Doing Accelerator Physics Using SDDS, UNIX, and EPICS[1]

M. Borland, L. Emery, N. Sereno
Argonne National Laboratory; 9700 S. Cass Avenue; Argonne, Il 60439

Abstract

The use of the SDDS (Self-Describing Data Sets) file protocol, together with the UNIX operating system and EPICS (Experimental Physics and Industrial Controls System), has proved remarkably powerful during the commissioning of the APS (Advanced Photon Source) accelerator complex. The SDDS file protocol has permitted a tool-oriented approach to developing applications, wherein generic programs are written that function as part of multiple applications. While EPICS-specific tools were written for data collection, automated experiment execution, closed-loop control, and so forth, data processing and display are done with the SDDS Toolkit. Experiments and data reduction are implemented as UNIX shell scripts that coordinate the execution of EPICS-specific tools and SDDS tools. Because of the power and generic nature of the individual tools and of the UNIX shell enviroment, automated experiments can be prepared and executed rapidly in response to unanticipated needs or new ideas. Examples are given of application of this methodology to beam motion characterization, beam-position-monitor offset measurements, and klystron characterization.

1 The SDDS Protocol and Toolkit

The Self-Describing Data Set file protocol and program toolkit are an essential part of accelerator experiments at APS. The usefulness of SDDS stems less from the data protocol itself than from the large number of existing generic SDDS-compliant programs, referred to collectively as the "SDDS Toolkit." The file protocol and the toolkit are described in detail in the references [1, 2].

Self-describing data (SDD) are data that are referred to and accessed by name only. SDD usually have metadata that are available, again by name only. We believe the limited application of SDD to date is related to attempts to achieve excessive generality. With increasing generality comes increasing difficulty in application development and use, and longer development times. An overly general protocol makes it difficult to create generic tools, losing the principle motivation for SDD.

SDDS protocol is general enough to be useful, but simple enough to be usable. The data model permits an arbitrary number of "data pages" of homogeneous structure. The structure defines an arbitrary collection of parameters (i.e., single values), multidimensional arrays, and a data table. Each page is an instance of this structure. The various elements of the page are available via calls to a library of C routines, as is metadata about the elements (e.g., units, data type, number of array entries). Various data types (e.g., string, floating-point, integer) are supported.

SDDS files may be either binary or ASCII format. The ASCII variant is simple enough that input data can be readily created "by hand" when required. Data from non-compliant programs can often be converted with the addition of a simple file header.

While the SDDS protocol is able to store most data conveniently, using it would not be advantageous if there were no tools available to operate on the data. The SDDS Toolkit is a group of over 40 independent programs that accept SDDS files as input. Almost all of the programs also create SDDS files as output. This means that complicated data processing sequences can be created by combining the function of several programs, without worrying about whether one program can

[1]Work supported by U.S. Department of Energy, Office of Basic Energy Sciences under Contract No. W-31-109-ENG-38.

use the output of another or create output suitable for another. Such sequences may use intermediate files or they may be constructed as pipelines. Applications are made more independent of each other, in that the presence of additional data in a data stream will not break an application. Further, applications can verify the validity of the data they are given, e.g., by checking for the presence of data, checking units, or checking the data type.

Using SDDS, individuals may add programs for use in their own sequences without restriction. Individuals may create shell scripts of data processing sequences, and combine these scripts without restriction. Centralized control of such activity is neither required nor desirable, in contrast to the now common all-in-one approach to data processing applications.

For a summary of many of the currently-available SDDS Toolkit programs, the reader may consult reference [2]. Some of the following sections assume the reader is familiar with this reference.

2 SDDS-Compliant EPICS Tools

The Experimental Physics and Industrial Control System[3] (EPICS) is used at APS and at a number of other accelerator facilities. At APS, with very few exceptions, SDDS is used for collection and processing of all commissioning data, both from EPICS and from other sources.

For example, SDDS is used for the following controls applications: saving, restoring, and comparing machine configurations; configuring composite knobs; storing magnet conditioning instructions; saving and restoring GPIB device configurations; storing data from digital oscilloscopes and spectrum analyzers; archival data logging; collection of data during automated experiments; and configuration data for generalized feedback on process variables (e.g., orbit correction).

Using the same philosophy as for the SDDS Toolkit, we have developed generic programs that interact with EPICS while using SDDS files for input and output. That is, while the tools take data from EPICS, they are configured by SDDS files and produce SDDS files as output. Hence, we have not needed to write any EPICS-specific data processing or display programs, but have simply used the SDDS Toolkit.

For a summary of many of the currently-available SDDS-Compliant EPICS tools, the reader may consult reference [2]. Some of the following sections assume the reader is familiar with this reference.

3 Using SDDS, EPICS, and UNIX for Experiments

All of the SDDS programs can, of course, be executed from the commandline. They can also be executed via a button-push in, for example, a Tcl/Tk [4] GUI application. More importantly, script languages such as `csh` and `tclsh` provide a ready-made environment within which experimenters can combine the power of many programs. This provides the capability of making complicated data collection and processing scripts on demand. Any particular script may make use of other scripts as component commands. Scripts may invoke any of the multitude of powerful UNIX commands (e.g., grep, tr), and they may make use of built-in features of the script language, such as flow control and mathematical expression evaluation.

For APS commissioning, SDDS-compliant EPICS tools provide generic data gathering and control functions. Instrument-specific SDDS-compliant tools provide for control and integration of data from "off line" sources such as portable oscilloscopes or spectrum analyzers. Data analysis involving all of these sources is performed using the SDDS toolkit. Frequently, these capabilities are incorporated into scripts written on-the-fly by experimenters on shift. For a basic introduction to working with SDDS within the EPICS and UNIX framework, see reference [5].

The following is a list of some of the accelerator physics experiments and observations that have been done to date using only UNIX scripts and the SDDS tools: response matrices for closed orbits and trajectories; dispersion and chromaticity measurement by variation of rf frequency; integer tune measurement from difference orbits; beam motion characterization (see below); transverse tunes vs beam current; lifetime and vacuum system response to variations in bunch pattern and beam current; rf voltage calibration using the synchrotron tune; longitudinal acceptance measurements; beam-excited higher-order-mode searches of rf cavities; injection kicker pulse shape measurement using beam and a fluorescent screen; pulsed power supply amplitude stability and jitter; beam lifetime vs current, scraper position, and bump height; beam-position-monitor (BPM) offset measurements (see below); automated determination of bad BPMs; klystron gain characterization (see below); linac efficiency optimization scans; linac bunch length measurement using back-phasing; linac energy gain calibration vs input power, temperature, and frequency; linac emittance measurements.

There is a trend in control systems toward using graphical user interfaces (GUIs), which is clearly desirable for routine operations. However, for experimental work, GUIs can be a serious impediment when used exclusively. There are several reasons for this, which motivated our approach.

First, such work is by definition open-ended, while GUIs almost by definition confine the user to preconceived choices. Second, experimental work is often procedural, whereas GUI applications are user-driven; a GUI that permits designing and executing *general* procedural experiments is essentially a script composition tool, the power of which is dependent entirely on the capability of the script language and the power of individual commands. Third, GUI applications usually cannot be coordinated, as the user is required to provide direction to each GUI using the mouse. Ideally, one shouldn't need to run a particular GUI interface in order to use a particular algorithm (e.g., orbit correction). The algorithm should be available conveniently through a script language, and not simply through a subroutine call in a compiled program.

In the following sections, we provide several detailed case studies that illustrate our approach. These are arranged in order of increasing complexity. The reader should note how data is taken through multiple stages of processing, with each stage frequently reusing the programs invoked in early stages for different specific tasks. Note also how the distinct applications use many of the same basic tools. Everything that is discussed in the examples is done entirely using UNIX scripts and SDDS tools; there are no manual steps in any of the data processing.

4 Example: Beam Motion Characterization

Since the APS has a tight tolerance for the electron beam oscillatory motion, it is important to first characterize the beam motion and then locate any sources of such motion. These are typically magnet power supply ripple and magnet girder excitation. Rather than manually screening every power supply and girder with portable instruments, one can obtain information from beam observations and experimentation. At APS, we gather raw beam position monitor (BPM) data, then process the data down to its essentials with UNIX shell scripts of SDDS commands, all in a few minutes. In this section we will describe how scripts and SDDS tools were used in the course of identifying and locating a 6.5-Hz beam motion in the storage ring.

We use the raw BPM data obtained by the "slow" BPM history modules. These modules collect up to 4080 readings at a 60-Hz rate, each reading being the average of 2048 turns. Forty synchronized BPM histories, one from each of 40 storage ring sectors, can be acquired in this way. A script prepares the BPM history modules and the local input/output controllers (IOCs) for the acquisition, sends a trigger signal to all IOCs simultaneously, waits for the 4080-point history buffers to fill, and downloads the history buffer waveforms to an SDDS file using `sddswmonitor`.

The script makes heavy use of some channel access commandline tools, such as **cavput**, which assigns values to lists of process variables, and **cawait**, which waits for process variable values to fulfill specified conditions before executing specified actions. Since these channel access tools don't deal with files, they are not SDDS tools, but they have become indispensable in automating data acquisition. Acquiring 40 BPM histories and downloading them to a file takes about 100 seconds. To do all 360 BPMs, the script repeats the acquisition nine times, once for each BPM in a sector, producing nine files, and taking about sixteen minutes.

The data from the slow beam histories is processed in several ways, among which are the following:

- Calculation of the standard deviation, (σ_x), of each BPM history: Each raw data file contains a data column corresponding to a BPM history. After using **sddssmooth** to determine and remove the slowly-varying components, one can use **sddsprocess** to create for each BPM data column a new parameter whose value is equal to the standard deviation of the corresponding column. The sequence of the commands **sddscollapse**, **sddstranspose**, and **sddsconvert**, with appropriate options, transforms the file containing parameters into a new file of one string and one double data column named BPM and StdDev, for instance, which can be plotted with **sddsplot**.

  It is useful to plot the quantity $\sigma_x/\sqrt{\beta_x}$, where β_x is the horizontal beta function obtained from an optics modeling program output file. With the use of **sddsxref**, the data column β_x is transferred from the optics program output file to the file with the columns BPM and StdDev mentioned in the previous paragraph; **sddsxref** is able to match the BPM names from the measurement and the optics program in order to get the correct value of β_x for each BPM. Once this is done, the new column for $\sigma_x/\sqrt{\beta_x}$ is computed using **sddsprocess**. Plotting this result revealed a function that was constant around the ring, a sure indication of many sources of beam motion.
- Calculation of the beam motion spectrum for each BPM: The BPM raw data file is Fourier analyzed with the **sddsfft** command. The output file contains one FFT data column per BPM. Each spectral peak corresponds to a source frequency. A spectrograph plot of the power spectral density (PSD) as a function of BPM is produced using **sddscontour** to reveal a pattern in the peaks; such a plot is shown in Figure 1. One can see the ubiquitous 6.5-Hz motion, plus a hint of 12-Hz motion due to a girder resonance. Continuing the analysis, the PSD within various frequency bands is summed for each BPM's FFT using **sddsprocess**, and then plotted as a function of BPM name using **sddsplot**. When the power in the band from 5.5-7.5 Hz is normalized to $1/\sqrt{\beta_x}$ as per the previous paragraph, the data shows the 6.5-Hz line is also caused by many sources.

As the 6.5-Hz frequency of the dominant horizontal motion did not correspond to any known girder resonance, we investigated power supply sources. Power supply readbacks were acquired and analyzed using **sddsstatmon** at a hardware-limited 1-Hz rate. The output file contained one standard deviation data column for each supply monitored. A list of worst offenders was drawn up by sorting the rows of the file with **sddssort** according to the value of the standard deviation. Some of the worst supplies were analyzed locally with a spectrum analyzer, revealing 6.5-Hz lines in some sextupoles.

Before making any modifications to these supplies, we performed a series of before-and-after experiments using the BPM histories. In each case we were able to determine within a few minutes whether the beam motion changed due to, say, turning off a family of sextupoles. Figures 2 and 3 show plots of the reduced data from one of the experiments, revealing that the sextupole magnet convertors were the sources of the 6.5-Hz beam motion.

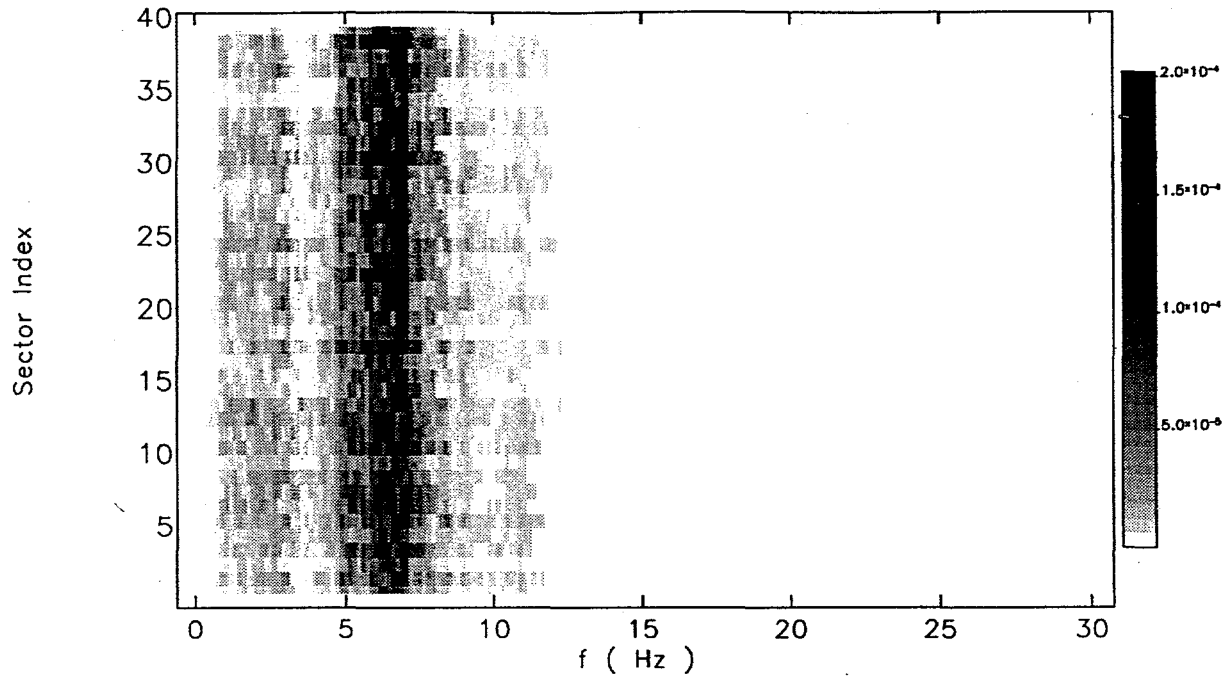

Figure 1: PSDs of orbit motion at one BPM per sector with 6.5-Hz motion present.

5 Example: Linear Accelerator Klystron Gain Measurement and Analysis

The purpose of this application is to measure the gain of the APS linear accelerator (linac) klystron high power amplifiers and determine from the data an empirical formula relating the klystron parameters. The klystron amplifier transforms a low-power rf signal at its input to a high-power rf signal at its output. In addition, the output power level depends on the tube voltage applied between the klystron cathode and anode. The application consists of using **sddsexperiment** to measure the klystron output power as a function of input power for various tube voltages. Subsequent data analysis is carried out using the SDDS Toolkit.

Data analysis consists of fitting an appropriate function to the measured output power data for a given tube voltage. The chosen fitting function,

$$P_{out} = a_1 + a_2 e^{-a_3 P_{in}}, \tag{1}$$

reproduces the variation of output power with input power at a given tube voltage. For each tube voltage V, the fit will be slightly different and the fit coefficients in equation 1 can be considered functions of the tube voltage V. Each coefficient is then fit according to a simple linear function with a slope and intercept. In this way, equation 1 yields an empirical formula for the klystron output power as a function of input power and tube voltage.

Data acquisition was done using `sddsexperiment`. The program used EPICS process variables for drive and input power as well as tube voltage defined for each klystron. The experiment was done using two loops, where the tube voltage setpoint was varied in the outer loop and the input power was varied by an inner loop. A complication arises in this measurement because the input

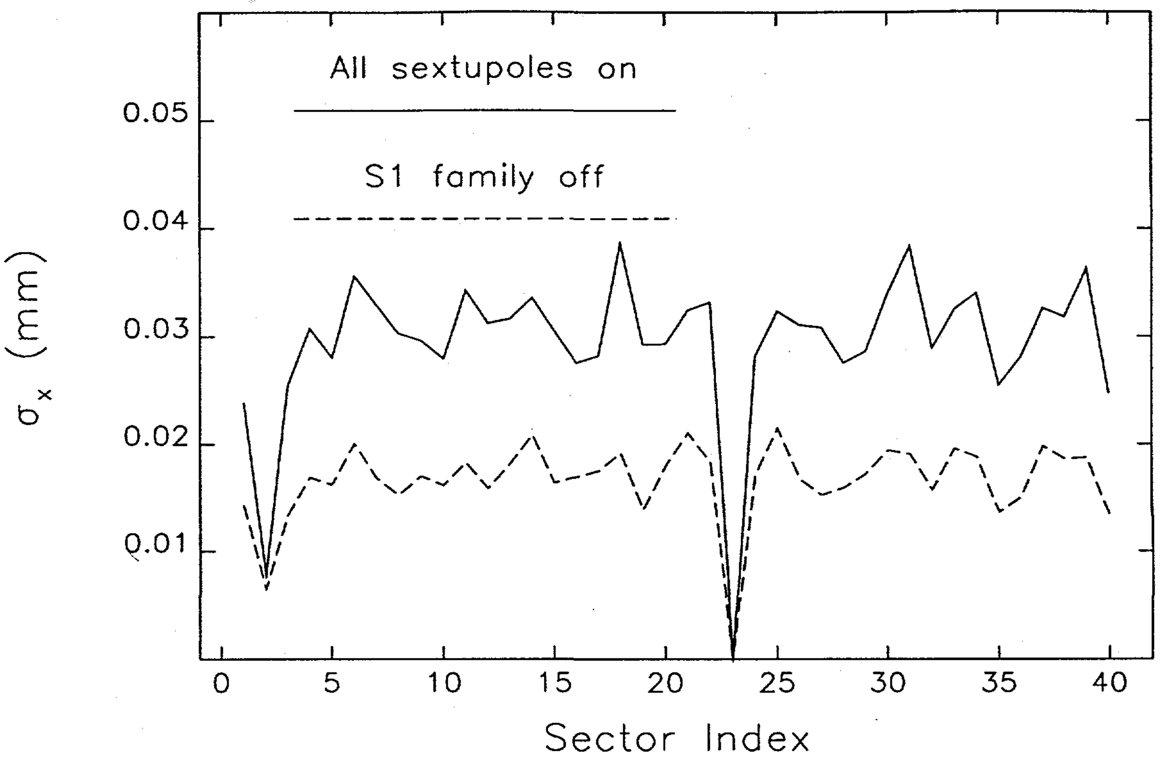

Figure 2: rms orbit motion as a function of BPM around the ring for two lattice configurations.

power is actually indirectly adjusted via a programmable attenuator. A C-shell script was written to manually adjust the attenuator until a given target drive power was reached. `sddsexperiment` invokes the script in the inner loop as a subprocess to vary the drive power.

Figure 4 shows the data taken for the "sector L4" klystron as well as the best fit to the data using equation 1. The voltage shown in the legend is actually the pulse forming network voltage (PFN) that controls the tube voltage. The PFN voltage is stepped up by a scale factor of seven to the full tube voltage by a transformer.

Data analysis is performed according to the following procedure using various Toolkit programs in a C-shell script: 1. Sort the data according to increasing PFN voltage V_p using `sddssort`. 2. Break the data into pages using `sddsbreak` according to changes in V_p. 3. Compute the average V_p for each gain curve in Figure 4 using `sddsprocess`. 4. Perform the exponential fit given by equation 1 using `sddsexpfit`. 5. Remove outlier data points beyond two standard deviations from the best fit line using `sddsoutlier`. Eliminated data is apparent from gaps in Figure 4. 6. Perform an exponential fit on the data remaining after outlier elimination. 7. Use `sddsxref` and `sddscollapse` to collect the fit coefficients and PFN voltages into two data columns in an SDDS file. 8. Perform a linear fit to the coefficient vs V_p data using `sddspfit`.

Figure 5 shows the coefficient a_3 as a function of V_p along with a linear fit. The slope and intercept of the fit are displayed on the figure using `sddsplot`'s ability to display string parameters from files as part of plot labels. The other two coefficients show a similar linear dependence. The slope and intercept for each coefficient curve along with equation 1 yields the desired empirical relationship between the klystron output power, input power, and tube (PFN) voltage.

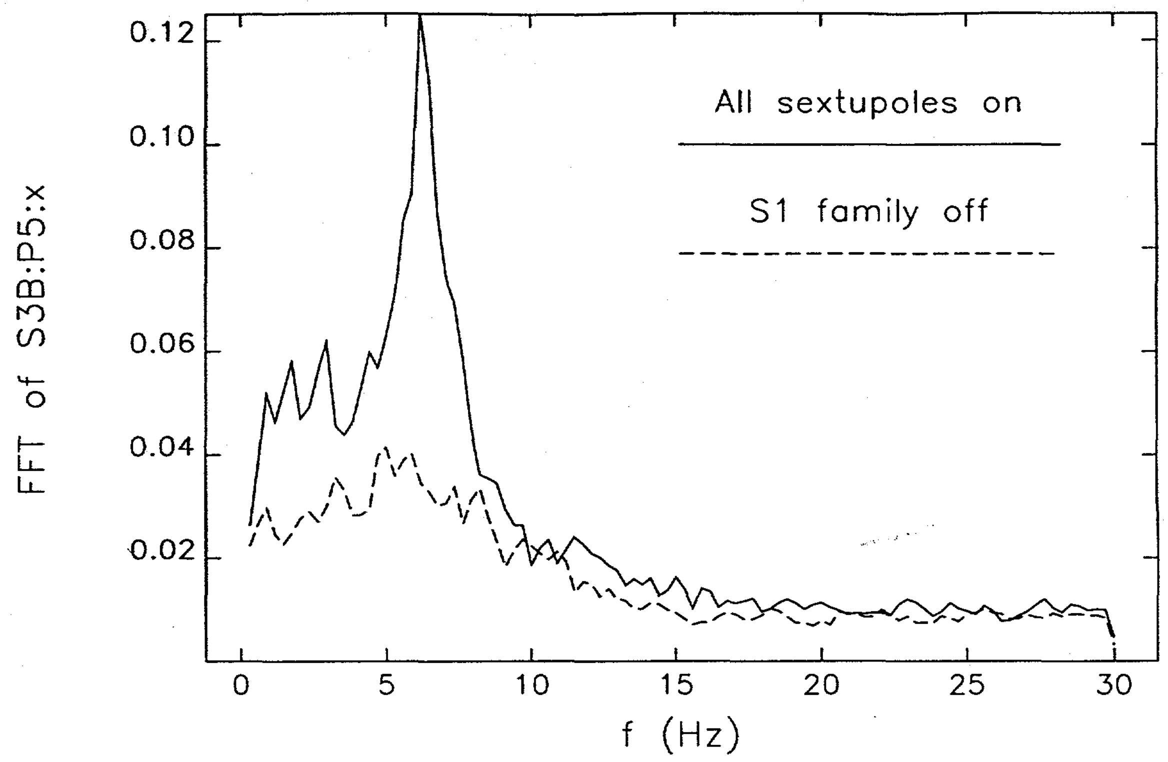

Figure 3: FFT of orbit motion at one BPM for two lattice configurations.

6 Example: BPM Offset Measurements

In order to obtain optimum performance from a third-generation light source like the APS, it is important to accurately center the beam in the magnetic elements. Many APS beam position monitors (BPMs) are located close to quadrupoles; hence, we calibrate the centers of such BPMs relative to the centers of adjacent quadrupoles. Initially, the beam is steered to center it in the BPMs as well as possible. If it passes off-center through a quadrupole magnet due to a position offset in a BPM, a small quadrupole strength change will produce an orbit shift at every BPM, the shift being linear in the change and in the offset. By finding the beam position for which no shift occurs, one determines the quadrupole-center-to-BPM-center offset.

We found that taking a single measurement of the orbit change for a single quadrupole change was very noise-sensitive. Instead, we varied a beam bump centered on the quadrupole of interest, which varies the effect of quadrupole strength changes. Taking data for several bump positions permitted inferring the offset from a set of linear fits.

The measurement script takes as an argument the name of the quadrupole for which the offset is desired. While the entire experiment could be done within a single `sddsexperiment` run, it is convenient to break it into two nested runs, since this allows testing and use of the inner loop in stand-alone fashion. In the outer loop, the beam position in a quadrupole is varied by $\sim \pm 2.5$mm in five steps. In the inner loop, the quadrupole strength is varied by $\sim \pm 2\%$ in 11 steps. The outer loop is implemented as a script that prepares input for `sddsexperiment` and executes it. This `sddsexperiment` run in turn repeatedly executes another script, which itself prepares input for a second `sddsexperiment` run and executes it, thus implementing the inner loop. It is in the inner loop that BPM data are collected. For each of the 55 points on the two-dimensional experimental grid, 20 readings are averaged for each of 80 BPMs spaced around the ring, meaning that 88,000

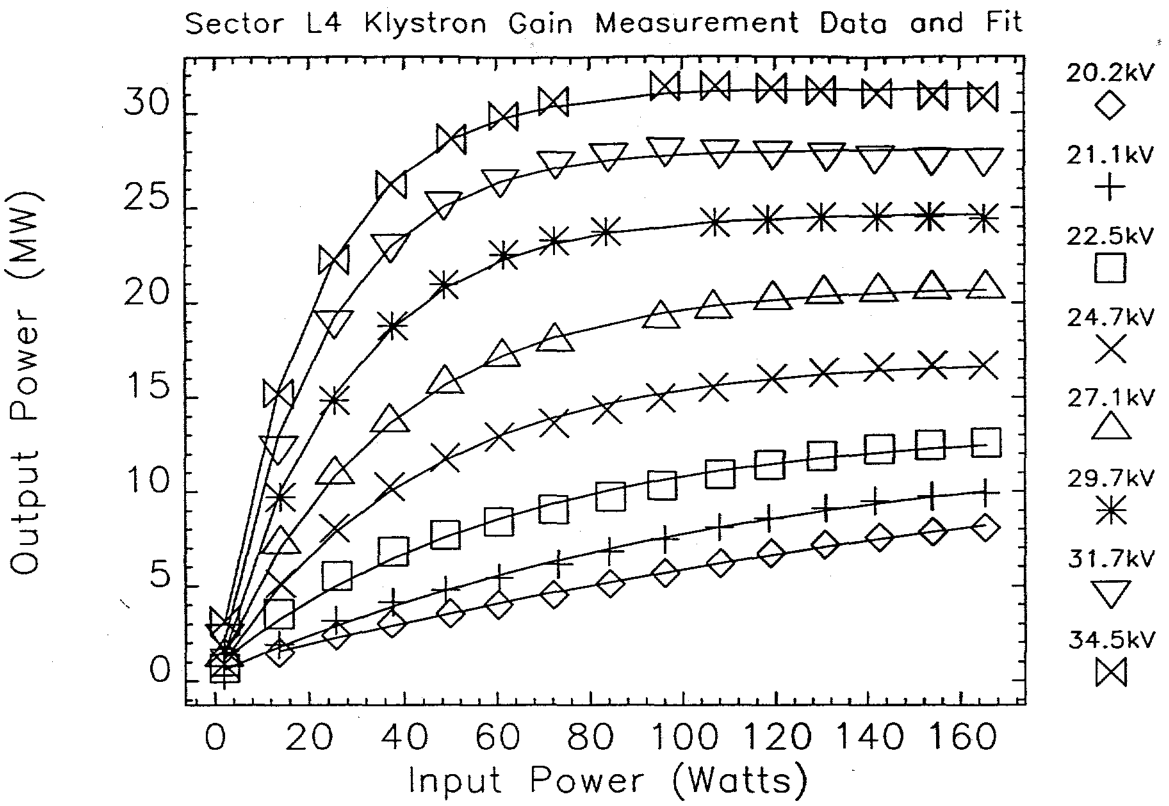

Figure 4: Measurements and fits for klystron output power vs input power for various PFN voltages.

readings go into each offset result.

The data processing script uses `sddsslopes` to find, for each bump height, the slope of the orbit change at each BPM with respect to the change in quadrupole strength; we call these the "quad slopes." After using `sddscombine`, `sddscollapse`, and `sddstranpose` to collate and reorganize the data, the script then applies `sddsslopes` again to fit lines to the quad slopes as a function of measured bump height. `sddsprocess` is then used to compute 80 estimates of the offset (one for each of the BPMs used) and an error estimate for each. The script performs outlier elimination on the offsets using `sddsoutlier`, then uses `sddsprocess` to compute the error-weighted mean offset (from the accepted data) and an estimate of its error.

The script also produces several graphics, allowing the experimenter to evaluate the quality of the data. Figure 6 shows one of these, a histogram of the offsets derived from each BPM, along with a title giving the mean offset; the title text is composed by `sddsprocess` in the same invocation that computes the results, then extracted from the file by `sddsplot` for display with the data. Figure 7 shows another, a display of the individual quad slopes along with the corresponding offset errors; this illustrates that, as expected, BPMs with quad slopes that are larger in magnitude have smaller errors. Using the ability of `sddsprocess` to evaluate complicated user-defined equations involving data in a file, it was possible to compute the error-weighted mean offset, which makes use of even the less certain data to provide an improved result.

Once many offsets are measured, they are collated into a single data file using `sddscombine` and `sddscollapse`. This permits analyzing the final measured offsets as a group, or sending them to the controls system for subtraction from BPM readings.

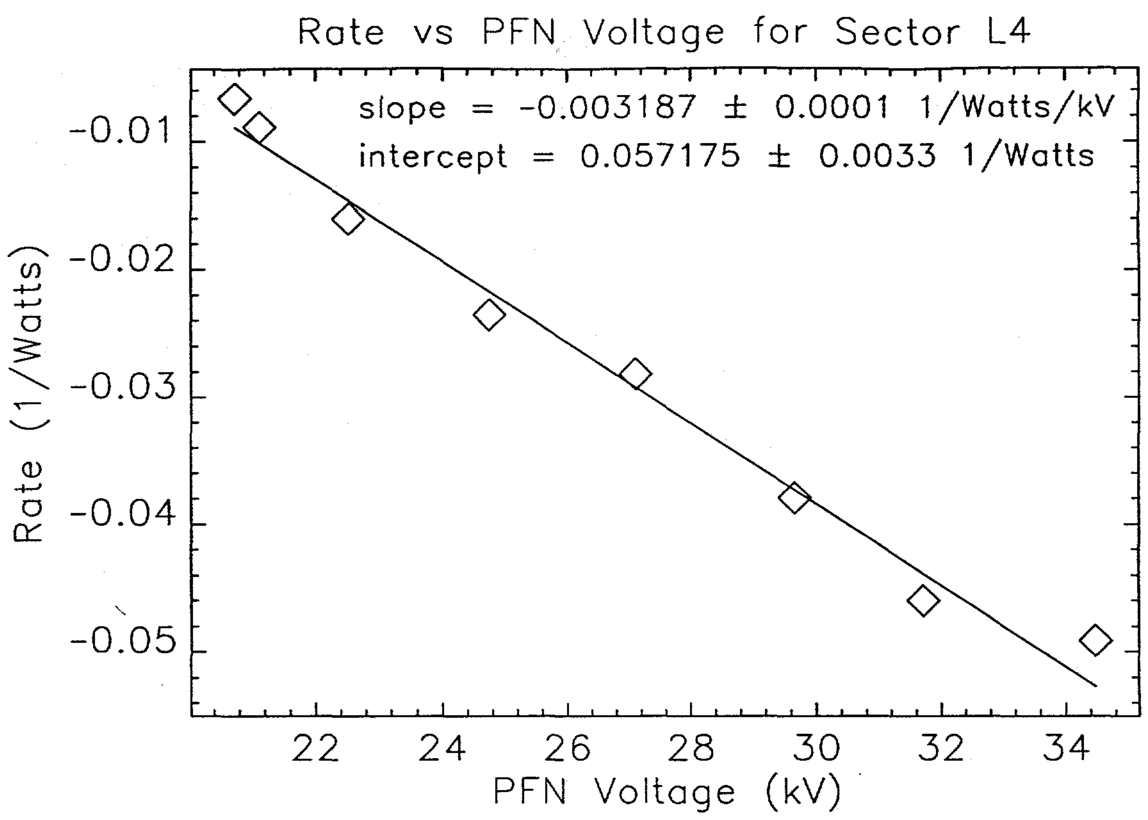

Figure 5: Exponential decay constant of fit as a function of PFN voltage.

7 Acknowledgements

The authors wish to acknowledge the following individuals for contributions to the SDDS toolkit and EPICS-specific tools, either in the form of suggestions, bug discoveries, or contributed programs: J. Carwardine (APS), Y. Chung (APS), K. Evans (APS), N. Karonis (FNAL), E. Lessner (APS), S. Milton (APS), G. Rinehart (IPNS), C. Saunders (APS), M. White (APS).

References

[1] M. Borland, "A Self-Describing File Protocol for Simulation Integration and Shared Post-processors," Proceedings of the 1995 Particle Accelerator Conference, May 1-5, 1995, Dallas, Texas, to be published.

[2] M. Borland, L. Emery, "The Self-Describing Data Sets File Protocol and Toolkit," these proceedings.

[3] L. R. Dalesio, M. R. Kramer, A. J. Kozubal, "EPICS Architecture," in *ICALEPCS* 1991, pp. 278–281.

[4] J. K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley, 1994.

[5] J. A. Carwardine, "An Introduction to Plant Monitoring Through the EPICS Control System," these proceedings.

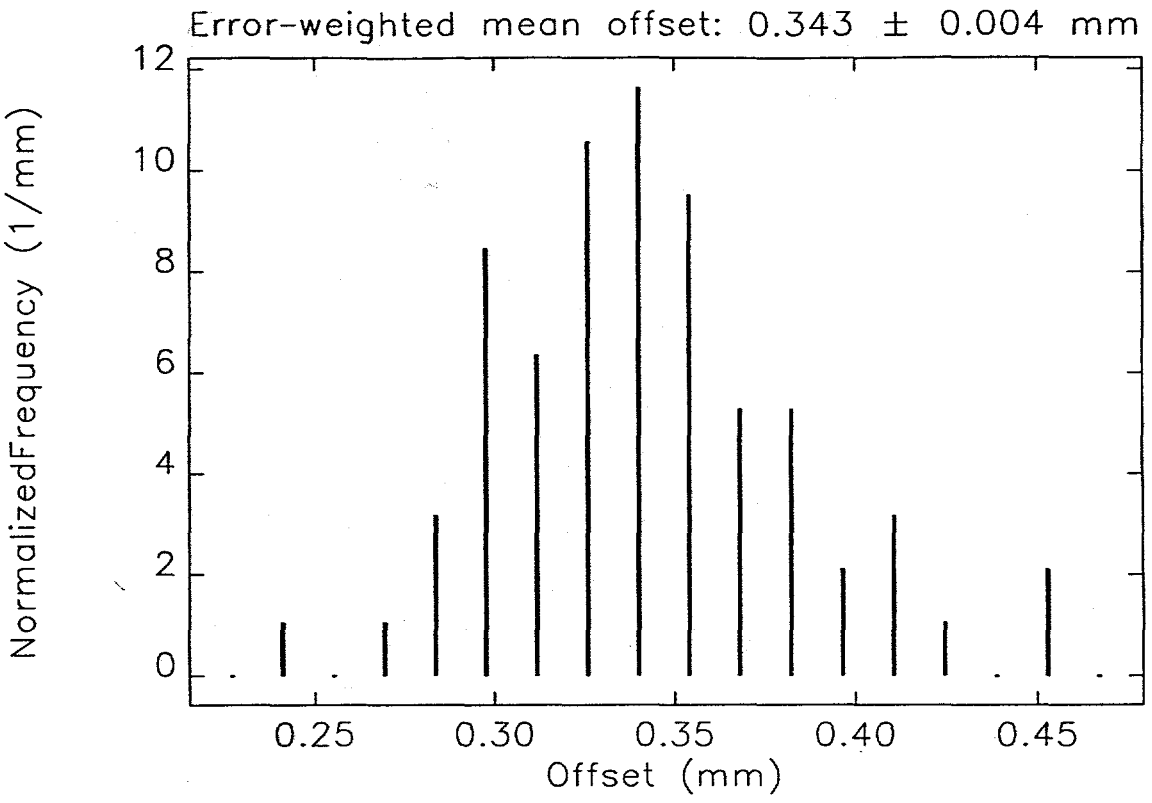

Figure 6: Histogram of particular BPM-to-quadrupole offset as estimated from numerous BPMs.

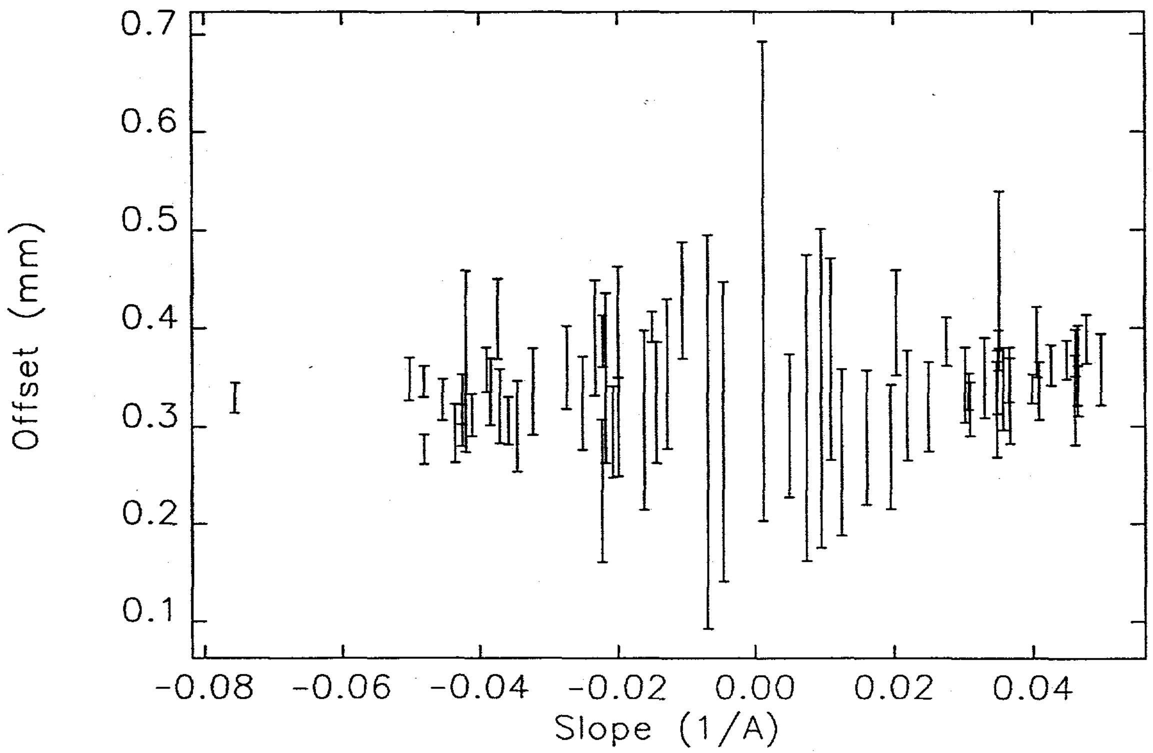

Figure 7: Estimates of BPM-to-quadrupole offset and "quad slopes" from numerous BPMs.