

Admittance measurements: Algorithm & Application

December 05 Study Period

P.F. Derwent
AD/Pbar Source

December 29, 2005

Abstract

Documentation of admittance measurements application and algorithm.

1 Admittance measurements

We have made admittance measurements in the Debuncher and Accumulator using scraper scans for years. The practice has been to make a fast time plot of a loss monitor and spectrum analyzer video out versus scraper position. A determination of the touch point (from where the loss monitor 'takes off') and the extinction point (where the video out 'flat lines') is done and then the distance is measured from these two points. There are lots of opportunities for personal bias in defining the touch point and extinction point. An algorithm has been developed to take out the human element.

The algorithm looks for the point of maximum curvature of the loss monitor count vs scraper position to define the touch point and similarly for the video out vs scraper position for the extinction point. A smoothing algorithm is included to take out noise in the sampling. It was developed by Andrey Gvozdev during the summer of 2005 under guidance from Keith Gollwitzer, Steve Werkema, and Paul Derwent.

The curvature of a function y is defined as

$$\rho = \frac{|y''|}{(1 + (y')^2)^{\frac{3}{2}}} \quad (1)$$

However, if we let y go to $3y$, the maximum of the curvature is in a different place. Andrey realized that using $(a^2 + (y')^2)^{\frac{3}{2}}$, where a is the norm (the maximum of y') of the function y will preserve the position of the maximum curvature to changes in normalization (e.g., changes in beam intensity). y is derived from the data by a regularization algorithm. This algorithm does smooth out large scale deviations (e.g., hiccups in the data taking process). A full description of the mathematics of the algorithm is included as an appendix to this note. This algorithm has been implemented in a Java application available on the APPIX P page.

2 Data taking and application

Rather than using fast time plots, the scraper position, loss monitor counts, and spectrum analyzer video out devices are logged in a lumberjack sampling at 15 Hz. By knowing when the

Plane	Loss Monitor	Touch Point Range	Extinction Point Range
Debuncher Horizontal	D:LM3Q5	25 - 35 mm	10 - 16 mm
Debuncher Vertical	D:LM3Q8	12 - 22 mm	0 - 6 mm
Accumulator Horizontal	A:LM4Q1	15 - 25 mm	3 - 9 mm
Accumulator Vertical	A:LM306	10 - 16 mm	-5 - 1 mm

Table 1: Defaults used in Java application.

scraper scan is complete, we can pull the data out of the lumberjack and implement the touch and extinction point algorithm. The application uses default loss monitors and ranges for the touch point and extinction point (see table 1), taking 3 minutes of data from the data logger. A sample of the data and the identified touch and extinction points are shown in figure 1.

The regularization algorithm does impart a bias in the calculation of the extinction point. As there is a discontinuity where the power above the noise floor goes to zero and the video out 'flat lines', the algorithm smooths out this discontinuity over 200μ , thus biasing the extinction point slightly to the left (on the scale of 100μ).

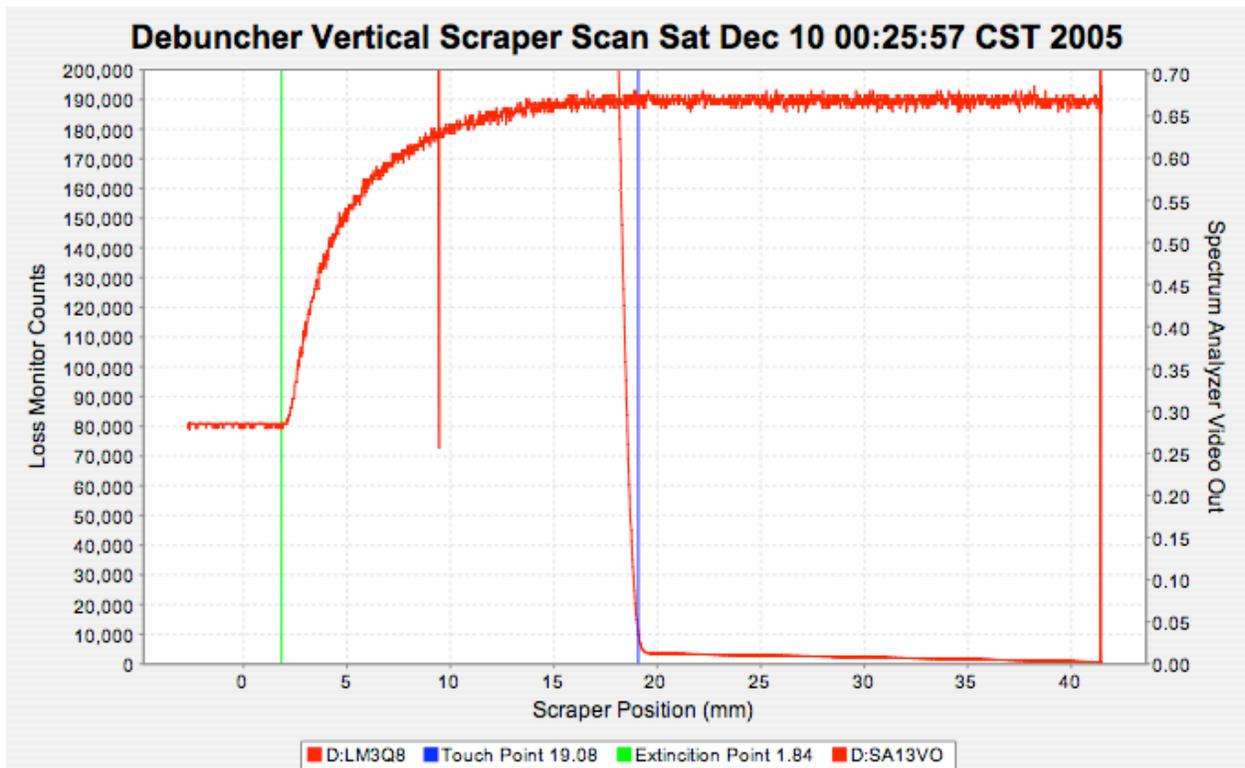


Figure 1: Aperture scan for the Debuncher vertical position taken December 10th. The touch and extinction points are listed in the legend. The admittance is the square of the difference divided by the value of the β function at the scraper.

Andrey Gvozdev
Novosibirsk State University
Phone: +7 913 927 4008
Email: SPDooh@gmail.com

TouchPoint is a module written on C++ for touch point definition from scraper's scans.

Short description of methods and variables using in TouchPoint.h

Public methods:

setData(vector<double>& xNew, vector<double>& yNew) or void SetData(string fName)

Input data from out source. First way is using two vectors. Second is reading data from file. In the file there should be two columns: x-data and y-data correspondingly.

void Run()

This function is used for running methods described below. It should be run after **SetData**. This function processes two vectors. Firstly, it sort data (**MSort**). Secondly, it throw away erroneous and seriate points (**Mthrow**). After this it reduce data to a constant x step (**EqStep**). Then it differentiate function (**Mdiff**) and, at the end, calculate **CURV** and **TP**.

void Run1()

void Run1ad(double NORM)

This two functions together mostly do the same that **Run**, but instead of automatically **NORM** calculation in **MCalc** you can set it by hand or calculate it in another way, looking at first derivative.

Private methods:

SLUSolve(DMatr& M, vector<double>& res)

This function is used for solving system of linear equations $Q \cdot X = F$ by Gauss method. It takes matrix M with dimensions $N \times N+1$ as a parameter, where N is a number of equations. First N columns are the matrix of the system (Q), and the last one is a right part. After calculations function write the result to vector **res**.

PolyC(vector<double>& x, vector<double>& y, int n, vector<double>& PCoeff)

This function is used for calculating coefficients of the approximation polynomial by method of least squares. Let write a functional $\Omega = \sum_{x_i} [y_i - f(x_i)]^2$, where $f(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_n x^n$ - polynomial with unknown coefficients a_i . If polynomial is not far from our points, then Ω is small, and vice versa. For Ω minimization let put to zero derivatives $\frac{d\Omega}{da_i} = 0$. Thus we can get linear system of equations with respect to a_i :

$M \times A = F$, where $M_{kl} = \sum_i x_i^{k+l-2}$, $F_k = \sum_i y_i x_i^{(k-1)}$, A - coefficients vector. (x,y) are points to be approximated, n is a power of the polynomial and **PCoeff** is a resulting vector A.

void MSort(vector<double>& x, vector<double>& y)

This function is just a sorting by method of fon Neuman. It sorts array (x,y) with respect to x.

void PThrew(vector<double>& x, vector<double>& y)

This function is used for throwing away erroneous seriate points. Firstly, we can do several measurements in the same position. In this case we leave only one x-element with this value of position and write to the corresponding y the average value. Secondly, several points could be an error of the experiment. Most of all points lies on the smooth line. However some point could be far from its neighbours. In this way we suppose that it is an error and throw it away. For this we inscribe our curve to the square in order to achieve the equivalent axis. Then we take the left point, write it into the new array and find the next point, which is the nearest to our one. This point we also write to the idem array and so on.

Thus, if the point is the experimental's fault (i.e. if the distance between its neighbours is less than distance to the neighbour) then it will be skipped.

void EqStep(vector<double>& x, vector<double>& y, double Env, double h)

In most numerical methods it is more simply to work when distance between points is constant. This function takes the old array (x,y) and builds the new one with constant step h. The first new x point is x[0]+Env, the next is x[0]+Env+h and so on, until the distance to the x[N] is less than Env. For calculating y[i] we calculate parabola which is the best approximation for all old points in [x[i]-Env;x[i]+Env], and then compute its value in x[i]. Thus we have new array (x,y) with a constant step by x, and at the same time get some smoothing.

Parameters: (x,y) – array for calculation, h – step in the new array, Env – neighborhood of the point for parabola calculation.

void MDiff(vector<double>& x, vector<double>& y, vector<double>& FDiff, vector<double>& SDiff, double alpha1, double alpha2)

This function is used for calculation the first and the second derivatives. If the function is known from measurements, there are some random errors. So if we begin to calculate derivative by the nearest points, we can get great deviation. So we have to find some regularization algorithm for differentiation. One way to solve this problem is to get integral equation from differential one:

$$U = f' \rightarrow \int_a^b \left(\int_t^b U(\xi) d\xi - f(b) + f(t) \right)^2 dt = 0$$

If U is a solution of the left equation, it also minimized a functional written on the right. But we want to get a stable solution with respect to the small deviation of f. So we can add item to the functional, which will be responsible for the sleekness of the derivative:

$$\int_a^b \delta x^2 dt + \alpha \int_a^b \left[U^2(t) + \left(\frac{dU(t)}{dt} \right)^2 \right] dt = \min$$

where α - is a regularization parameter, and $\delta x = \int_t^b U(\xi) d\xi - f(b) + f(t)$ - discrepancy, which will be achieved after substitution of our function to the initial equation. After that we apply numerical integration and differentiation. To do this let calculate $\int (U')^2 dt$ using formula of average, on the same time replace differentiation by difference:

$$\int_{t_i}^{t_{i+1}} \left(\frac{dU}{dt} \right)^2 dt \approx \frac{(U_{i+1} - U_i)^2}{h}$$

Other integrals we calculate using trapezium rule:

$$\int U^2 dt = h \sum_{i=0}^N c_i U_i^2 \quad U(i) = U(t_i)$$

$$\delta x(t_i) = h \sum_{m=0}^N B_{im} U_m - f_N + f_i$$

$$\int (\delta x(t))^2 dt = h \sum_{i=0}^N c_i (\delta x_i)^2$$

where introduced following table of symbols:

$$B_{im} = \begin{cases} 1/2 & \text{if } m=1 \text{ or } m=N \\ 1 & \text{if } m > i \text{ or } m \neq N \\ 0 & \text{else} \end{cases}$$

$$c_i = \begin{cases} 1/2 & \text{if } i=1 \text{ or } i=N \\ 1 & \text{else} \end{cases}$$

$$x_i = x_0 + (i-1) * h$$

Collecting all together we will achieve following expression:

$$\Phi = h \sum_{i=0}^N c_m [\delta x_i]^2 + \alpha h \sum_{i=1}^N c_i U_i^2 + \frac{\alpha}{2h} \sum_{i=0}^N (U_{i+1} - U_i) = \min$$

For the functional minimization let equate to zero its derivatives. Thus we get linear system of equations with respect to U_i :

$$\frac{d\Phi}{dU_i} = 2 \alpha h c_i U_i + \alpha \Lambda(U_i) + F_i + \sum_{m=0 \text{ to } N} N Q_{im} = 0$$

where introduced following table of symbols:

$$x_{i+1} - x_i = \text{const} \quad \forall i$$

$$F_i = 2h \sum_{l=0}^N c_l B_{li} (f_l - f_N)$$

$$Q_{im} = 2h^2 \sum_{k=0}^N c_k B_{ki} B_{km}$$

Solving this system of equations we can get f derivative. Using this method two times we can get second derivative.

As a parameter this procedure gets the function, which we want to differentiate (x,y) with constant interval between points ($x_{i+1} - x_i = \text{const} \quad \forall i$), regularization parameters alpha and alpha2 for the first and the second derivatives correspondingly, and returns two vectors **FDIFF** and **SDIFF**.

void MCalc()

It was assumed that curvature of scraper's data has minimum in touch point. Curvature is given by following expression:

$$CU = \frac{|y''|}{(1 + (y')^2)^{\frac{3}{2}}}$$

But this function doesn't have maximum in the same place for y and const*y. Therefore better value is:

$$C = \frac{|y''|}{(a^2 + (y')^2)^{\frac{3}{2}}}$$

Where a is a function's norm. As the norm was chosen maximum of the first derivative: $a = \max_{[-\infty; +\infty]} (y')$. Physical meaning of this norm is maximum of the beam's envelope. Since data can be without maximum on the first derivative, we should use extrapolation to find it. We suppose that Gauss function with $\sigma \approx 10 \text{ mm}$ is a good approximation for the beam's envelope. So we can find our maximum.

Then using class variables FDIFF, SDIFF and X we calculate function C and write it into the CURV. CURV's maximum is in the MTP.

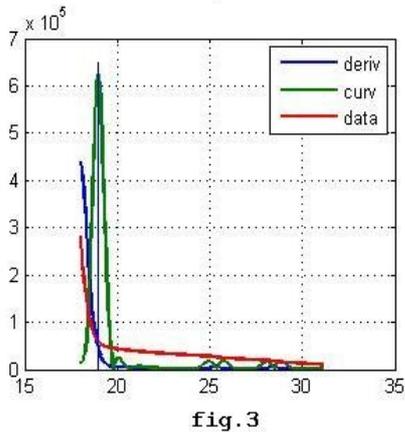
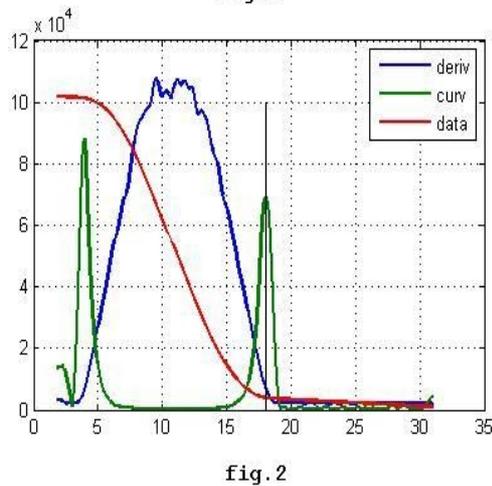
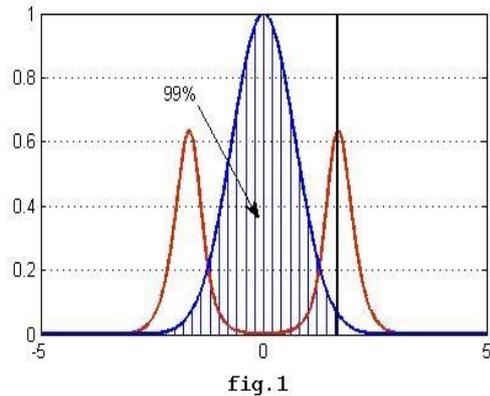
Variables

(**X,Y**) – set of points

(**X,FDIFF**) - first derivative of (**X,Y**)

(**X,SDIFF**) – second derivative

(**X,CURV**) – C from **MCalc**



Features

There must be at least 3 points in input data at every $2 \cdot \text{ENV}$ interval (**ENV** declaration see at EqStep).

Gauss extrapolation in **MCalc** doesn't work well, so bigger error can appear when there is no maximum in first derivative. In this case better precision can be achieved if **NORM** in **MCalc** is set by hand. But anyway TP has small dependence from **NORM**.

Also, sometimes there are several maximum at graphic as shown at fig.2. But we need to find only left one. So sometimes it is better to look at graphics. So this program is not really automatized.

If the beam has the gaussian distribution, then MTP is a point which bound 99% of gauss's area (fig.1)

At fig. 2 shown result of function's calculations. We can see that C-graphic has well defined maximum. MTP bounds 99.5% of beam's square. At fig.3 shown another example. Cause there is no second edge we can not calculate what area is bounded. But mostly picture look like fig.2.