SLAC-PUB-14640

1.1 Matlab Based LOCO

Greg Portmann
Mail to: gregportmann@lbl.gov
Lawrence Berkeley National Lab, Berkeley, CA 94720, USA

James Safranek and Xiaobiao Huang
Stanford Linear Accelerator Center, Menlo Park, CA 94025, USA

1.1.1 Introduction

The LOCO algorithm has been used by many accelerators around the world. Although the uses for LOCO vary, the most common use has been to find calibration errors and correct the optics functions. The light source community in particular has made extensive use of the LOCO algorithms to tightly control the beta function and coupling. Maintaining high quality beam parameters requires constant attention so a relatively large effort was put into software development for the LOCO application.

The LOCO code was originally written in FORTRAN. This code worked fine but it was somewhat awkward to use. For instance, the FORTRAN code itself did not calculate the model response matrix. It required a separate modeling code such as MAD to calculate the model matrix then one manually loads the data into the LOCO code. As the number of people interested in LOCO grew, it required making it easier to use.

The decision to port LOCO to Matlab was relatively easy. It's best to use a matrix programming language with good graphics capability; Matlab was also being used for high level machine control; and the accelerator modeling code AT, [5], was already developed for Matlab. Since LOCO requires collecting and processing a relative large amount of data, it is very helpful to have the LOCO code compatible with the high level machine control, [3]. A number of new features were added while porting the code from FORTRAN and new methods continue to evolve, [7][9]. Although Matlab LOCO was written with AT as the underlying tracking code, a mechanism to connect to other modeling codes has been provided.

1.1.2 The Matlab Code and Functionality

1.1.2.1 Graphical User Interface (GUI)

Figure 1 shows the Matlab-LOCO GUI. The GUI was the most time consuming part the coding process but without a GUI it would be much more difficult to follow, understand, and direct what is going on during a LOCO run.

Work supported in part by US Department of Energy contract DE-AC02-76SF00515.

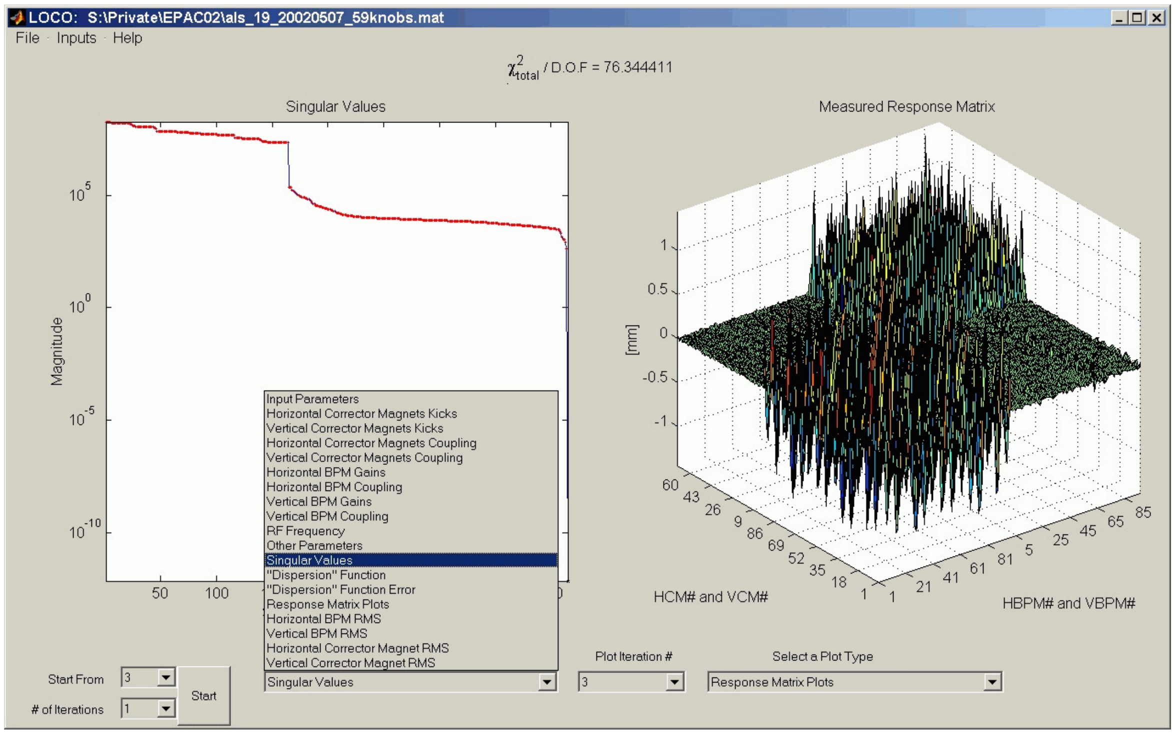


Figure 1: Matlab LOCO GUI.

LOCO uses an iterative method to converge to the minimum χ^2. The process is initiated using the “Start” button in the bottom left of Figure 1 and the “# of Iterations” pull-down menu selects the iterations. The information displayed on each plot is selectable from the plot selection menu shown expanded in the figure. It’s common to go back to a previous iteration with the “Start From” pull-down menu, change an input, and rerun the iteration. The ability to quickly experiment with different LOCO setups and view the results is quite useful. The “Inputs” menu is used to make choices concerning the fitting for each iteration. The “Inputs” menu is shown below and many of the options are discussed in the following sections.

Input Menu

└ Minimization Algorithm

Gauss-Newton
Gauss-Newton (with parameter costs)
Levenberg-Marquardt
Scaled Levenberg-Marquardt

└ Fit BPM Coupling

└ Edit BPM List

└ Fit Corrector Magnet Kicks

└ Fit Corrector Magnet Coupling

└ Edit Corrector Magnet List

└ Include Off-Diagonal Response Matrix Terms

└ Fit Energy Shifts at the Horizontal Corrector Magnets

└ Fit Energy Shifts at the Vertical Corrector Magnets

└ Response Matrix Calculator

Linear or Full Response Matrix Calculator
Fixed Path Length or Fixed Momentum

└ Response Matrix Measurement Method (Bi-directional or uni-directional)

└ Include " Dispersion" as part of the Response Matrix

Weight for horizontal dispersion
Weight for vertical dispersion

└ Fit the RF Frequency for Measured Dispersion

└ Dispersion Measurement Method (Bi-directional or uni-directional)

└ Auto-Correct Deltas

└ Singular Value Selection Method

└ Normalize (column weights)

└ Outlier Rejection

└ Calculator Error Bars

└ Single or Double Precision

No information is saved in the GUI. All information is contained in the Matlab data file which is saved after each iteration. Hence, if the computer crashes or runs out of memory on the sixth iteration, the last five will have already been saved.

1.1.2.2 The Minimization Algorithm

The algorithm is discussed in detail in a number of references so it will not be discussed here. The original Gauss-Newton search LOCO algorithm can be found in Reference [1] and [6]. Reference [7] discusses a weighted version of the Gauss-Newton method as well as the Levenberg-Marquadt method, [9]. All methods are selectable from the GUI.

1.1.2.3 Orbit Response Matrix Calculation

There are a couple options for computing the model response matrix. A linear approximation option was introduced for calculation time reasons. The linear method orbit response matrix computation is based on the numerically obtained 4-by-4 transfer matrixes at each corrector and BPM, the model dispersion function, and model momentum compaction factor. The nonlinear method iteratively searches for a closed orbit in the presence of the corrector magnet kick. For many accelerators the time spent computing a full nonlinear model is not necessary on the first couple iterations if at all. The nonlinear methods are slower but include the non-linear effects due to sextupoles and other non-linear elements.

The second option is to choose whether to hold the path length fixed or the momentum fixed. How these options are implemented in AT is shown Table 1.

	Constant Momentum	Constant Path Length
Linear	Transfer Matrix	Transfer Matrix + Dispersion Term
Full NonLinear	findorbit4	findsyncorbit

Table 1: Response Matrix Calculation Methods.

The response matrix calculation method can change more than just how the response matrix is numerically generated. The merit function minimized in LOCO is just the difference between the model and measured response matrix. However, when using the constant momentum method, model response matrix is modified by the fit energy change at the corrector magnets.

$$M_{model} = (\Delta p/p)_{fit}\, \eta_{measured} + M_{AT\ model}$$

where $\eta_{measured} = -\alpha * RF * \Delta Orbit / \Delta RF$.

$\Delta p/p$ is the energy change due to that corrector when generating the response matrix. The constant momentum method was developed to tackle the problem that orbit errors in the quadrupoles and sextupoles cause dispersion errors which are not accounted for in the model. To correct this, the energy shift at the correctors are fit and the model is altered by the energy shift times the actual dispersion in the machine. Hence, even with a perfect fit of all LOCO parameters the model and measured dispersion will not be forced to match. Note that the model response matrix has been changed by a term proportional to the measured dispersion which means measurement errors have been added to the model. Although it is a little odd to introduce measurement errors to a model, this method was chosen since changing the measured response matrix introduces other logical problems. In general, one would not include the dispersion as a column of the response matrix or fit the RF frequency when using the constant momentum method. This method was developed to zero the weighted dispersion in the least squares algorithm.

When using LOCO to correct the optics, localizing the source of the error is not important. All the calibration and magnet feeddown errors need to be corrected with the available power supplies. The constant path length response matrix is preferred for this case.

1.1.2.4 Dispersion Fitting

Since the dispersion function is relatively fast and easy to measure, there is an option to include it in the fit as an additional column in the response matrix – much like including another horizontal steering magnet.

Fitting the response to steering magnets alone usually gives a model with the correct tunes and beta functions, but does not always reproduce the dispersion as accurately. This can happen because the beta functions do not vary much with dipole magnet errors around the ring, while the dispersion does. Or it can happen when fitting a subset of the actual gradient errors in order to find the best correction as opposed to looking for the source of the error. The non-local gradient corrections can correct beta function distortion associated with the gradient error, but not necessarily the dispersion distortion. Including dispersion explicitly as a column of the response matrix forces LOCO to generate a model that accurately reflects both the beta functions and the dispersion of the real storage ring. The dispersion fitting option can be useful for controlling dispersion to achieve low emittance.

Including the dispersion in the fit can also provide an absolute calibration of the BPMs and correctors magnets. Without the dispersion there is always a degeneracy between the scaling of the BPMs and corrector magnets. This will create a small singular value per plane which needs to be removed. If dispersion is included and the RF frequency is not fit (which is often the case since the accuracy of RF changes is usually very good) then the horizontal degeneracy should disappear. If there is substantial vertical dispersion then the vertical degeneracy will also disappear. To force this, the coupling is sometimes increased just to obtain a good vertical BPM calibration.

1.1.2.5 BPM and Corrector Magnet Calibration

There is an option to compute the BPM and corrector magnet gain and coupling corrections. Basically the BPM corrections are applied to the model response matrix to best match the measurement using the follow equation.

$$\begin{pmatrix} x_{meas} \\ y_{meas} \end{pmatrix} = \begin{pmatrix} g_{x,loco} & c_{x,loco} \\ c_{y,loco} & g_{y,loco} \end{pmatrix} \begin{pmatrix} x_{\mathrm{mod}el} \\ y_{\mathrm{mod}el} \end{pmatrix}$$

There are also conversion functions to gain-crunch-roll format that was used in the FORTRAN LOCO code.

$$\begin{pmatrix} x_{meas} \\ y_{meas} \end{pmatrix} = \frac{1}{\sqrt{1-C^2}} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & C \\ C & 1 \end{pmatrix} \begin{pmatrix} g_x x_{\mathrm{mod}el} \\ g_y y_{\mathrm{mod}el} \end{pmatrix}$$

It is also possible to include single view BPMs.

For corrector magnets LOCO computes the gain and coupling terms which relate the kick applied to the accelerator (measured) to fit value of the kick (actual). These values are often converted to gain and roll coordinates (horizontal corrector shown below).

$$\begin{pmatrix} \delta_{x,Actual} \\ \delta_{y,Actual} \end{pmatrix} = \begin{pmatrix} g_{loco} \\ c_{loco} \end{pmatrix} \delta_{x,meas} = \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \end{pmatrix} * gain * \delta_{x,meas}$$

where δ refers to the steering magnet kick angle. Including BPM and steering magnet calibrations in LOCO fitting is nearly always the best choice.

1.1.2.6 Error Bar Propagation

Matlab LOCO includes an analytical calculation of the error bars on the fit parameters associated with BPM random measurement noise. This calculation gives a lower bound on the error bars, because is does not include systematic errors. The calculated error bars, however, are still useful as a guide to the relative accuracy with which the parameters can be fit. Fit parameter error bars can also be determined empirically by analyzing multiple response matrices and calculating the rms variation in the fit parameters over the different fit models.

Since the error bar calculation takes computer time there is an option to turn it off. The calculation gets particularly lengthy if the dispersion function is used without unity weights. For instance, in the typical least square equations, $y = Ab + e$, the assumed variance of input error is $\mathrm{var}(e) = \sigma^2 I$ and the variance of the least squares estimate $\hat{b} = (A'A)^{-1} A'y$ is $\mathrm{var}(\hat{b}) = \sigma^2 (A'A)^{-1}$. If all the input error variances are not equal then a row weight is used to force them to be so. This is done in LOCO with the BPM sigma. However, if a non-unity weight is put on the dispersion then the simplified variance calculation cannot be used. The full equation is $\mathrm{var}(\hat{b}) = (A'A)^{-1} A'\Sigma A (A'A)^{-1}$ where Σ is the covariance matrix of the input error. (Note: $\hat{b}$ is no longer an unbiased estimator.) The default in LOCO is to output only the diagonal of the covariance matrix of $\hat{b}$, but there is an option in the GUI to output the full covariance matrix.

When singular values are removed from the gradient matrix, A, interpreting the error bar is more difficult. Basically, removing a singular value removes a dimension so it reduces the ability of the least squares algorithm to find the best minimum and it reduces a way for the input error to project onto the estimates $\hat{y}$ and $\hat{b}$. The net effect is to increase the residual error, $y - \hat{y}$, but decrease the variance of the fit, $\mathrm{var}(\hat{y})$. Usually only very small singular values are removed. Very small singular values corrupt the numerical inverse and removing them usually has very little impact on the residual error or the variance.

1.1.2.7 Outlier Data Rejection

There is also an option to reject outlier data based a user defined threshold. Before each iteration, the rms difference between the model and measured response matrices is calculated. This difference for each point is compared against the threshold times the total rms. The calculation is done twice. The first test will remove very large outliers, if they exist, so that second test has a more accurate rms calculation. Using outlier data rejection is usually the best choice.

1.1.2.8 Numerical Derivative Calculation

The χ^2 minimization routine requires the numerical derivatives of the model response matrix with respect to each of the fit AT parameters. In order to numerically compute the gradient w.r.t. each parameter being fit a reasonable delta change in each parameter needs to be found. The user can determine the appropriate step sizes by trial and error or the code will automatically determine the appropriate step sizes. This is the "Auto-Correct Deltas" input. The deltas are chosen to keep the RMS change in the response matrix equal to one micron when calculating numerical derivatives. Including auto-correct deltas is usually the best choice.

1.1.2.9 Normalization

Nomalization refers to column weighting of the gradient matrix in the least squares problem, min|Ab - y|. (Row weighting of this matrix was discussed in the Error Bar Propagation section above.) Normalization can be switch on or off. Normalization is used to help with potential numerical precision issues. If the columns of the A-matrix have very different relative scaling, then the SVD calculation could run into numerical problems. Column weight is the same as changing the units of the parameter fits. Normalization also affects the size of the singular values. The effect of normalization when singular values are removed has not been studied. Using normalization is usually the best choice.

1.1.2.10 Memory Requirements and Speed

The memory requirement is usually driven by the size the response matrix and the number of parameters to be estimated. Matlab uses 8-byte precision for matrices, so the numerical derivative of the model response matrices requires 8 * # of parameters * #BPMs * #Correctors bytes of memory for the fully coupled case. A rough estimate of the memory required for the LOCO subroutine is 2.5 times this number.

There is a single precision option to reduce the required memory but there have been cases where single precision produced incorrect results. With 64-bit operating systems now widely available, the memory issue is not the problem it once was.

The LOCO code converges on the order of three times faster than the original FORTRAN code. The time required for a particular storage ring depends strongly on the size of the response matrix and number of fit parameters. To give an example, it takes about 15 minutes to converge to a solution for the Advanced Light Source without coupling, when running on a CPU with speed equivalent to about 2 GHz.

1.1.3 Data Flow

The LOCO algorithm requires a lot of data. A great attempt was made to come up with sensible method for data handling which makes calling LOCO recursively relatively easy. That said, it’s still a lot of data to organize. There are seven data structures based on the model, measurements, fit parameters (BPMs, corrector magnets, and other parameters), LOCO setup variable, and LOCO outputs. For accelerators that use the Matlab Middlelayer (MML) for high level control, this process has been automated to a large extent.

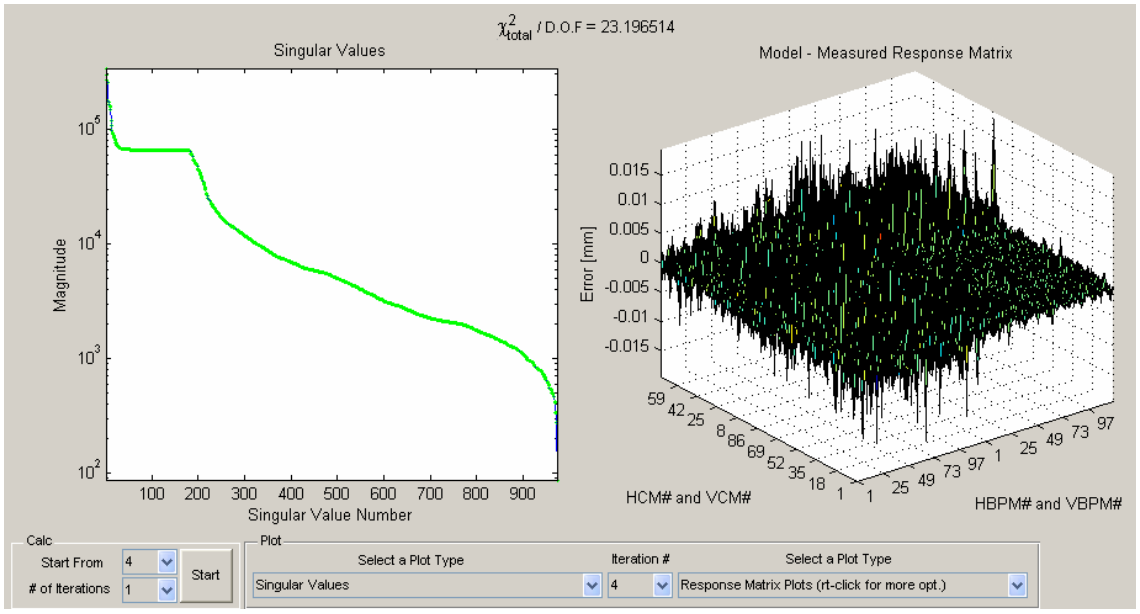
The most complicated data structure is the one that specifies which parameters in the model to fit. It needed to be very flexible so that anything can be varied. There is a function (mkparamgroup) to help build this structure for the most common parameters, like normal and skew quadrupole gradient. A fit parameter can be linked to one element in the model, like a magnet, or effect many elements in the model, like a power supply connected to a series of magnets.

1.1.4 LOCO Output and Plotting Options

Many papers have been written showing successful applications of the LOCO algorithm, [1][6][7][8][9] to name a few. This section will show some of the plotting options available in the LOCO GUI, using examples from various applications.

1.1.4.1 Goodness of Fit

There are a number of quick checks often made after completing a LOCO run. Figure 2 shows the singular values and a "chi-by-eye" check on the accuracy of the fit. Problems with individual corrector magnets or BPMs often are very apparent in the response matrix error plot. Figure 3 shows a histogram of the initial response matrix error and fourth iteration. Note that the BPM errors are usually rather small compared with the systematic errors, so a χ^2 per degree of freedom of 23 (or greater) is often still a usable fit.


χ^2_{total} / D.O.F = 23.196514

Calc: Start From 4; # of Iterations 1

Plot: Select a Plot Type: Singular Values; Iteration # 4; Select a Plot Type: Response Matrix Plots (rt-click for more opt.)

Figure 2: Singular Values and Response Matrix Error.

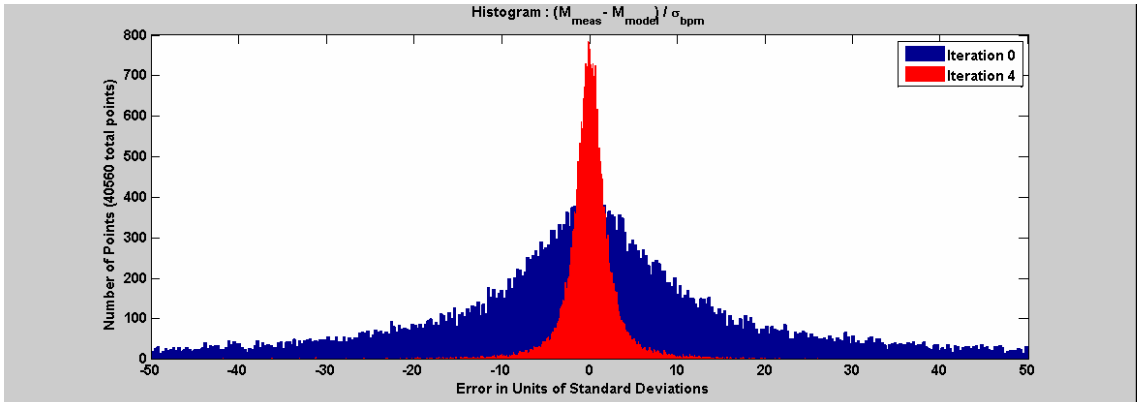
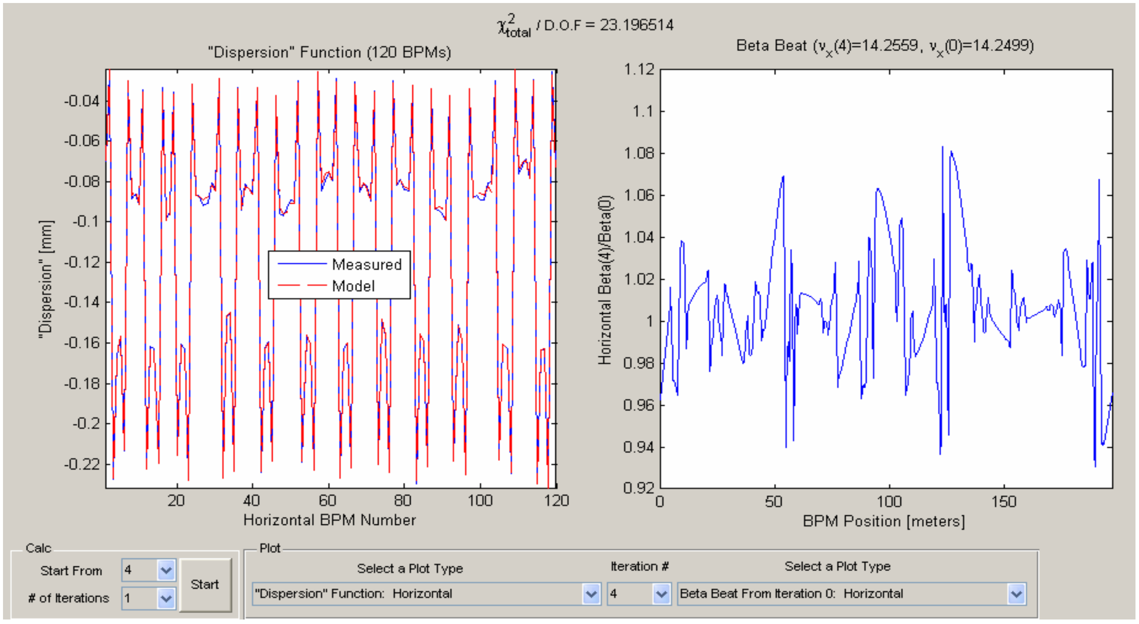


Figure 3: Histogram of the Response Matrix Error.

The dispersion function really needs to be inspected manually. The dispersion is a relatively small part of the response matrix, so it's possible to have a reasonably small χ^2 and still have a fitting problem with the dispersion, depending on the weight applied to the dispersion fit. Figure 4 shows a reasonably good fit for this ALS example. Note that when fitting energy shifts at the correctors, the model and measured dispersion may not match. The beta beat plot reflects how well the accelerator is calibrated to model.



Start From: 4
# of Iterations: 1
Select a Plot Type: "Dispersion" Function: Horizontal
Iteration #: 4
Select a Plot Type: Beta Beat From Iteration 0: Horizontal

Figure 4: Dispersion and Beta Functions.

1.1.4.2 BPM Gain and Coupling

Figure 5 shows the vertical BPM gain and coupling fits for the ALS. A 10% gain error seems to be quite common at light sources. The coupling error of -.7 on BPM(78) is horrible. The cause was later discovered to be an attenuation problem with one of the BPM buttons.

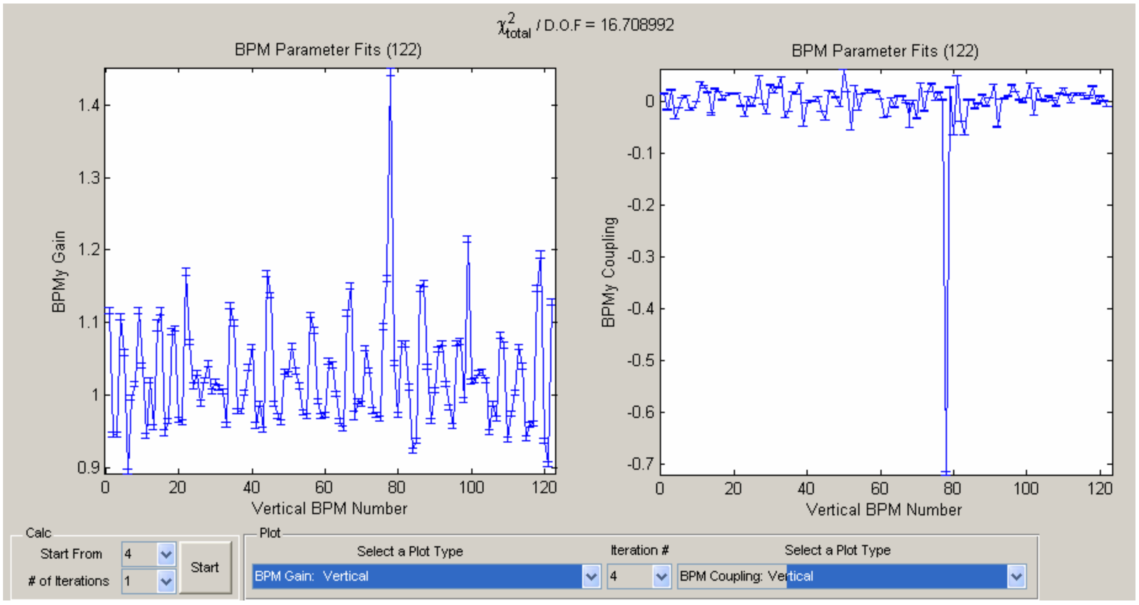


Figure 5: BPM Problem.

1.1.4.3 χ^2 Change vs. Fit Parameter

Figure 6 shows the change in χ^2 for each fit parameter and groups of parameters for the first iteration of a Gauss-Newton and Levenberg-Marquardt run (same measurement data). This plot provides some insight into what is going on in the multivariable minimization. If a particular fit parameter makes a large change in χ^2, then it usually means this parameter is fighting (in a degeneracy sense) with other parameters. If the χ^2 is not converging, then the parameter change could be projecting the fit into a nonlinear region. In either case, reducing the amount of parameter change per iteration often mitigates these problems. Adding a parameter cost or using the Levenberg-Marquardt method is a way to accomplish this goal, [7-9].

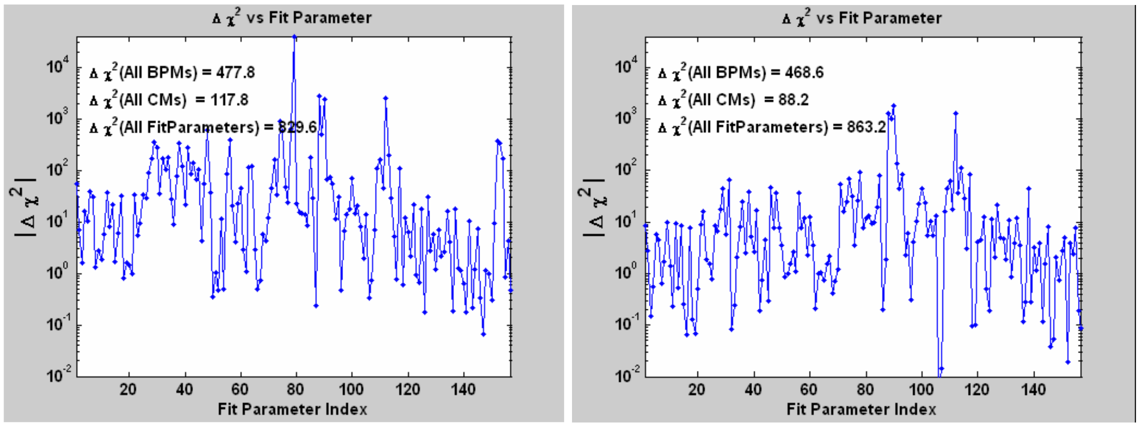
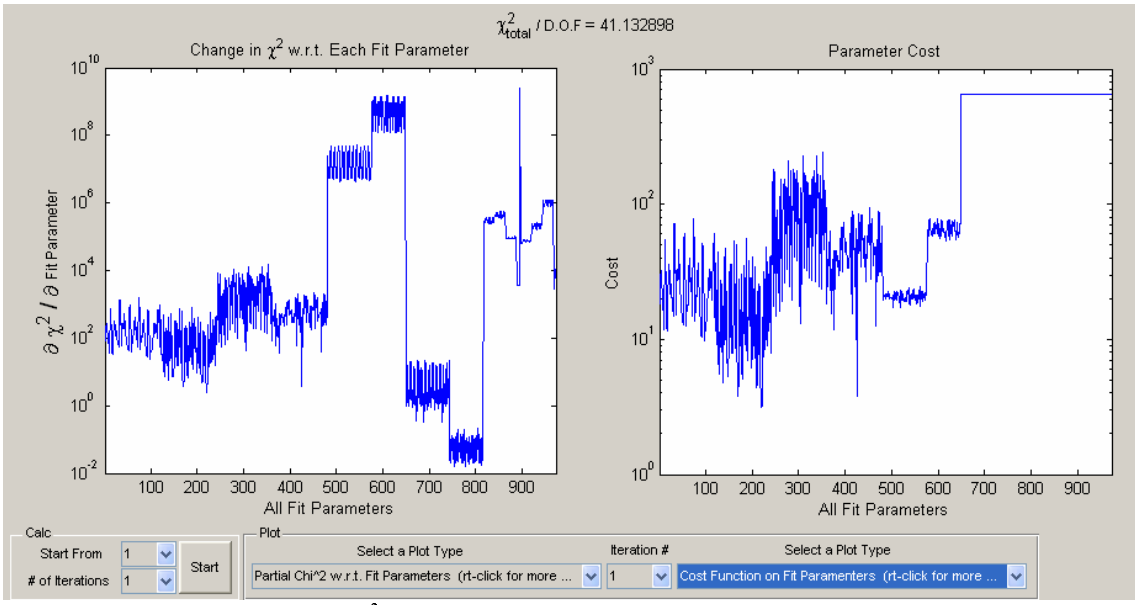


Figure 6: Gauss-Newton (Left), Levenberg-Marquardt (Right).

The Levenberg-Marquardt method is actually numerically equivalent adding parameter costs, Figure 7 (the cost function is flat for parameters 649 to 973 due to normalization). The difference between Levenberg-Marquardt and adding parameter costs is only in how the costs are chosen. The Levenberg-Marquardt method was developed to converge to the global minimum reliably. The standard Levenberg-Marquardt added costs according to their sensitivity on χ^2 but then scales the cost to adjust the rate of convergence. The scaling is adjusted to increase the rate of convergence when the solution is close to minimum. When choosing parameter costs manually, the ideal goal is to find a set of costs to minimize the effects of degeneracy between the parameters. The χ^2 vs. fit parameter plot can be used to choose that cost.



Start From 1
# of Iterations 1

Select a Plot Type: Partial Chi^2 w.r.t. Fit Parameters (rt-click for more ...
Iteration # 1
Select a Plot Type: Cost Function on Fit Paramenters (rt-click for more ...

Figure 7: Partial of χ^2 w.r.t. Fit Parameter and the Equivalent Cost Function.

1.1.5 Code availability

The MATLAB LOCO is available on the web or contact safranek@slac.stanford.edu or gjportmann@lbl.gov. A web search on "portmann loco" also seems to find the right place.

1.1.6 Acknowledgments

The authors thank A. Terebilo, C. Steier, L. Nadolski, W. Wittmer, D. Robin, J. Corbett, and M. Spencer for the many important discussions and contributions over many years.

1.1.7 References

1. J. Safranek, "Experimental Determination of Storage ring Optics using Orbit Response Measurements", Nucl. Inst. And Meth. A388, 27 (1997).
2. J. Safranek, G. Portmann, A. Terebilo, and C. Steier, "Matlab-based LOCO", EPAC'02, Paris, June 2002, pp. 1184-1186.
3. J. Corbett, G. Portmann, and A. Terebilo, "Accelerator Control Middle Layer", PAC'03, May 2003, Portland, pp. 2369-2371.
4. L. Nadolski et al., "Control Applications for SOLEIL Commissioning and Operation", EPAC'06, Edinburgh, July 2006, pp. 3056-3058.
5. A. Terebilo, "Accelerator Toolbox for Matlab", SLAC-PUB8732, May 2001,

http://www-ssrl.slac.standford.edu/at.

6. See contribution by J. Safranek in this ICFA beam dynamics newsletter.
7. See contribution by X. Huang in this ICFA beam dynamics newsletter.
8. See contribution by L. Nadolski in this ICFA beam dynamics newsletter.
9. L. Yang, et al, “A new code for orbit response matrix analysis”, Proceedings of PAC07, Albuquerque, NM.