%
		3	90.81%
		4	92.92%
		5	94.90%
		6	96.03%

The 103 independent variables were the resulting number of absorption variables each with its particular frequency from the previous variable reduction sections that were run against the 24 data. The code generated these same frequency variables for the 2, 4, and 6 hour data. Substituting the disease predictor model, ypred, that was generated by these 6 orthogonal factors into a Logistic regression as model variables allowed the generation of the ROC curves. The below table describes the ROC curves for each group at the sensitivity values of greatest interest.

Table 3. Partial ROC Curve for Mock Versus Hsv1 Data

Sensitivity	Specificity measurements at			
	2 hours	4 hours	6 hours	24 hours
99%	16.7%	52.4%	0.0%	100.0%
95%	46.3%	63.4%	63.9%	100.0%
90%	51.8%	71.9%	76.4%	100.0%
80%	87.0%	82.9%	86.1%	100.0%

3.4 Test for Distribution Equivalence Within Mock and Hsv1 Groups

Because we compared 24 hour mock to 2, 4, and 6 hour mock, this resulted in new PLS models. In each of these models we used 6 factors resulting in 95 % to 96% of the independent variable variance being explained.

Table 4. Results of PLS analysis for 24 Hour Mock Versus 2, 4, and 6 Hour Mock

groups compared	number of independent variables	factors and total X variation accounted for	
2 hour vs 24 hour	103	1	68.0%
		2	78.7%
		3	91.0%
		4	92.8%
		5	94.9%
		6	95.4%
4 hour vs 24 hour	103	1	69.5%
		2	81.4%
		3	91.6%
		4	92.8%

		5 6	94.2% 96.5%
6 hour vs 24 hour	103	1 2 3 4 5 6	65.2% 83.8% 91.1% 92.3% 94.8% 96.6%

When we ran the Kolmogorov-Smirnov tests on the above models, we had hoped to find that the mock groups all belonged to the same distribution and that the 2, 4, or 6 hour hsv1 groups had a different distribution than the 24 hour hsv1 group. What we discovered in the below results was that there was a strong statistical difference (p value < 0.0001) between the 24 hour mock group and each of the other mock groups (2, 4, and 6 hour mock). We also ran similar test for the hsv1 groups. They also had different distributions (p value < 0.0001 for 2, 4, and 6 hour hsv1) from the 24 hour hsv1 group, but then that was expected.

Table 5. Kolmogorov-Smirnov Results for 24 Hour Mock Versus 2, 4, and 6 Hour Mock

Group comparison	Kolmogorov-Smirnov Two-Sample Test (Asymptotic)	p value
2 hour vs 24 hour	5.911106	< 0.0001
4 hour vs 24 hour	6.697076	< 0.0001
6 hour vs 24 hour	6.456331	< 0.0001

Table 6. Kolmogorov-Smirnov Results for 24 Hour Hsv1 Versus 2, 4, and 6 Hour Hsv1

Group comparison	Kolmogorov-Smirnov Two-Sample Test (Asymptotic)	p value
2 hour vs 24 hour	4.769696	< 0.0001
4 hour vs 24 hour	4.437060	< 0.0001
6 hour vs 24 hour	4.961718	< 0.0001

3.5 New Directions: Loading Eight Mock and Hsv1 Groups at 24 Hours Exposure

After loading the 8 sets of mock and hsv1 sample data, and standardizing the second dataset using basically the same algorithm that we used for the first dataset, we calculated the means of the absorption values for each frequency variable for each of the 8 hsv1 and 8 mock 24 hour groups. The following plot shows these 16 mean curves.

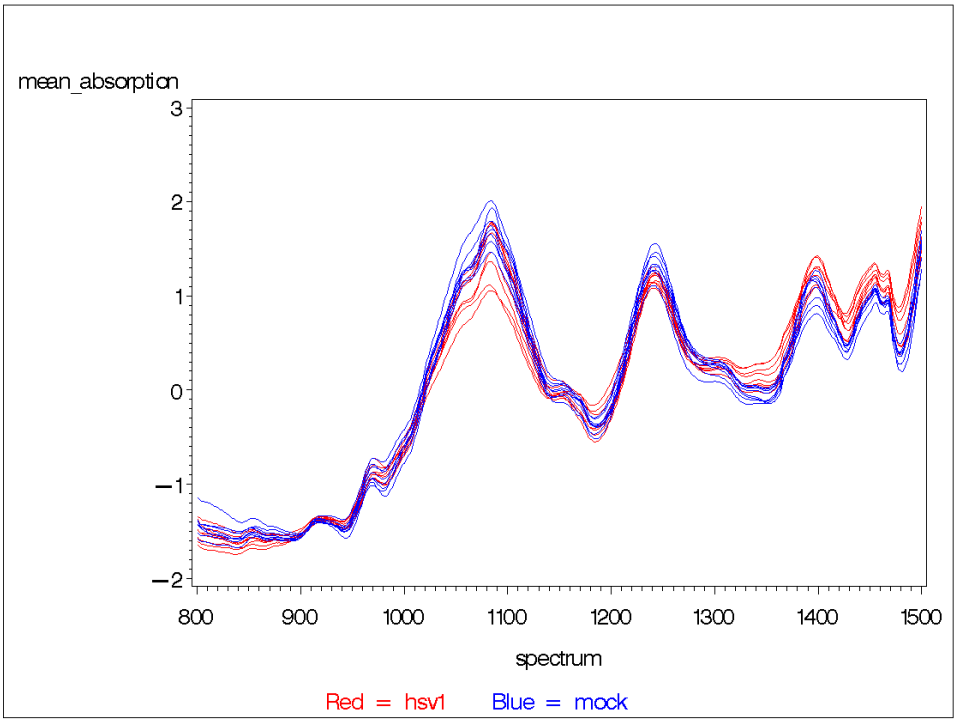

Figure 13. Mean Absorption Curves for 8 Hsv1 and 8 Mock Samples at 24 Hours

It will be noticed that the plot of these 16 mean curves closely resembles the plot of the initial dataset. The overall location and relative scale of the peaks and valleys is very similar, however, the normalized first dataset exhibited far greater variance in the height and depth of the peaks and valleys of the 24 hour data than what is seen here. Some of this may be explainable by the fact that we are averaging out the extremes within each sample in this plot of means above. However, part of the variance is likely also due to variance between each sample. Ascertaining how much of the variance reduction is due to taking the means will require further sampling in the future. Currently our goal is to find an indicator statistic, hence, the next stage of analysis.

3.6 Calculation of the Wilcoxon-Mann-Whitney Curve

Next, we plotted the Wilcoxon-Mann-Whitney Z scores which show the spectrum variables that had significant variation between hsv1 and mock mean curves.

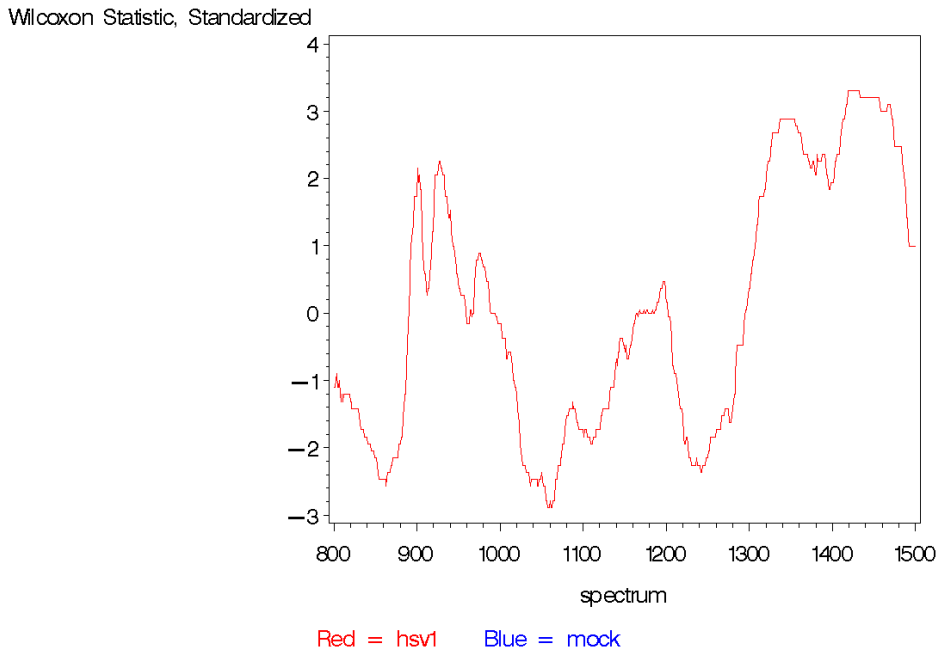

Figure 14. Wilcoxon-Mann-Whitney Statistic's Z Score for 8 Hsv1 Mean Curves Versus 8 Mock Mean Curves at 24 Hours

In the interest of verifying whether using the mean curves to calculate a Z statistic gives a different result than using the standardized data, the Wilcoxon-Mann-Whitney Z Score was also calculated on the standardized data (that was used to generate the means) and yielded the following plot which is almost identical except for the scale of the Z value and some small differences in two peaks near 900 cm^{-1} and 1000 cm^{-1}.

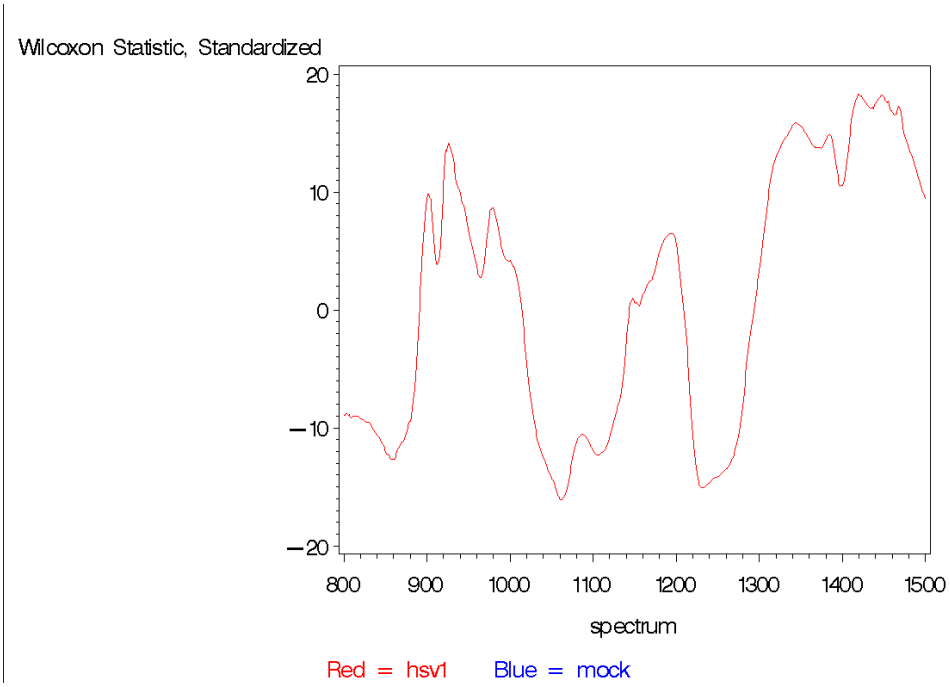

Figure 15. Wilcoxon-Mann-Whitney Statistic's Z Score for Hsv1 Standardized Data Versus 8 Mock Standardized Data at 24 Hours

3.7 Generation of a Difference of Composites Statistic D_C

From the above plot we can see that the spectrum from 1415 cm^{-1} to 1455 cm^{-1} has the highest positive Z values. In this case $Z > 3.0$. The region with the largest negative Z values were in the range 1050 cm^{-1} to 1070 cm^{-1}. Using the code in GetDataForBootStrap.sas, the following 8 pairs of D_C values were generated and their mean and variance values.

Table 7. Difference of Composites Statistic, $D_{C,}$ for Hsv1 and Mock for 8 Samples

	Sample 1	Sample 2	Sample 3	Sample 4	Sample 5	Sample 6	Sample 7	Sample 8
Hsv1	21.1279	27.9380	12.8544	11.3004	19.1465	8.7847	21.9278	5.1373
Mock	7.1749	1.6203	2.6050	3.9264	5.5607	3.0025	-10.1459	-2.0627

Table 8. Mean and Variance of Difference of Composites Statistic, $D_{C,}$ for Hsv1 and Mock for 8 Samples

	Mean	Variance	Standard Deviation
Hsv1	16.0271	59.4121	7.7079
Mock	1.4602	29.4831	5.4298

It would appear that mock sample 7 could be an outlier since without it the mean of the remaining 7 mock values is 3.1182 and the standard deviation is 2.9563 which places -10.1459 more than 3 standard deviations away. However since there are only 8 mock samples, excluding it would be premature. Furthermore, to perform the later bootstrap test we need to maintain normality in our data values and selectively removing values would remove this normality.

3.8 Evaluation of D_C Statistics by ROC AUC value, Sensitivity, Specificity, and Confidence Intervals

Using the previously mentioned formula for the area under the ROC curve, we get the following value. $AUC = P\left(Z < \frac{16.0271-1.4602}{59.4121+29.4831}\right) = P(Z<1.5450) = 0.9388$.

Using the bootstrap3.py, which uses the nonparametric Bootstrap algorithm to generate 100,000 samples with replacement, where each sample consists of the 8 paired hsv1/mock means, we get for the 0.9388 AUC value a moderately narrow 95% confidence interval of [0.8907, 0.9924].

Since it is over 90%, a 0.9388 AUC would indicate a very good ability of the test statistic to discriminate between diseased and healthy cells.

From a medical diagnostic perspective, the more important numbers are the specificity values, S_p, given sensitivity values, S_e, of 0.95, 0.9, and 0.8. Using the previously described equation for specificity we get the following three values.

$$S_p\,(at\ S_e{=}0.95) \;=\; P\left(Z<\frac{16.0271-1.4602-1.6449\cdot 7.7079}{5.4298}\right) \;=\; P(Z<0.3478) \;=\; 0.6360$$

$$S_p\,(at\ S_e{=}0.90) \;=\; P\left(Z<\frac{16.0271-1.4602-1.2816\cdot 7.7079}{5.4298}\right) \;=\; P(Z<0.8635) \;=\; 0.8061$$

$$S_p\,(at\ S_e{=}0.80) \;=\; P\left(Z<\frac{16.0271-1.4602-0.8416\cdot 7.7079}{5.4298}\right) \;=\; P(Z<1.4880) \;=\; 0.9316$$

Using the same nonparametric Bootstrap algorithm with 100,000 iterations, we get for 63.6% specificity a 95% confidence interval of [0.04427, 0.9544]. For the 80.61% specificity the 95% confidence interval is [0.4199, 0.9964]. Finally, for the 0.9316 specificity, the 95% confidence interval is [0.8345, 0.9999]. These confidence intervals are disappointingly large, but it might be expected.

It should be noted that calculating 100,000 iterations enabled us to get a very stable estimate of these three confidence intervals. The 95% confidence intervals for specificity are so wide because, in addition to the width of the AUC CI, the standard deviation term on the top of the specificity equation increases the CI width because of its variability from sample to sample. Also, the smaller standard deviation value in the bottom term of the equation also contributes to the width of the confidence intervals.

$$AUC \;=\; P\left(Z<\frac{\overline{D}_{C_{Hsv1}}-\overline{D}_{C_{Mock}}}{S^2_{Dc_{Hsv1}}-S^2_{Dc_{Mock}}}\right)$$

$$Specificity \;=\; P\left(Z<\frac{\overline{D}_{C_{Hsv1}}-\overline{D}_{C_{Mock}}-Z_{\alpha}\cdot S_{Dc_{Hsv1}}}{S_{Dc_{Mock}}}\right)\text{where}\,\alpha=0.95,\ 0.9,\ 0.8$$

4. CONCLUSIONS AND DISCUSSION

In conclusion, the analysis was able to find a simple test statistic, D_C, to predict disease status. The mean and variance of the D_C statistics for hsv1 and mock data resulted in distribution curves for the statistic that were well separated as indicated by the AUC value of 0.9388. The resulting specificity values were moderately good – 63%, 80%, and 95% for sensitivity values of 95%, 90%, and 80% respectively. However, the confidence intervals for both the AUC and even more so for the specificity values were highly impacted by the small sample size of 8 mean mock and hsv1 curves. However, based on the first dataset, we found it difficult to draw a conclusion regarding whether an earlier cell culture than 24 hours could be used to predict disease status, because of the significant difference in distributions between the 24 hour and the 2, 4, and 6 hour groups for mock.

In the future, to confirm the usefulness of the D_C statistic for predicting disease status, an increased sample size is needed. This will help to reduce the confidence interval width for the specificity measurements and it would enable the use of cross-validation to estimate the AUC after shrinkage. One could have just 10 hsv1 and 10 mock measurements per sample pair, but have 10 sample pairs for 2 hour data, 10 sample pairs for 4 hour data, et cetera. It might also be beneficial to draw additional samples from time periods in between 24 hours and 6 hours to better understand the nature of the inversion of Z scores between these two groups.

Finally, it might be worthwhile to explore other methods of reducing the number of frequency variables, such as Fourier decomposition. One could find the Fourier equation coefficients that explain at least 95% of the variability in the mean values and use these coefficients for PLS or other model generation.

REFERENCES

Chakravarti, R. and Laha, R.G. (1967), “Handbook of Methods of Applied Statistics, Volume I”, John Wiley and Sons, pp. 392-394.

Efron, B. and Tibshirani, R.J. (1993), “An Introduction to the Bootstrap”, Chapman & Hall.

Andrei N. Kolmogorov. “Grundbegrie der Wahrscheinlichkeitsrechnung”. Springer, Berlin, 1933. English translation (1950): Foundations of the theory of probability. Chelsea, New York.

SAS Institute Inc., 2003-2005 *SAS OnlineDoc*, version 9.1.3.

Sheets, R. (2000), “History and Characterization of the Vero Cell Line for Vaccines and Related Biological Products Advisory Committee”, FDA. www.fda.gov/ohrms/dockets/ac/00/backgrd/3616b1a.pdf

Sprent, P. and Smeeton, N.C. (2001), “APPLIED NONPARAMETRIC STATISTICAL METHODS, Third Edition”, Chapman & Hall/CRC, pp. 43-44.

Tobias, R.D., “An Introduction to Partial Least Squares Regression”, SAS Institute., Cary, NC.

Wang, H. (2007), “Some Conclusions of Statistical Analysis of the Spectropscopic Evaluation of Cervical Cancer”, Masters Thesis, Georgia State University.

Wold, H. (1966), "Estimation of Principal Components and Related Models by Iterative Least Squares," in *Multivariate Analysis*, ed. P. R. Krishnaiah, New York: Academic Press, 391 - 420.

APPENDIXES

The following appendixes contain the Python and SAS code used to perform the data management and analysis.

APPENDIX A: PYTHON SCRIPT FOR LOADING DATA FILES

```
#Python program, makeDataSet.py, for loading dataset
#
#It assumes the loaded data file contains several prefix lines
#of alpha text which are then followed by comma separated numeric
#lines with a spectrum frequency floating point number, a comma,
#and the absorption floating point number.
#
#The code also filters out any spectrum value less than 800 or
#greater than 1500.  Finally, it filters out one data file that
#contains an outlyer sample caused by the initial sample having
#a miniscule standard deviation (around.001 or so) and
#resulted in a totally different curve once it was standardized.
import os,os.path,sys


def getFiles(destFileName):
    df=0
    fileList=[]
    try:
        df = open(destFileName,'a+')
    except Exception,err:
        print "Failed opening file, "+os.path.basename(destFileName)
+". "+str(err)
    else:
        cwd=os.getcwd()
        fileList=os.listdir(cwd)
    return df,fileList

def getHeader(line,f):
    display='.'
    peak='.'
    size=0
    for i in range(0,10):
        linePair=line.upper().split(',')
        if 'DISPLAYDIRECTION' in linePair[0]:
            display=linePair[1].strip(' \r\n')
        if 'PEAKDIRECTION' in linePair[0]:
            peak=linePair[1].strip(' \r\n')
        if linePair[0][0].isdigit():
            size=i
            break
```

```
        line=f.readline()
    return display,peak,size,line

def loadLine(line,xyList):
    x,y = line.split(',')
    x = x.strip(' \n\r')
    y = y.strip(' \n\r')
    if (x[0].isdigit() or x[0] in ('+','-','.')) and (y[0].isdigit()
or y[0] in ('+','-','.')):
        if 800.0 < float(x) < 1500.0 :
            xyList.append((float(x),x,y))

def writeLine (x, y,
destFile,lineNum,fileNum,fileType,fileName,hours):
    out  = '%02.2d  ' % int(hours)
    out += '%03.3d  ' % fileNum
    if fileType[0].upper() == 'V':
        out += 'v  ' + '%4.4s  ' % fileType[1:5]
    else:
        out += 'n  ' + '%4.4s  ' % fileType[0:4]
    out += '%03.3d  ' % lineNum
    out += ('%-12.12s'%x + '      ')[0:13]
    out += ('%-12.12s'%y[0:9] + '      ')[0:10]
    out += ' '*14
    out += (fileName.replace(' ','') + ' '*75)[0:75]
    destFile.write(out+'\n')

def loadFile(fileName):
    xyList=[]
    f=open(fileName,'r')
    line=f.readline()
    display,peak,size,line=getHeader(line,f)
    while len(line)>10:
        loadLine(line,xyList)
        line=f.readline()
    f.close()
    return xyList

def main():
    (destFile,fileList) = getFiles(sys.argv[1])
    fileType=sys.argv[2]
    hours=sys.argv[3]
    fileNum=0
    if destFile != 0:
        for fileName in fileList:
```

```
                fileNum += 1
                if (fileName == "v-mock-human-t-nofix-24hpi-041608-
2cm-1_2000-700 filter(8).csv" and
                    hours=="24" and fileType=="vmock"
                   ):
                    continue
                print "Reading",fileName
                xyList = []
                xyList = loadFile(fileName)
                xyList.sort()
                lineNum=0
                for node in xyList:
                    lineNum += 1
                    writeLine(node[1],node[2],destFile,lineNum,fileNu
m,fileType,fileName,hours)

        destFile.close()
    else:
        print "Error opening file: "+sys.argv[1]

if len(sys.argv) == 1:
    print "syntax: ",sys.argv[0],"destFile fileType hours"
else:
    main()
```

APPENDIX B: DOS BATCH CODE FOR LOADING DATA FILES

```
REM Dos batch program to load data files, makeDSwith1call.bat
echo "LOAD DATA"
echo "---------------------"
cd "M:\Data\all mock had1 hsv1 2 hip 2cm-1 032608-032708\had1\all
1500-800"
m:\python25\python.exe  m:\data\makeDataSet.py
m:\data\mastDataSet.txt had1 02

cd "M:\Data\all mock had1 hsv1 2 hip 2cm-1 032608-032708\hsv1\ALL
1500-800"
m:\python25\python.exe  m:\data\makeDataSet.py
m:\data\mastDataSet.txt hsv1 02

cd "M:\Data\all mock had1 hsv1 2 hip 2cm-1 032608-032708\mock\all
1500-800"
m:\python25\python.exe  m:\data\makeDataSet.py
m:\data\mastDataSet.txt mock 02

cd "M:\Data\all mock had1 hsv1 4 hip 2cm-1 032808\had1\all 1500-800"
m:\python25\python.exe  m:\data\makeDataSet.py
m:\data\mastDataSet.txt had1 04

cd "M:\Data\all mock had1 hsv1 4 hip 2cm-1 032808\hsv1\all 1500-800"
m:\python25\python.exe  m:\data\makeDataSet.py
m:\data\mastDataSet.txt hsv1 04

cd "M:\Data\all mock had1 hsv1 4 hip 2cm-1 032808\mock\all 1500-800"
m:\python25\python.exe  m:\data\makeDataSet.py
m:\data\mastDataSet.txt mock 04

cd "M:\Data\all mock had1 hsv1 6 hip 2cm-1 032808-033108\had1\all
1500-800"
m:\python25\python.exe  m:\data\makeDataSet.py
m:\data\mastDataSet.txt had1 06

cd "M:\Data\all mock had1 hsv1 6 hip 2cm-1 032808-033108\hsv1\all
1500-800"
m:\python25\python.exe  m:\data\makeDataSet.py
m:\data\mastDataSet.txt hsv1 06

cd "M:\Data\all mock had1 hsv1 6 hip 2cm-1 032808-033108\mock\ALL
```

```
1500-800"
m:\python25\python.exe  m:\data\makeDataSet.py
m:\data\mastDataSet.txt mock 06

cd "M:\Data\all mock had1 hsv1 24hip 2cm-1 032408-041808\mock hsv1
had1 24hip 2cm-1 032408-040708\had1 24hip 2cm-1 032508-040708\all
1500-700"
m:\python25\python.exe  m:\data\makeDataSet.py
m:\data\mastDataSet.txt had1 24

cd "M:\Data\all mock had1 hsv1 24hip 2cm-1 032408-041808\mock hsv1
had1 24hip 2cm-1 032408-040708\hsv1 24hip 2cm-1 032508-040708\all
high hsv1 24hip 2cm-1 032408-040708"
m:\python25\python.exe  m:\data\makeDataSet.py
m:\data\mastDataSet.txt hsv1 24

cd "M:\Data\all mock had1 hsv1 24hip 2cm-1 032408-041808\mock hsv1
had1 24hip 2cm-1 032408-040708\mock 24hip 2cm-1 032508-040708\all
high mock 24hip 2cm-1 032408-040708"
m:\python25\python.exe  m:\data\makeDataSet.py
m:\data\mastDataSet.txt mock 24

cd "M:\Data\all mock had1 hsv1 24hip 2cm-1 032408-041808\mock hsv1
had1 24hpi 2cm-1 041708--041808\v-had1-human-t-nofix-24hpi-041708--
041808-2cm-1_2000-700 filter\all 1500-800"
m:\python25\python.exe  m:\data\makeDataSet.py
m:\data\mastDataSet.txt vhad1 24

cd "M:\Data\all mock had1 hsv1 24hip 2cm-1 032408-041808\mock hsv1
had1 24hpi 2cm-1 041708--041808\v-hsv1-human-t-nofix-24hpi-041808-
2cm-1_2000-700 filter\all 1500-800"
m:\python25\python.exe  m:\data\makeDataSet.py
m:\data\mastDataSet.txt vhsv1 24

cd "M:\Data\all mock had1 hsv1 24hip 2cm-1 032408-041808\mock hsv1
had1 24hpi 2cm-1 041708--041808\v-mock-human-t-nofix-24hpi-041608--
041708-2cm-1_2000-700 filter\all 1500-800"
m:\python25\python.exe  m:\data\makeDataSet.py
m:\data\mastDataSet.txt vmock 24

cd \data

echo "3 digit rounding of Spectrum"
echo "-----------------------------"
m:\python25\python.exe m:\data\JoinSlices3.py m:\data\mastDataSet.txt
```

```
m:\data\mastDataset3digit.txt

echo "Standardize DataSet"
echo "-------------------"
m:\python25\python.exe m:\data\makeStdData.py
m:\data\mastDataset3digit.txt m:\data\mastDataset3digitSTD.txt
```

APPENDIX C: PYTHON SCRIPT FOR PLOTTING RAW DATA

```
#This Python program, , extracts data from the
#master dataset by hour and by tissue type into subset files.
#It then modifies a SAS script using the corresponding data
#and tissue types to make the target SAS file which it then runs.
#The SAS script processes the data and GPLOTS it.  This program then
#renames the generate GIF file to a filename that indicates the
#type of data that was plotted.

import os,sys

def MakeDataFile(dataSetFileName,dataSubsetFileName,hour,tiss1):
    hr=0;file=1;v=2;tiss=3
    row=4;x=5;y=6;avg=7;std=8;stdy=9;
    disp=10;peak=11;fname=12

    fp=open(dataSetFileName,'r')
    fpout=open(dataSubsetFileName,'w')
    d1={}
    d2={}

    linenum=0
    while True:
        linenum += 1
        if linenum%10000 == 0:
            print 'reading line',linenum
        line = fp.readline()
        if len(line) == 0:
            break
        t=line.split()
        if t[hr] != hour:
            continue
        if t[tiss] not in (tiss1,'mock'):
            continue

        if t[tiss]==tiss1:
            if not d1.get(float(t[x])):
                d1[float(t[x])]={}
            d1[float(t[x])][t[int(file)]] = t[y]
        else:
            if not d2.get(float(t[x])):
                d2[float(t[x])]={}
```

```
            d2[float(t[x])][t[int(file)]] = t[y]
    fp.close()

    xlist = d1.keys()
    xlist.sort()
    tissCount=0
    mockCount=0
    for xval in xlist:
        print 'WRITING line',xval
        fpout.write(('%4.3f    '%xval)[0:10])
        cols1 = d1[xval].keys()
        cols2 = d2[xval].keys()
        cols1.sort()
        cols2.sort()
        oldTissCount=tissCount
        tissCount = len(cols1)
        if tissCount!=oldTissCount and oldTissCount!=0:
            print "Warning",tiss1," count of",tissCount,"does not
match old count of",oldTissCount,"for hour",hour
        oldMockCount=mockCount
        mockCount = len(cols2)
        if mockCount!=oldMockCount and oldMockCount!=0:
            print "Warning mock count of",mockCount,"does not match
old count of",oldMockCount,"for hour",hour
        for col in cols1:
            fpout.write((d1[xval][col]+'    ')[0:11])
        fpout.write(' '*11)
        for col in cols2:
            fpout.write((d2[xval][col]+'    ')[0:11])
        fpout.write('\n')

    fpout.close()
    return tissCount,mockCount

def WriteSasFiles(hour,tiss,tissCount,mockCount):
    fTmpl = open('RawDataPlot.sas','r')
    buf = fTmpl.read()
    fTmpl.close()
    varOffsets = ''
    varList = ''
    for i in range(0,tissCount):
        varOffsets += 'y'+str(i)+' '+str(11+i*11)+'-'+str(11+i*11+8)+'
'
        varList += 'y'+str(i)+'*x=1 '
        if i%10 == 9:
```

```
            varOffsets += '\n' + ' '*10
            varList += '\n'+' '*9
    varOffsets += '\n'+' '*10
    varList += '\n'+' '*9
    for j in range(0,mockCount):
        varOffsets += 'y'+str(j+tissCount)+' '+str(11+
(j+tissCount)*11)+'-'+str(11+(j+tissCount)*11+8)+'  '
        varList += 'y'+str(j+tissCount)+'*x=2 '
        if j%10 == 9:
            varOffsets += '\n' + ' '*10
            varList += '\n'+' '*9

    recLen = 10 + tissCount*11 + 11 + mockCount*11 + 1
    newBuf = buf.replace("@HR", hour).replace("@TISS",
tiss).replace('@RECLEN',str(recLen))
    finalBuf = newBuf.replace('@VAROFFSETS',
varOffsets).replace('@VARLIST', varList)
    fSas = open('RawDataPlot'+hour+tiss+'.sas','w')
    fSas.write(finalBuf)
    fSas.close()

def RunSasFiles(hour,tiss):
    os.system('"M:\Program Files\SAS\SAS 9.1\sas.exe"
RawDataPlot'+hour+tiss+'.sas')
    os.system('copy gplot.gif RawDataPlot'+hour+tiss+'.gif')
    os.system('copy sashtml.htm RawDataPlot'+hour+tiss+'html.htm')


def main():
    for hour in ['02','04','06','24']:
        for tiss in ['had1','hsv1']:
            tissCount,mockCount = MakeDataFile
("mastDataset3digitSTD.txt", "RawDataSubSet"+hour+tiss+".dat", hour,
tiss)
            WriteSasFiles(hour,tiss,tissCount,mockCount)
            RunSasFiles(hour,tiss)

main()
```

APPENDIX D: SAS TEMPLATE FOR PLOTTING RAW DATA

```
/*SAS template program for plotting raw data of a tissue type
  contrasted against mock for a particular hour */
goptions device=gif reset=global gunit=pct border cback=white
         colors=(black blue green red)
         ctext=black
         ftitle=swissb ftext=swiss htitle=4 htext=3;

filename logfile 'm:\data\RawDataPlot@HR@TISS.log';
filename output  'm:\data\RawDataPlot@HR@TISS.lst';
proc printto  log=logfile print=output new;
run;

data std@HR@TISS;
    infile 'm:\data\RawDataSubSet@HR@TISS.dat'  lrecl=@RECLEN;
    input x 1-8
          @VAROFFSETS;
run;

ods html;
ods graphics on;

/*
title1 height=5 'Raw data';
title2 height=4 'Hour @HR @TISS vs Mock';
*/
footnote1 c=red f=swissb h=3 'red = @TISS   '
          c=blue f=swissb h=3 '  blue = mock';
/*footnote2 j=c c=green ' Each line is raw data, not standardized';
*/

symbol1 color=red interpol=join;
symbol2 color=blue interpol=join;

axis1 label=('Spectrum')
       order = 800 to 1501 by 100
       offset=(3)
       width=3;
axis2 label=('Absorption')
      order=(-0.4 to 1.6 by 0.2)
      width=3;
```

```
proc gplot data=std@HR@TISS;
    plot @VARLIST
         / overlay
         haxis=axis1 hminor=1
         vaxis=axis2 vminor=1
         caxis=black;
run;

ods graphics off;
ods html close;

quit;
```

APPENDIX E: PYTHON CODE FOR

REMOVING SIXTH DECIMAL POINT VARIATION

```
#This code, JoinSlices3.py, unites spectrum frequency values that
#were measured to5 decimal points with other samples that had spectrum
#frequencies recorded to 6 decimal positions but which are the
#same point minus this miniscule roundoff difference.
#
#This prevents interesting plotting problems were the small set
#of samples with 6 decimal places are not evenly balanced with
#other data.

import os,sys

fp=open(sys.argv[1],'r')

ds={}
hour=0
file=1
v=2
tiss=3
row=4
x=5
y=6
disp=7
peak=8
fname=9

line = fp.readline()
while len(line) > 0:
    t=line.split()
    if ds.get('%04.1f'%float(t[x])):
        if t[x] not in ds['%04.1f'%float(t[x])]:
            ds['%04.1f'%float(t[x])].append(t[x])
            if len(t[x]) > len(ds['%04.1f'%float(t[x])][0]):
                ds['%04.1f'%float(t[x])].reverse()
    else:
        ds['%04.1f'%float(t[x])] = [t[x]]
    line = fp.readline()

fp.close()
```

```
fp=open(sys.argv[1],'r')
fpout=open(sys.argv[2],'w')

line = fp.readline()
while len(line) > 0:
    t=line.split()
    t[x] = ds['%04.1f'%float(t[x])][0]
    temp = '%04.3f' % float(t[x])

    out = line[0:23]
    out += ('%-12.12s' % temp + '          ')[0:13]
    out += line[36:]
    fpout.write(out)
    line = fp.readline()
fp.close()
fpout.close()
```

APPENDIX F: PYTHON CODE FOR STANDARDIZING ABSORPTION DATA

```
#Python program, makeStdData.py, for standardizing the
#absorption values in the consolidated master data set
import os,sys,math

if len(sys.argv) == 1:
    print "Syntax:  makeStdData  srcfile  destfile"
    sys.exit(1)
srcfile=sys.argv[1]
destfile=sys.argv[2]
fp=open(srcfile,'r')
fpout=open(destfile,'w')
d={}
hr=0;file=1;v=2;tiss=3;row=4
x=5;y=6;disp=7;peak=8;fname=9;norm=10
#status values
readdata=1
calcvals=2
last=3
done=4
status=readdata
linenum=0
oldrec=0
sum=0.0
while status != done:
    if status == readdata:
        line = fp.readline()
        if len(line) <= 2:
            status = last
            fp.close()
        else:
            linenum += 1
            if linenum%10000 == 0:
                print 'read line',linenum
            t=line.split()
            if len(t)==11:
                t[9]=t[9]+' '+t[10]
                t[10]=0
            elif len(t)==10:
                t.append('0')
            r=int(t[row])
            if oldrec+1 <= r:
```

```
                d[r] = t
                oldrec = r
            else:
                status = calcvals
    elif status==calcvals or status==last:
        sum = 0.0
        for i in range(1,oldrec+1):
            sum += float(d[i][y])
        avg = sum/oldrec;
        var=0.0
        for i in range(1,oldrec+1):
            diff = float(d[i][y]) - avg
            var += diff*diff/(oldrec-1)
        stddev=math.sqrt(var)
        for i in range(1,oldrec+1):
            #testing
            #if linenum > 508000:
            #    print d[i]
            d[i][norm] = (float(d[i][y])-avg)/stddev
        for i in range(1,oldrec+1):
            out  = d[i][hr]+'  '
            out += d[i][file]+'  '
            out += d[i][v]+'  '
            out += d[i][tiss]+'  '
            out += d[i][row]+'  '
            out += ('%-9.9s'%d[i][x] + '      ')[0:10]
            out += ('%-12.12s'%d[i][y] + '      ')[0:10]
            out += '%01.8f  '%avg
            out += '%01.8f  '%stddev
            out += ('%02.8f    '%d[i][norm])[0:13]
            out += (d[i][disp] + ' '*7)[0:7]
            out += (d[i][disp] + ' '*7)[0:7]
            out += (d[i][fname] + ' '*75)[0:75]
            fpout.write(out+'\n')
        if status==calcvals:
            sum = float(t[y])
            d={r:t}
            oldrec=r
            status = readdata
        elif status==last:
            status=done
            fpout.close()
print "Done!!!"
```

APPENDIX G: PYTHON SCRIPT TO HELP PLOT STANDARDIZED DATA

```
#program name: StdDataPlotsMake.py
#This Python script filters the standardized absorption by tissue
#and hour group, modifies a template of a SAS script, runs the
#modified SAS code to plot the data.  It then renames the
#generated GIF file appropriately.

import os,sys

def MakeDataFile(dataSetFileName, dataSubsetFileName, hour, tiss1):
    hr=0;file=1;v=2;tiss=3;row=4
    x=5;y=6;avg=7;std=8;stdy=9;disp=10;peak=11;fname=12

    fp=open(dataSetFileName,'r')
    fpout=open(dataSubsetFileName,'w')
    d1={}
    d2={}
    linenum=0
    while True:
        linenum += 1
        if linenum%10000 == 0:
            print 'reading line',linenum
        line = fp.readline()
        if len(line) == 0:
            break
        t=line.split()
        if t[hr] != hour:
            continue
        if t[tiss] not in (tiss1,'mock'):
            continue

        if t[tiss]==tiss1:
            if not d1.get(float(t[x])):
                d1[float(t[x])]={}
            d1[float(t[x])][t[int(file)]] = t[stdy]
        else:
            if not d2.get(float(t[x])):
                d2[float(t[x])]={}
            d2[float(t[x])][t[int(file)]] = t[stdy]
    fp.close()

    xlist = d1.keys()
```

```
    xlist.sort()
    tissCount=0
    mockCount=0
    for xval in xlist:
        print 'WRITING line',xval
        fpout.write(('%4.3f    '%xval)[0:10])
        cols1 = d1[xval].keys()
        cols2 = d2[xval].keys()
        cols1.sort()
        cols2.sort()
        oldTissCount=tissCount
        tissCount = len(cols1)
        if tissCount!=oldTissCount and oldTissCount!=0:
            print "Warning",tiss1," count of",tissCount,"does not
match old count of",oldTissCount,"for hour",hour
        oldMockCount=mockCount
        mockCount = len(cols2)
        if mockCount!=oldMockCount and oldMockCount!=0:
            print "Warning mock count of",mockCount,"does not match
old count of",oldMockCount,"for hour",hour
        for col in cols1:
            fpout.write((d1[xval][col]+'    ')[0:13])
        fpout.write(' '*13)
        for col in cols2:
            fpout.write((d2[xval][col]+'    ')[0:13])
        fpout.write('\n')

    fpout.close()
    return tissCount,mockCount

def WriteSasFiles(hour,tiss,tissCount,mockCount):
    fTmpl = open('StdDataPlot.sas','r')
    buf = fTmpl.read()
    fTmpl.close()
    varOffsets = ''
    varList = ''
    for i in range(0,tissCount):
        varOffsets += 'y'+str(i)+'
'+str(11+i*13)+'-'+str(11+i*13+10)+'  '
        varList += 'y'+str(i)+'*x=1 '
        if i%10 == 9:
            varOffsets += '\n' + ' '*10
            varList += '\n'+' '*9
    varOffsets += '\n'+' '*10
    varList += '\n'+' '*9
```

```
    for j in range(0,mockCount):
        varOffsets += 'y'+str(j+tissCount)+' '+str(11+
(j+tissCount)*13)+'-'+str(11+(j+tissCount)*13+10)+'  '
        varList += 'y'+str(j+tissCount)+'*x=2 '
        if j%10 == 9:
            varOffsets += '\n' + ' '*10
            varList += '\n'+' '*9

    recLen = 10 + tissCount*13 + 13 + mockCount*13 + 1
    newBuf =
buf.replace("@HR",hour).replace("@TISS",tiss).replace('@RECLEN',str(re
cLen))
    finalBuf =
newBuf.replace('@VAROFFSETS',varOffsets).replace('@VARLIST',varList)
    fSas = open('StdDataPlot'+hour+tiss+'.sas','w')
    fSas.write(finalBuf)
    fSas.close()

def RunSasFiles(hour,tiss):
    os.system('"M:\Program Files\SAS\SAS 9.1\sas.exe"
StdDataPlot'+hour+tiss+'.sas')
    os.system('copy gplot.gif StdDataPlot'+hour+tiss+'.gif')
    os.system('copy sashtml.htm StdDataPlot'+hour+tiss+'html.htm')

def main():
    for hour in ['02','04','06','24']:
        for tiss in ['had1','hsv1']:
            tissCount,mockCount = MakeDataFile
("mastDataset3digitSTD.txt", "StdDataSubSet"+hour+tiss+".dat", hour,
tiss)
            WriteSasFiles(hour,tiss,tissCount,mockCount)
            RunSasFiles(hour,tiss)
main()
```

APPENDIX H: SAS TEMPLATE TO PLOT STANDARDIZED DATA

```
/* Template, StdDataPlot.sas, of SAS script to help plot
   standardized data given the tissue type and hour group. */
goptions device=gif reset=global gunit=pct border cback=white
         colors=(black blue green red)
         ctext=black
         ftitle=swissb ftext=swiss htitle=4 htext=3;

filename logfile 'm:\data\StdDataPlot@HR@TISS.log';
filename output  'm:\data\StdDataPlot@HR@TISS.lst';
proc printto  log=logfile print=output new;
run;

data std@HR@TISS;
    infile 'm:\data\StdDataSubSet@HR@TISS.dat'  lrecl=@RECLEN;
    input x 1-8
          @VAROFFSETS;
run;

ods html;
ods graphics on;

/*
title1 height=5 'Standardized raw data';
title2 height=4 'Hour @HR @TISS vs Mock';
*/

footnote1 c=red f=swissb h=3 'red = @TISS   '
          c=blue f=swissb h=3 '  blue = mock';
/*
footnote2 j=c c=green ' Each line is standardized with mu=0 var=1';
*/

symbol1 color=red interpol=join;
symbol2 color=blue interpol=join;

axis1 label=('Spectrum')
       order = 800 to 1501 by 100
       offset=(3)
       width=3;
axis2 label=('Absorption')
      order=(-2.5 to 2.5 by 0.25)
```

```
      width=3;

proc gplot data=std@HR@TISS;
    plot @VARLIST
         / overlay
         haxis=axis1 hminor=1
         vaxis=axis2 vminor=1
         caxis=black;
run;

ods graphics off;
ods html close;

quit;
```

APPENDIX I: PYTHON SCRIPT TO CALCULATE

WILCOXON-MANN-WHITNEY Z SCORE PLOTS

```
#Python script, CalcWmwZ.py, to generate SAS script to calculate
#Wilcoxon-Mann-Whitney statistic's Z score in the SAS
#listing file and then to call a script to extract those
#scores to csv files based on hour and tissue.

import os,sys

def calc(hour,tiss1,tiss2):
    fp=open("CalcWmwZ.sas","r")
    buf=fp.read()
    fp.close()
    newbuf=buf.replace('@1',hour).replace('@2',tiss1).replace('@3',ti
ss2)
    fpout=open('CalcWmwZ'+hour+tiss1+tiss2+'.sas', 'w')
    fpout.write(newbuf)
    fpout.close()
    os.system('"M:\Program Files\SAS\SAS 9.1\sas.exe"
CalcWmwZ'+hour+tiss1+tiss2+'.sas')
    os.system('M:\Python25\python.exe M:\data\makeplotWmwZs.py '
              +'M:\data\WmwZ'+hour+tiss1+tiss2+'.lst '
              +'M:\data\WmwZ'+hour+tiss1+tiss2+'.csv')

for x in ('02','04','06','24'):
    for y in ('hsv1','had1'):
        print 'running data for',x,y,'mock'
        calc(x,y,'mock')
```

APPENDIX J: SAS TEMPLATE SCRIPT TO GENERATE

WILCOXON-MANN-WHITNEY Z SCORES

```
/*SAS template script,CalcWmwZ.sas, to calculate
  the WMW statistics Z score*/
filename logfile 'm:\data\WmwZ@1@2@3.log';
filename output 'm:\data\WmwZ@1@2@3.lst';
proc printto  log=logfile print=output new;
run;

data masterset;
    infile 'm:\data\mastDataset3digitSTD.txt';
    input hours 1-2 srcfilenum 5-7 vtype $ 10 tissue $ 13-16 linenum
19-21
          spectrum 24-31 absorbtion 34-41 avg 44-53 stdv 59-65
absorbstd 68-78
          displaydirection 81-85 peakdirection 88-92 srcfile $ 95-
160;
run;

title "Remaining Data excluding outlier";
data subset1; set masterset;
    if hours=@1 and (tissue='@2' or tissue='@3');
run;

proc sort;
    by spectrum;
run;

proc npar1way wilcoxon;
    class tissue;
    var absorbstd;
    by spectrum;
run;

proc printto; run;
```

APPENDIX K: PYTHON SCRIPT TO EXTRACT WILCOXON-MANN-WHITNEY Z SCORE TO A CSV FILE

```
#Python script, makeplotWmwZs.py, to extract the
#Wilcoxon-Mann-Whitney Z Score statistic to a csv file for plotting.
import os,sys

if len(sys.argv) != 3:
    print "Syntax:  "+os.path.basename(sys.argv[0])+'  infile
outfile'
    sys.exit(1)

fpin=open(sys.argv[1],'r')
fpout=open(sys.argv[2],'w')

spec=1
specstr='spectrum='
z=2
zstr='Z                      '

state=spec
line = fpin.readline()
while len(line) > 0:
    if state==spec:
        offset = line.find(specstr)
        if offset > -1:
            specval = line[(offset+len(specstr)):].split(' ')[0]
            state=z
    elif state==z:
        offset = line.find(zstr)
        if offset > -1:
            zval = line[(offset+len(zstr)):].strip(' ').split(' ')
[0].strip('\n\r ')
            state=spec
            fpout.write(specval+','+zval+'\n')
    line = fpin.readline()

fpin.close()
fpout.close()
```

APPENDIX L: PYTHON PROGRAM TO GENERATE PLS MODEL AND CALL LOGISTIC REGRESSION TO GENERATE ROC CURVE

```
#Python program, PlsNoCvMake.py, to reduce via 5 point kernel
#smoothing fuction, the number of variables, run PLS to
#generate a model, and run Logistic to generate an ROC curve

import os,sys,time

FNprefix = 'PlsNoCv'

#indexes of line elements
hr=0 #hour index
file=1
v=2
tiss=3
row=4
xIdx=5
yIdx=6
avg=7
dev=8
yStd=9

#global debug variable
debug=False


def Getz5set (fz, clump, zlimit):
    eof=False
    z5set=[]
    zline = fz.readline()
    idx = 0
    while (len(zline) > 0) and (len(z5set) < clump):
        x,z=zline.split(',')
        x="%3.3f" % float(x)
        z=float(z)
        if idx%clump==0 or len(z5set)==idx%clump:
            if abs(z) > zlimit:
                z5set.append(x)
            else:
                z5set=[]
        else:
```

```
            z5set=[]
        idx += 1
        if len(z5set) < clump:
            zline = fz.readline()

    if len(zline) == 0:
        eof = True
    if debug: print eof,z5set
    return eof,z5set


def GetZset (zFileName, clump, zlimit):
    zset = []
    fz=open(zFileName,'r')  #Z score file
    if debug: print "Reading Zfile:",zFileName
    eof=False
    zsetCount = 0
    while not eof:
        eof,z5set = Getz5set(fz, clump, zlimit)
        if len(z5set)==clump:
            zset += z5set
            zsetCount += 1
    fz.close()
    ft=open(FNprefix+'ZsetCounts.txt','a+')
    msg=("Filtering standardized dataset using "+zFileName+" where Z >
"+str(zlimit)+" gives "+str(zsetCount)+
             " variables.\nEach variable is the average absorption for
a cluster of "+str(clump)+" adjacent points\n")
    ft.write(msg)
    print msg,
    ft.close()
    if debug: print "\nEntire ZSET:",zset
    return zset, zsetCount


def clearOldFiles():
    f=open(FNprefix+'ZsetCounts.txt','w')
    f.close()
    for hour in ['02','04','06','24']:
        for tissue in ['hsv1']:
            f=open(FNprefix+'5set'+hour+tissue+'dataset.dat','w')
            f.close()


def writeData (firstLine, xset, yset, idx, clump, t, fout, tissCount,
```

```
mockCount):

    if debug: print "Average x and y",xset, yset
    #For now, instead of using median, use mean since we are taking
the mean of the Y values, taking the mean of the
    #X values seems to make more sense
    #avgx = xset[clump//2] #get median x in clump (choose an odd (not
even) length for clump size for best results)
    avgx = sum(xset)/(1.0*len(xset))
    avgy = sum(yset)/(1.0*len(yset))
    buf=""
    if idx//clump == 0:
        if not firstLine:
            buf =  "\n"
        #write data before first "clumped" variable
        buf += "%2s  %3s  %4s  %3s  " % (t[hr],t[file],t[tiss],t[row])
#t[row] is the ending row
        if t[tiss]=='mock':
            buf += "0  "
        else:
            buf += "1  "
        if t[tiss]=='mock':
            mockCount[0] += 1
        else:
            tissCount[0] +=1

    #write the clumped variables frequency
    buf += ("%4.3f    " % avgx)[0:10]
    #write the clumped variable's absorption
    buf += ("%1.8f     " % avgy)[0:14]
    fout.write(buf)
    if debug: print "Writing:",buf


def CreatePlsSasFile(hour,tissue,variables):
    filteredFilename = FNprefix+'5set'+hour+tissue+'dataset.dat'
    if hour=='24':
        sasFname = FNprefix+'24'+'.sas'
    else:
        sasFname = FNprefix+'XX'+'.sas'
    fTemplate = open(sasFname,'r')
    template=fTemplate.read()
    fTemplate.close()
    varList = []
    varOffsetStr=''
```

```
    varNameStr=''
    for i in range(0,variables):
        offset=34+24*i
        offsetStr=str(offset)+'-'+str(offset+10)
        varName='v'+str(i)
        varNameStr += varName+' '
        varOffsetStr += varName+' '+offsetStr+'  '
        if i%10==9:
            varOffsetStr += '\n'
            varNameStr += '\n'
    newTemplate =
template.replace('@HR',hour).replace('@TISS',tissue).replace('@RECLEN'
,str(offset+11+5))
    finalText =
newTemplate.replace('@VarOffsets',varOffsetStr).replace('@Vars',varNam
eStr)
    f=open(FNprefix+hour+tissue+'.sas','w')
    f.write(finalText)
    f.close()


def RunPls(hour,tissue):
    os.system('"M:\Program Files\SAS\SAS 9.1\sas.exe"
'+FNprefix+hour+tissue+'.sas')
    #os.system("copy ROCCurve0.gif
PlsLogis"+hour+tissue+"ROCCurve0.gif")
    #os.system("copy ROCOverlay1.gif
PlsLogis"+hour+tissue+"ROCOverlay1.gif")
    #os.system("copy sashtml.htm  PlsLogis"+hour+tissue+"sashtml.htm")
    print ' '


def FilterDataUsingZFile (hour, tissue, clump, zlimit,
dataSourceFile):
    zset,variables = GetZset('WmwZ24'+tissue+'mock.csv', clump,
zlimit)
    if debug:
        print "Got Zset.  sleeping 10"
        time.sleep(2)
    zsetLen=len(zset) #total number of significant dots within
significant clumps
    idx = 0
    xset = []
    yset = []
    fd=open(dataSourceFile,'r') #dataset file
```

```
    filteredFilename=FNprefix+'5set'+hour+tissue+'dataset.dat'
    if debug: print "Writing to filtered dataset
file:",filteredFilename
    fout=open(filteredFilename,'w')
    firstLine=True
    tissCount = [0]
    mockCount = [0]
    line=fd.readline()
    while len(line)>2:
        t=line.split() #get tuple t from line
        if t[hr]==hour and t[tiss] in [tissue,'mock'] and t[xIdx] in
zset:
            xset.append(float(t[xIdx]))
            yset.append(float(t[yStd]))
            if idx%clump == (clump - 1):
                writeData(firstLine, xset, yset, idx, clump, t, fout,
tissCount, mockCount)
                firstLine=False
                xset=[]
                yset=[]
            idx+=1
        if idx==zsetLen: #reached end of a the significant dots within
one sample
            idx=0
        line=fd.readline()

    fc=open(FNprefix+'ZsetCounts.txt','a+')
    text = ('tissue samples:'+str(tissCount)+'  mock
samples:'+str(mockCount)+'\n' +
            'samples per variable from smaller sample = ' +
str(1.0*min(tissCount[0],mockCount[0])/(variables+0.000001)) +'\n' )
    fc.write(text)
    print text
    fc.close()
    fd.close()
    fout.close()
    CreatePlsSasFile(hour,tissue,variables)  ##SAS code


def main():
    #Setup
    global debug
    StandardizedDataSet = 'mastDataset3digitSTD.txt'
    offset=0
    if len(sys.argv)>1 and 'help' in sys.argv:
```

```
        print "Syntax:",sys.argv[0]," [clump  z_02had1 z_02hsv1
z_04had1 z_04hsv1 z_06had1 z_06hsv1 z_24had1 z_24hsv1]"
        return 1
    if len(sys.argv)>1 and sys.argv[1]=='debug':
        debug=True
        offset=1
        print "Reading from standardized dataset
file:",StandardizedDataSet
    if len(sys.argv)==1:
        clump = 5
        zLimitList = [8,8,  8,8,  8,8,  8,8]
    else:
        clump=int(sys.argv[1])
        zLimitList = []
        for i in range(offset+2, offset+len(sys.argv)):
            zLimitList.append(float(sys.argv[i]))
    #Run
    clearOldFiles()
    i=0
    for hour in ['02','04','06','24']:
        for tissue in ['hsv1']:
            print "making",hour,tissue,"with Z =",zLimitList[i]
            FilterDataUsingZFile(hour, tissue, clump, zLimitList[i],
StandardizedDataSet)
            i+=1
    i=0
    for hour in ['02','04','06','24']:
        for tissue in ['hsv1']:
            print "running",hour,tissue,"with Z =",zLimitList[i]
            RunPls(hour,tissue)
            i+=1

main()
```

APPENDIX M: TWO SAS TEMPLATES TO GENERATE PLS MODEL AND CALL LOGISTIC REGRESSION TO GENERATE ROC CURVE

```
/*SAS template, PlsNoCv24.sas, to perform PLS modeling and
  ROC curve creation via proc Logistic for 24 hour data, out
  of 2,4,6,24 dataset*/
filename grafout 'M:\Data\graphs\@HR@TISS\';
goptions device=gif gsfname=grafout gsfmode=replace;

filename logfile 'm:\data\PlsNoCv@HR@TISS.log';
filename output  'm:\data\PlsNoCv@HR@TISS.lst';
proc printto  log=logfile print=output new;
run;

data sourceset;
    infile 'm:\data\PlsNoCv5set@HR@TISSdataset.dat' lrecl=@RECLEN;
    input hours 1-2 srcfilenum 5-7 tissue $ 10-13 linenum 16-18
disease 21
          @VarOffsets;
run;

ods listing close;
ods output
    CenScaleParms = Cen_Scale_Parms_@HR@TISS
    PercentVariation = Pct_Vari_@HR@TISS
    XWeights = X_Wts_@HR@TISS;


proc pls data=sourceset details nfac=6;
    model disease = @Vars
          / solution;
    output out=outpls predicted=ypred xscore=xscr yscore=yscr;
run;

ods output close;
ods listing;

proc print data=Pct_Vari_@HR@TISS;
run;

proc sort data=Cen_Scale_Parms_@HR@TISS;
    by disease;
```

```
run;
proc print;
run;

proc print data=X_Wts_@HR@TISS;
run;

proc print data=outpls; run;

proc logistic data=outpls;   /*outest=betas;*/
      model disease (event='1') = ypred
             / outroc=roc@HR@TISS ;
run;

/*title2 'Parameter Estimates at hour @HR for @TISS vs mock';
  proc print data=betas;run;
*/

title2 'Complete ROC data printout at hour @HR for @TISS vs mock';
proc print data=roc@HR@TISS;
run;


symbol1 interpol=join color=blue;

proc gplot data=roc@HR@TISS;
    plot _SENSIT_*_1MSPEC_;
    run;


proc printto;
run;
quit;
```

```
/*SAS template, PlsNoCvXX.sas, to perform PLS modeling
and ROC curve creation via proc Logistic for 2,4,6 hour data,
out of 2,4,6,24 dataset*/
filename grafout 'M:\Data\graphs\@HR@TISS\';
goptions device=gif gsfname=grafout gsfmode=replace;

filename logfile 'm:\data\PlsNoCv@HR@TISS.log';
filename output  'm:\data\PlsNoCv@HR@TISS.lst';
proc printto  log=logfile print=output new;
run;

title 'PLS Model at hour @HR for @TISS vs mock';
data sourceset24;
    infile 'm:\data\PlsNoCv5set24@TISSdataset.dat' lrecl=@RECLEN;
    input hours 1-2 srcfilenum 5-7 tissue $ 10-13 linenum 16-18
disease 21
          @VarOffsets;
run;

data new24;
    set sourceset24; diseasecopy=disease;
run;

data sourceset@HR;
    infile 'm:\data\PlsNoCv5set@HR@TISSdataset.dat' lrecl=@RECLEN;
    input hours 1-2 srcfilenum 5-7 tissue $ 10-13 linenum 16-18
disease 21
          @VarOffsets;
run;

data new@HR;
    set sourceset@HR; diseasecopy=disease;
run;

data whole; set new24 new@HR(drop=disease);run;

ods listing close;
ods output
    CenScaleParms = Cen_Scale_Parms_@HR@TISS
    PercentVariation = Pct_Vari_@HR@TISS
    XWeights = X_Wts_@HR@TISS;

proc pls data=whole details nfac=6;
    model disease = @Vars
          / solution;
```

```
    output out=outpls predicted=ypred xscore=xscr yscore=yscr;
run;

data just@HR; set outpls;
    if hours=@HR; disease=diseasecopy;
run;

ods output close;
ods listing;

proc print data=Pct_Vari_@HR@TISS;
run;

proc sort data=Cen_Scale_Parms_@HR@TISS;
    by disease;
run;
proc print;
run;

proc print data=X_Wts_@HR@TISS;
run;

proc print data=just@HR; run;

proc logistic data=just@HR;   /*outest=betas;*/
      model disease (event='1') = ypred
             / outroc=roc@HR@TISS ;
run;

/*title2 'Parameter Estimates at hour @HR for @TISS vs mock';
  proc print data=betas;run;
*/
title2 'Complete ROC data printout at hour @HR for @TISS vs mock';
proc print data=roc@HR@TISS;
run;

symbol1 interpol=join color=blue;
proc gplot data=roc@HR@TISS;
    plot _SENSIT_*_1MSPEC_;
    run;
proc printto;
run;
quit;
```

APPENDIX N: PYTHON PROGRAM TO RUN KOLMOGOROV-SMIRNOV TEST AGAINST 24 HOUR VERSUS 2, 4, 6 HOUR DATA OF SAME DISEASE STATUS

```
#Python program, KsPlsMake.py, to perform Kolmogorov-Smirnov test
#on 24 hour versus 2, 4, and 6 hour mock data to
#verify whether mock data belongs to same population or not
#also will run against hsv1 data.
import os,sys,time

FNprefix = 'KsPls'

#indexes of line elements
hr=0 #hour index
file=1
v=2
tiss=3
row=4
xIdx=5
yIdx=6
avg=7
dev=8
yStd=9

#global debug variable
debug=False


def Getz5set (fz, clump, zlimit):
    eof=False
    z5set=[]
    zline = fz.readline()
    idx = 0
    while (len(zline) > 0) and (len(z5set) < clump):
        x,z=zline.split(',')
        x="%3.3f" % float(x)
        z=float(z)
        if idx%clump==0 or len(z5set)==idx%clump:
            if abs(z) > zlimit:
                z5set.append(x)
            else:
                z5set=[]
        else:
```

```
            z5set=[]
        idx += 1
        if len(z5set) < clump:
            zline = fz.readline()

    if len(zline) == 0:
        eof = True
    if debug: print eof,z5set
    return eof,z5set


def GetZset (zFileName, clump, zlimit):
    zset = []
    fz=open(zFileName,'r')  #Z score file
    if debug: print "Reading Zfile:",zFileName
    eof=False
    zsetCount = 0
    while not eof:
        eof,z5set = Getz5set(fz, clump, zlimit)
        if len(z5set)==clump:
            zset += z5set
            zsetCount += 1
    fz.close()
    ft=open(FNprefix+'ZsetCounts.txt','a+')
    msg=("Filtering standardized dataset using "+zFileName+" where Z >
"+str(zlimit)+" gives "+str(zsetCount)+
             " variables.\nEach variable is the average absorption for
a cluster of "+str(clump)+" adjacent points\n")
    ft.write(msg)
    print msg,
    ft.close()
    if debug: print "\nEntire ZSET:",zset
    return zset, zsetCount


def clearOldFiles():
    f=open(FNprefix+'ZsetCounts.txt','w')
    f.close()
    for hour in ['02','04','06']:
        for tissue in ['hsv1','mock']:
            f=open(FNprefix+'5set24'+hour+tissue+'dataset.dat','w')
            f.close()


def writeData (firstLine, xset, yset, idx, clump, t, fout, count24,
```

```
counthr):

    if debug: print "Average x and y",xset, yset
    #For now, instead of using median, use mean since we are taking
the mean of the Y values, taking the mean of the
    #X values seems to make more sense
    #avgx = xset[clump//2] #get median x in clump (choose an odd (not
even) length for clump size for best results)
    avgx = sum(xset)/(1.0*len(xset))
    avgy = sum(yset)/(1.0*len(yset))
    buf=""
    if idx//clump == 0:
        if not firstLine:
            buf =  "\n"
        #write data before first "clumped" variable
        buf += "%2s  %3s  %4s  %3s  " % (t[hr],t[file],t[tiss],t[row])
#t[row] is the ending row
        #if t[tiss]=='mock':
        #    buf += "0  "
        #else:
        #    buf += "1  "
        if t[hr]=='24':
            count24[0] += 1
            buf += "1  "
        else:
            counthr[0] +=1
            buf += "0  "

    #write the clumped variables frequency
    buf += ("%4.3f    " % avgx)[0:10]
    #write the clumped variable's absorption
    buf += ("%1.8f     " % avgy)[0:14]
    fout.write(buf)
    if debug: print "Writing:",buf


def CreatePlsSasFile(hour,tissue,variables):
    fTemplate = open(FNprefix+'.sas','r')
    template=fTemplate.read()
    fTemplate.close()
    varList = []
    varOffsetStr=''
    varNameStr=''
    for i in range(0,variables):
        offset=34+24*i
```

```
        offsetStr=str(offset)+'-'+str(offset+10)
        varName='v'+str(i)
        varNameStr += varName+' '
        varOffsetStr += varName+' '+offsetStr+'  '
        if i%10==9:
            varOffsetStr += '\n'
            varNameStr += '\n'
    newTemplate =
template.replace('@HR',hour).replace('@TISS',tissue).replace('@RECLEN'
,str(offset+11+5))
    finalText =
newTemplate.replace('@VarOffsets',varOffsetStr).replace('@Vars',varNam
eStr)
    f=open(FNprefix+'24'+hour+tissue+'.sas','w')
    f.write(finalText)
    f.close()


def RunPls(hour,tissue):
    os.system('"M:\Program Files\SAS\SAS 9.1\sas.exe"
'+FNprefix+'24'+hour+tissue+'.sas')
    #os.system("copy ROCCurve0.gif
StepLogis"+hour+tissue+"ROCCurve0.gif")
    #os.system("copy ROCOverlay1.gif
StepLogis"+hour+tissue+"ROCOverlay1.gif")
    #os.system("copy sashtml.htm
StepLogis"+hour+tissue+"sashtml.htm")
    print ' '


def FilterDataUsingZFile (hour, tissue, clump, zlimit,
dataSourceFile):
    zset,variables = GetZset('WmwZ24hsv1mock.csv', clump, zlimit)
    if debug:
        print "Got Zset.  sleeping 10"
        time.sleep(2)
    zsetLen=len(zset) #total number of significant dots within
significant clumps
    idx = 0
    xset = []
    yset = []
    fd=open(dataSourceFile,'r') #dataset file
    filteredFilename=FNprefix+'5set24'+hour+tissue+'dataset.dat'
    if debug: print "Writing to filtered dataset
file:",filteredFilename
```

```
    fout=open(filteredFilename,'w')
    firstLine=True
    count24 = [0]
    counthr = [0]
    line=fd.readline()
    while len(line)>2:
        t=line.split() #get tuple t from line
        if t[hr] in [hour,'24'] and t[tiss] == tissue and t[xIdx] in
zset and t[v]=='n':
            xset.append(float(t[xIdx]))
            yset.append(float(t[yStd]))
            if idx%clump == (clump - 1):
                writeData(firstLine, xset, yset, idx, clump, t, fout,
count24, counthr)
                firstLine=False
                xset=[]
                yset=[]
            idx+=1
        if idx==zsetLen: #reached end of the significant dots within
one sample
            idx=0
        line=fd.readline()

    fc=open(FNprefix+'ZsetCounts.txt','a+')
    text = ('tissue:'+tissue + '  hour 24 samples:'+str(count24)+'
hour '+hour+' samples:'+str(counthr)+'\n' +
            'samples per variable from smaller sample = ' +
str(1.0*min(counthr[0],count24[0])/(variables+0.000001)) +'\n' )
    fc.write(text)
    print text
    fc.close()
    fd.close()
    fout.close()
    CreatePlsSasFile(hour,tissue,variables)  ##SAS code


def main():
    #Setup
    global debug
    StandardizedDataSet = 'mastDataset3digitSTD.txt'
    offset=0
    if len(sys.argv)>1 and 'help' in sys.argv:
        print "Syntax:",sys.argv[0]," [clump  z_02had1 z_02hsv1
z_04had1 z_04hsv1 z_06had1 z_06hsv1 z_24had1 z_24hsv1]"
        return 1
```

```
    if len(sys.argv)>1 and sys.argv[1]=='debug':
        debug=True
        offset=1
        print "Reading from standardized dataset
file:",StandardizedDataSet
    if len(sys.argv)==1:
        clump = 5
        zLimitList = [8,8,  8,8,  8,8]
    else:
        clump=int(sys.argv[1])
        zLimitList = []
        for i in range(offset+2, offset+len(sys.argv)):
            zLimitList.append(float(sys.argv[i]))
    #Run
    clearOldFiles()
    i=0
    for hour in ['02','04','06']:
        for tissue in ['mock','hsv1']:
            try:
                os.makedirs('graphs\\24'+hour+tissue)
            except:
                pass
            print "making",hour,tissue,"with Z =",zLimitList[i]
            FilterDataUsingZFile(hour, tissue, clump, zLimitList[i],
StandardizedDataSet)
            i+=1
    i=0
    for hour in ['02','04','06']:
        for tissue in ['mock','hsv1']:
            print "running",hour,tissue,"with Z =",zLimitList[i]
            RunPls(hour,tissue)
            i+=1

main()
```

APPENDIX O: SAS TEMPLATE TO RUN KOLMOGOROV-SMIRNOV TEST AGAINST 24 HOUR VERSUS 2, 4, 6 HOUR DATA OF SAME DISEASE STATUS

```
/* SAS template to perform Kolmogorov-Smirnov test against
   24 vs 2,4,6 hour tissue of same type (mock vs mock,
   hsv vs hsv) */
filename grafout 'M:\Data\graphs\24@HR@TISS\';
goptions device=gif gsfname=grafout gsfmode=replace;

filename logfile 'm:\data\KsPls24@HR@TISS.log';
filename output  'm:\data\KsPls24@HR@TISS.lst';
proc printto  log=logfile print=output new;
run;

title 'PLS Model for @TISS at hour 24 vs @HR';
data sourceset;
    infile 'm:\data\KsPls5set24@HR@TISSdataset.dat' lrecl=@RECLEN;
    input hours 1-2 srcfilenum 5-7 tissue $ 10-13 linenum 16-18
hoursCat 21
          @VarOffsets;
run;

proc pls data=sourceset details nfac=6;
    model hoursCat = @Vars
          / solution;
    output out=OutPls predicted=ypred xscore=xscr;
run;

proc print data=OutPls;
run;

proc npar1way data=OutPls  EDF;
    class hours;
    var ypred;
    /*exact ks;*/
run;

proc printto;
run;
```

APPENDIX P: THREE PROGRAMS TO LOAD SECOND DATASET AND STANDARDIZE IT

```
#Program for loading dataset
import os,os.path,sys

def getFiles(destFileName):
    df=0
    fileList=[]
    try:
        df = open(destFileName,'a+')
    except Exception,err:
        print "Failed opening file, "+os.path.basename(destFileName)
+". "+str(err)
    else:
        cwd=os.getcwd()
        fileList=os.listdir(cwd)
    return df,fileList

def getHeader(line,f):
    display='.'
    peak='.'
    size=0
    for i in range(0,10):
        linePair=line.upper().split(',')
        if 'DISPLAYDIRECTION' in linePair[0]:
            display=linePair[1].strip(' \r\n')
        if 'PEAKDIRECTION' in linePair[0]:
            peak=linePair[1].strip(' \r\n')
        if linePair[0][0].isdigit():
            size=i
            break
        line=f.readline()
    return display,peak,size,line

def loadLine(line,xyList):
    x,y = line.split(',')
    x = x.strip(' \n\r')
    y = y.strip(' \n\r')
    if (x[0].isdigit() or x[0] in ('+','-','.')) and (y[0].isdigit()
or y[0] in ('+','-','.')):
        if 800.0 < float(x) < 1500.0 :
```

```
            xyList.append((float(x),x,y))

def writeLine (x, y, destFile, lineNum, fileNum, curve, fileType,
fileName,hours):
    out  = '%02.2d  ' % int(hours)
    out += '%03.3d  ' % fileNum
    if fileType[0].upper() == 'V':
        out += 'v  ' + '%4.4s  ' % fileType[1:5]
    else:
        out += 'n  ' + '%4.4s  ' % fileType[0:4]
    out += '%03.3d  ' % lineNum
    out += ('%-12.12s'%x + '      ')[0:13]
    out += ('%-12.12s'%y[0:9] + '      ')[0:10]
    out += '%1s  ' % curve
    #out += (display + ' '*7)[0:7]
    #out += (peak + ' '*7)[0:7]
    out += (fileName.replace(' ','') + ' '*75)[0:75]
    destFile.write(out+'\n')

def loadFile(fileName):
    xyList=[]
    f=open(fileName,'r')
    line=f.readline()
    display,peak,size,line=getHeader(line,f)
    while len(line)>10:
        loadLine(line,xyList)
        line=f.readline()
    f.close()
    return xyList

def main():
    (destFile,fileList) = getFiles(sys.argv[1])
    fileNum=0
    if destFile != 0:
        for fileName in fileList:
            if fileName[-4:]=='.csv':
                fileNum += 1
                print "Reading",fileName
                xyList = []
                xyList = loadFile(fileName)
                xyList.sort()
                lineNum=0
                for node in xyList:
                    lineNum += 1
                    writeLine(node[1],node[2],destFile, lineNum,
```

```
fileNum, sys.argv[4], sys.argv[2],fileName,sys.argv[3])
        destFile.close()
    else:
        print "Error opening file: "+sys.argv[1]

if len(sys.argv) == 1:
    print "syntax: ",sys.argv[0],"destFile fileType hours curve"
else:
    main()
```

```
#Program for removing variation in spectrum frequency values at the
#fifth and sixth decimal points
import os,sys

fp=open(sys.argv[1],'r')
ds={}
hour=0;file=1;v=2;tiss=3
row=4;x=5;y=6;curve=7;fname=8

line = fp.readline()
lineCount = 0
while len(line) > 0:
    lineCount += 1
    if lineCount%10000 == 0:
        print "read line",lineCount
    t=line.split()
    if ds.get('%04.1f'%float(t[x])):
        if t[x] not in ds['%04.1f'%float(t[x])]:
            ds['%04.1f'%float(t[x])].append(t[x])
            if len(t[x]) > len(ds['%04.1f'%float(t[x])][0]):
                ds['%04.1f'%float(t[x])].reverse()
    else:
        ds['%04.1f'%float(t[x])] = [t[x]]
    line = fp.readline()
fp.close()

fp=open(sys.argv[1],'r')
fpout=open(sys.argv[2],'w')

line = fp.readline()
while len(line) > 0:
    t=line.split()
    t[x] = ds['%04.1f'%float(t[x])][0]
    temp = '%04.3f' % float(t[x])
    out = line[0:23]
    out += ('%-12.12s' % temp + '          ')[0:13]
    out += line[36:]
    fpout.write(out)
    line = fp.readline()
fp.close()
fpout.close()
```

```
#Program for standardizing absorption data in dataset
import os,sys,math

if len(sys.argv) == 1:
    print "Syntax:  makeStdData  srcfile  destfile"
    sys.exit(1)
srcfile=sys.argv[1]
destfile=sys.argv[2]

fp=open(srcfile,'r')
fpout=open(destfile,'w')

d={}
hr=0;file=1;v=2;tiss=3;row=4
x=5;y=6;curve=7;fname=8;norm=9

#status values
readdata=1
calcvals=2
last=3
done=4

status=readdata
linenum=0
oldrec=0
sum=0.0
while status != done:
    if status == readdata:
        line = fp.readline()
        if len(line) <= 2:
            status = last
            fp.close()
        else:
            linenum += 1
            if linenum%10000 == 0:
                print 'read line',linenum
            t=line.split()
            t.append('0')
            r=int(t[row])
            if oldrec+1 <= r:
                d[r] = t
                oldrec = r
            else:
                status = calcvals
    elif status==calcvals or status==last:
```

```
        sum = 0.0
        for i in range(1,oldrec+1):
            sum += float(d[i][y])
        avg = sum/oldrec;
        var=0.0
        for i in range(1,oldrec+1):
            diff = float(d[i][y]) - avg
            var += diff*diff/(oldrec-1)
        stddev=math.sqrt(var)
        for i in range(1,oldrec+1):
            #testing
            #if linenum > 508000:
            #    print d[i]
            d[i][norm] = (float(d[i][y])-avg)/stddev
        for i in range(1,oldrec+1):
            out  = d[i][hr]+'  '
            out += d[i][file]+'  '
            out += d[i][v]+'  '
            out += d[i][tiss]+'  '
            out += d[i][row]+'  '
            out += ('%-9.9s'%d[i][x] + '      ')[0:10]
            out += ('%-12.12s'%d[i][y] + '      ')[0:10]
            out += '%01.8f  '%avg
            out += '%01.8f  '%stddev
            out += ('%02.8f    '%d[i][norm])[0:13]
            out += d[i][curve]+'  '
            out += (d[i][fname] + ' '*75)[0:75]
            fpout.write(out+'\n')
        if status==calcvals:
            sum = float(t[y])
            d={r:t}
            oldrec=r
            status = readdata
        elif status==last:
            status=done
            fpout.close()
print "Done!!!"
```

APPENDIX Q: PROGRAM TO PLOT MEAN CURVES AND WILCOXON-MANN-WHITNEY CURVES

```
/*program name: Curves.sas
  purpose: to plot mean and variance curves of second dataset and
  to calculate and plot the Wilcoxon-Mann-Whitney curve for
  the means curves and for the overall standardized dataset.*/
goptions device=gif reset=global gunit=pct border cback=white
         colors=(black blue green red)
         ctext=black
         ftitle=swissb ftext=swiss htitle=4 htext=3;

filename logfile 'm:\data2\CurvesAll.log';
filename output 'm:\data2\CurvesAll.lst';
proc printto  log=logfile print=output new;
run;

title "DataSet";
data masterset;
    infile 'm:\data2\mastDataset3digitSTD.txt';
    input hours 1-2 srcfilenum 5-7 vtype $ 10 tissue $ 13-16 linenum
19-21
          spectrum 24-31 absorbtion 34-41 avg 44-53 stdv 56-65
absorbstd 68-78
          curve 81 srcfile $ 84-155;
run;
%let ctotal=8;  /*how many curves*/

proc sort data=masterset out=results1;
    by curve tissue spectrum;
run;
/*proc print data=smaster (obs=500);run;*/

title "Means";
proc means n mean std;by curve tissue spectrum;
    var  absorbstd;
    output out=resultsm mean=mean_absorption std=stddev_absorption;
run;
proc print data=resultsm (obs=2); run;

%macro singleCurvTxt(hdr);
    title "  ";
```

```
    symbol1 color=red interpol=join;
    symbol2 color=blue interpol=join;
    footnote1 c=red 'Red = hsv1  ' c=blue  '  Blue = mock';
%mend singleCurvTxt;

%singleCurvTxt(Mean Curves);
proc gplot data=resultsm;plot mean_absorption*spectrum=tissue/overlay;
by  curve;  run;

%singleCurvTxt(StdDev Curves);
proc gplot data=resultsm;plot
stddev_absorption*spectrum=tissue/overlay; by  curve;  run;

%macro mcurv;
    data multcurv;set resultsm;
    %do c=1 %to &ctotal;
        if curve=&c and tissue='hsv1' then ctiss=&c;
        if curve=&c and tissue='mock' then ctiss=%eval(&c+&ctotal);
    %end;
    run;
%mend mcurv;
%mcurv;

%macro multCurvTxt(hdr);
    title "   ";
    %do j=1 %to &ctotal;
        symbol&j color=red interpol=join;
        symbol%eval(&j+&ctotal) color=blue interpol=join;
    %end;
%mend multCurvTxt;

%multCurvTxt(Combined Mean Curves);
proc gplot data=multcurv;plot mean_absorption*spectrum=ctiss/overlay
nolegend;  run;

%multCurvTxt(Combined StdDev Curves);
proc gplot data=multcurv;plot stddev_absorption*spectrum=ctiss/overlay
nolegend;  run;

proc print data=multcurv (obs=20); run;

/*Generate WMW Z plot from means*/
/* title "Wilcoxon Mann Whitney Z Score"; */
title " ";
proc sort; by spectrum; run;
```

```
proc npar1way wilcoxon;
    class tissue;
    var mean_absorption;
    by spectrum;
    output out=meanwz wilcoxon;
run;

%singleCurvTxt(Wilcoxon Mann Whitney Z Score of Means);
proc print data=meanwz (obs=20); run;
proc gplot data=meanwz;plot z_wil*spectrum;  run;

%singleCurvTxt(Wilcoxon Mann Whitney Z Score of Raw Data);
proc sort data=masterset out=wmwraw; by spectrum; run;
proc npar1way data=wmwraw wilcoxon;
    class tissue;
    var absorbstd;
    by spectrum;
    output out=wmwrawz wilcoxon;
run;
proc gplot data=wmwrawz;plot z_wil*spectrum;  run;

proc printto; run;
```

APPENDIX R: PROGRAM TO CALCULATE THE D_C STATISTIC

```
/* Filename GetDataForBootStrap.sas
   Purpose: To calculate the Difference of Composites, DC, statistic.
*/
/*goptions device=gif reset=global gunit=pct border cback=white
         colors=(black blue green red)
         ctext=black
         ftitle=swissb ftext=swiss htitle=4 htext=3;
*/
filename logfile 'M:\data2\GetDataForBootStrap.log';
filename output 'M:\data2\GetDataForBootStrap.lst';
proc printto  log=logfile print=output new;
run;

title "DataSet";
data masterset;
    infile 'm:\data2\mastDataset3digitSTD.txt';
    input /*hours 1-2 srcfilenum 5-7 vtype $ 10*/
          tissue $ 13-16 /*linenum 19-21*/
          spectrum 24-31 /*absorbtion 34-41 avg 44-53 stdv 56-65*/
          absorbstd 68-78
          curve 81 /*srcfile $ 84-155*/;
run;
%let ctotal=8;  /*how many curves*/

proc sort data=masterset out=data_cts;
    by curve tissue spectrum;
run;

title "Means";
proc means n mean;
    by curve tissue spectrum;
    var  absorbstd;
    output out=data_means8
           mean=mean_absorption;
run;
proc print data=data_means8 (obs=20); run;

title "Create Ranges";
data data_ranges;
    set data_means8;
    if 1415<=spectrum<=1455 then ranges='1415-1455';
```

```
    else if 1050<=spectrum<=1070 then ranges='1050-1070';
    else ranges='allothers';
    if ranges='allothers' then delete;
run;
proc sort out=data_ranges_ctr;
    by curve tissue ranges;
run;
proc print data=data_ranges_ctr (obs=100); run;

title "Sum Within Ranges";
proc means sum data=data_ranges_ctr;
    by curve tissue ranges;
    var mean_absorption;
    output out=data_range_sums
           sum=sum_range;
run;
proc print data=data_range_sums; run;

title "Calculate per Curve and Tissue. Composite Difference DC =
1415range - 1050range";
data data_range_sum_pos;
    set data_range_sums;
    if ranges='1415-1455';
    DC=sum_range;
run;
data data_range_sum_neg;
    set data_range_sums;
    if ranges='1050-1070';
    DC=sum_range;
run;
proc compare outdif
             base=data_range_sum_neg
             compare=data_range_sum_pos
             out=data_diffs;
    var DC;
    by curve tissue;
run;
proc sort data=data_diffs out=data_diffs_sorted;
    by tissue curve;
run;
proc print; run;

title "Generate Means and SD's of Difference of Composite Sums
Variable DC for Mock and Hsv1";
proc means mean var data=data_diffs_sorted;
```

```
    by tissue;
    var DC;
    output out=data_diff_stats
           mean=DC_mean
           var=DC_var;
run;
proc print data=data_diff_stats; run;

proc printto; run;
```

APPENDIX S: PROGRAM TO CALCULATE USING BOOTSTRAP THE AUC AND SPECIFICITY CONFIDENCE INTERVALS

```
#Python script, bootstrap3.py, to calculate CI for AUC and specificity
import sys
import random
import math
debug=False

def mean(xlist):
    sum = 0.0
    for x in xlist:
        sum += float(x)
    return sum/len(xlist)

def svar(xlist):
    sum2 = 0.0
    xbar=mean(xlist)
    for x in xlist:
        sum2 += (float(x) - xbar)*(float(x) - xbar)
    return sum2 / (len(xlist) - 1)

def BootStrap(sampleSize,iters,zalpha):
    basex=[21.1279, 27.9380, 12.8544, 11.3004, 19.1465, 8.7847,
21.9278, 5.1373]
    basey=[ 7.1749, 1.6203, 2.6050, 3.9264, 5.5607, 3.0025, -10.1459,
-2.0627]
    llist=[]
    qlist=[]
    for i in range(0,iters):
        xlist=[]
        ylist=[]
        for j in range(0,sampleSize):
            idx=random.randint(0,7)
            xlist.append(basex[idx])
            ylist.append(basey[idx])
        li = (mean(xlist)-mean(ylist))/math.sqrt(svar(xlist)
+svar(ylist))
        llist.append(li)
        qi = (mean(xlist)-mean(ylist)-
zalpha*math.sqrt(svar(xlist)))/math.sqrt(svar(ylist))
        qlist.append(qi)
```

```
    return llist,qlist

def Quartile(xlist,pct):
    clist=xlist[0:]
    clist.sort()
    n=len(clist)
    low=clist[int(round((n-1)*pct/100.0))]
    high=clist[int(round((n-1)*(100-pct)/100.0))]
    return low,high

def BootStrapStats(sampleSize,iters,zalpha,alpha):
    #zalpha is the value from the t-distribution with alpha value of
alpha
    #and sampleSize-1 degrees of freedom
    pct=2.5
    llist,qlist = BootStrap(sampleSize,iters,zalpha)
    print 100*(1-2*pct/100.0),"percent CI for AUC
is",Quartile(llist,pct)
    print str(100-2.0*pct)+"% CI for specificity at
sensitivity="+str(alpha)+"% in 'Z' values is",Quartile(qlist,pct)


if len(sys.argv) > 1 and sys.argv[len(sys.argv)-1]=="debug":
debug=True
sampleSize=int(sys.argv[1])
iters=int(sys.argv[2])
#p values below from SPlus -  qt(0.95,7), qt(0.90,7), qt(0.80,7)
BootStrapStats(sampleSize,iters,1.894579,0.95)
BootStrapStats(sampleSize,iters,1.414924,0.95)
BootStrapStats(sampleSize,iters,0.8960296,0.95)
```