A Python Instrument Control and Data Acquisition Suite for Reproducible Research

Lucas J. Koerner, *Member, IEEE*, Thomas A. Caswell, Daniel B. Allan, and Stuart I. Campbell

***Abstract*—Tools that standardize and automate experimental data collection are needed for greater confidence in research results. The National Synchrotron Light Source-II (NSLS-II) has generated an open-source Python data acquisition, management, and analysis software suite that automates x-ray experiments and collects an experimental record that facilitates complete reproducibility. Here we show that the NSLS-II tools are not only useful for x-ray science at large-scale facilities by presenting an add-on package that adapts these tools for use in a small laboratory with common physics and electrical engineering instruments. The composite software suite eases and automates the execution of experiments, records extensive metadata, stores data in portable containers, and speeds analysis through tools for comprehensive searches. In total, this software suite increases the reproducibility of laboratory experiments. We demonstrate the software via the evaluation of two lock-in amplifiers — the miniature ADA2200 and the ubiquitous SRS SR810. The frequency resolution, signal-to-noise ratio, and dynamic reserve of the lock-in amplifiers are measured and presented. The usage of the software suite is described throughout these measurements so that the reader can implement the tools in their lab.**

***Index Terms*—Computerized instrumentation, Demodulation**

I. Introduction

Research that can be reproduced is able to be recreated with exact agreement at different times and in different laboratories. Reproducible research has become essential for rigorous computational science and data analysis. Tools that facilitate reproducibility in these fields follow a literate programming paradigm, one that interweaves documentation with code. Examples include *knitr* [1], R Markdown [2], pweave [3], and Jupyter notebooks [4]. However, methods to support reproducibility of experiments during data *collection* are less developed. Automated and reproducible experimental flows have been developed in a few specific fields. For example, the integrated circuit industry relies upon automated test equipment (ATE) systems for chip testing before parts are packaged and sold [5]. Commercial startups have launched automated laboratories which allow biologists and chemists to outsource wet-lab experiments (Emerald Cloud Laboratory, Transcriptic, and Riffyn) [6]. However, a need exists for experiment control software for small labs, such as those in physics, engineering, or biology, without staff dedicated to software support. We address this need by presenting an open-source instrument control, data collection, and data management software suite that eases the process of data collection and facilitates reproducible experiments.

L. Koerner is with the Department of Electrical and Computer Engineering, University of St. Thomas, St. Paul MN, 55105, USA e-mail: (koerner.lucas@stthomas.edu).

T. Caswell, D. Allan, and S. Campbell are with the National Synchrotron Light Source II, Brookhaven National Laboratory, Upton, NY 11973, USA

This work builds upon the Bluesky package developed for data acquisition, management, and analysis at the National Synchrotron Light Source-II (NSLS-II) at Brookhaven National Laboratory [7], [8]. The Bluesky suite for experimental x-ray science incorporates rich metadata, integrates analysis with data collection, and provides generic experimental procedures that can be reused across a wide range of disciplines with different hardware. The NSLS-II is a large national x-ray user facility that supports around 1500 users per year from both academia and industry. Since Bluesky was designed as a general experimental control system, it is not limited to use at large x-ray laboratories. To validate the generality of Bluesky, we adapt it to a small electrical engineering research laboratory, using a new instrument control Python package instrbuilder.

Here, we demonstrate the utility and usage of the composite software suite through detailed experiments that characterize two lock-in amplifiers. One such example illustrates the basic use case of this Bluesky software suite (see Figure 1). Instruments are controlled using a computer interface: Bluesky provides typical experimental procedures, instrbuilder abstracts instrument specific commands to a generic language, and Bluesky automatically saves data and metadata that describe the results and configuration. In the example of Figure 1, to determine the frequency resolution of the lock-in amplifier, a function generator was controlled over USB and provided an input stimulus that was stepped through a range of input frequencies by Bluesky using generic commands in the form of "set frequency". The lock-in amplifier was configured and controlled over RS232. The lock-in output data was read with a generic command similar to "read magnitude". At the completion of the experiment, Bluesky generated data and metadata organized in key-value pairs that describe the entire experiment.

Similar experiment control software developments have been presented in literature. For example, the PLACE package for Python experimental control is open-source and modular, but, is not able to leverage the large x-ray source user community [9]. Further, an x-ray facility consists of many different beamlines with varying instrumentation and measurement needs, which ensures that the Bluesky suite is generalized and extensible. We also believe that the instrument driver generation methodology of our package instrbuilder allows for quicker implementation of new instrument drivers than the code based approach of PLACE. Other efforts in open-source instrument control appear to have been retired based on the code repository, for example [10]. Again, we anticipate that the large x-ray community will provide a critical mass of users to maintain this software package.

This software suite implements a list of best practices for experimental data collection. The practices and the benefits provided are as follows. The suite:

- Captures rich metadata that is human-readable, is searchable, fully describes the provenance of data, and records the configuration of each measurement. Rich metadata simplifies the process of repeating an experiment and improves accuracy when experimental results are reported.
- Saves data tagged with a unique identifier into containers, which are portable, extensible, and fully describe the data. Data portability ensures retrieval of archived experimental results; data with self-contained descriptors removes dependencies between data collection and data analysis.
- Generalizes experimental procedures. Abstraction allows for experimental design reuse and for the sharing of procedures with other laboratories, even those that use different equipment.
- Displays and analyzes data during acquisition. Real-time feedback allows the user to promptly verify an experimental setup.

The best practices described above simplify the setup and improve the reproducibility and consistency of experiments. In addition to the benefits above, a laboratory adoption of this standardized and automated experimental flow would better support information sharing between researchers by enforcing a consistent data collection process.

II. Bluesky Description

The NSLS-II Bluesky suite consists of a set of Python packages: Bluesky, ophyd, and databroker. The first, Bluesky, is the experiment engine and collects data. The second, ophyd, abstracts hardware so that all devices present a consistent interface to the experiment engine. The final package, databroker, retrieves and searches data and metadata from a variety of sources [7].

A. *Experimental Control and Data Collection: Bluesky*

The Python package Bluesky designs and runs experiments by communicating with hardware over the unified instrument interface provided by ophyd. Common experimental procedures, or plans, are prepackaged (e.g. *scan* to sweep one control signal); alternatively Bluesky allows users to develop custom plans. During the course of an experimental run Bluesky saves data and metadata. The data and metadata is represented using Python dictionaries with a specified organization, these dictionaries are referred to as *documents*. Each experimental run saves the following documents:

- A run start document which includes metadata available at the start of the run, such as time, Bluesky plan name, and a unique identifier (UID).
- A run stop document of metadata available at the end of a run, such as the completion status and end time.
- A document of measurement events, which includes values and timestamps of instrument readings. This document is saved to a database (e.g. sqlite) that is named with the experiment UID.
- An event descriptor document which describes the data in the event document and is used to prime data analysis tools before data is opened [11].

The Bluesky run engine streams data during collection so that callback functions, such as visualization (tables and figures) and data processing, are available live as the experiment progresses.

B. *Data Retrieval and Metadata Searching: databroker*

The Python package databroker provides an interface to retrieve stored experimental data as well as experimental metadata. Search capabilities are available that query on, for example, a time-range or a particular metadata key. The powerful and easy-to-use Python package pandas is leveraged for data inspection and analysis. When queried, databroker returns data as pandas data-frames. These data-frames are flexible and high-performance data structures for data analysis using Python [12].

C. *Hardware Abstraction Layer: ophyd*

The ophyd Python package is an interface between low-level hardware communication and the Bluesky experiment engine. Each instrument interface is ensured a *read* command and, when implemented by the hardware, a *set* command. The ophyd package facilitates the hierarchical construction of complex devices, by allowing the user to build up from signals, which comprise components, which comprise a device. A device is the highest level of abstraction within ophyd and provides additional attributes, such as a list of configuration signals. Signals specified as a configuration parameter may be read automatically at the start and end of each run and included in the metadata.

The design of ophyd is agnostic to the underlying hardware control system. Control interfaces supported by ophyd are currently limited to the Experimental Physics and Industrial Control System (EPICS) [13] through the Python package pyepics [14]. EPICS is a soft real-time hardware control-system utilized at large experimental facilities, such as x-ray synchrotrons and particle accelerators. These large experimental facilities often consist of many networked computers. By way of EPICS each networked computer may access or command the state of any connected instrument using the channel access protocol. The capabilities of EPICS in terms of performance and scale are clear, yet EPICS is known to be a challenge to set up and to have a considerable learning curve [15]. We must create an instrument control-layer to replace EPICS to achieve the goal of Bluesky experimental control in a small laboratory.

III. Hardware Control in a Lab without EPICS: instrbuilder

To replace EPICS in the NSLS-II suite we developed instrbuilder [16], a Python package that reads and controls instruments which support ASCII-like communication protocols, such as the Standard Commands for Programmable Instruments (SCPI) [17]. The instrbuilder Python package is independent of the specific controller-to-instrument interface; both RS232 and USB interfaces have been implemented using the pySerial [18] and PyVISA [19] Python packages respectively. This package functions as a stand-alone instrument controller but is predominately intended to be a plug-in for experiment control using the NSLS-II Bluesky suite.

A. Automated Generation of Instrument Control Drivers Using instrbuilder

The instrbuilder control driver is built automatically from a list of commands. The command list is entered using a comma-separated-value file (CSV) (named *command.csv*) with each column defining the attributes of each command. The attributes indicate: 1) the name, 2) string sent, 3) whether the command gets values from the instrument, 4) a conversion function, 5) whether the command sets values of the instrument, 6) the range of allowable values to set, 7) if the command is a configuration parameter, 8) help information, and 9) the subsystem. Basic commands request a value by sending *'command-string?'* and set a value by sending *'command-string value'*. However, some commands that set or query a value require additional inputs. To accommodate these scenarios, an optional input dictionary is used with dictionary keys mapped to formatting keys in the sent string. This approach simplifies the software: only one generic `get` and one generic `set` method is used without a loss of coverage. At times, more complex commands are needed than can be communicated through the CSV files; in this case, the instrbuilder SCPI class is subclassed and complex commands are written in Python.

Commands optionally support a lookup table that maps values sent to and received from the instrument to more easily interpretable values. The lookup table is also entered via a CSV file (named *lookup.csv*) with each row containing the human-readable name and the corresponding value actually sent to or returned from the instrument. Commands may be automatically tested for communication errors and for an incorrect read-back (a `set` and `get` sequence). A help method returns the command documentation, the limits, return type, sub-system, and required configuration keys (if any). Help is requested on a single command, a sub-system, or all of the commands of an instrument. An instrument is comprised of a dictionary of commands, with key names taken from the name of the command.

An integrated circuit (IC) class is also available in instrbuilder. For the IC class, a list of registers replaces the list of commands that was described for instruments. The IC class has methods to read, write, and manage register settings. The register map of the IC is generated as a Python dictionary. Each register is represented as a Register class that has attributes to indicate the name; address; whether the register is read and write, read-only, or write-only; and whether the register is a static configuration value. An integrated circuit instance is created from the register map and, in addition to the list of registers, has attributes that include the slave address and methods to read and write a register.

B. Integration of instrbuilder with ophyd

An instrbuilder instrument consists of a dictionary of commands, a write method, and a read method. This organization allows for automated generation of ophyd devices–an ophyd component is created for each entry in the dictionary of commands. The ophyd device is the top of the hierarchy and contains all the components needed to control and acquire data from an instrument using Bluesky.

IV. Bluesky Application: A Comparison of Lock-in Amplifier Performance

Now that we have examined the construction of the control software let us turn to look at an example application. In this application, we study the performance of two lock-in amplifiers by controlling all measurements and capturing all data using the Bluesky suite integrated with instrbuilder. These demonstrations illustrate different use-cases and capabilities of Bluesky and instrbuilder. All experimental control code is maintained at https://github.com/lucask07/instrbuilder/tree/master/instrbuilder/bluesky_demo/lockin_tests.

Point-of-care (POC) diagnostic instruments are desired that result in inexpensive and portable test systems with diagnostic power similar to that of standard clinical instrumentation [20]. Despite progress made by the microfluidic community in optical sensing techniques, the detection of the final optical system typically relies upon standard laboratory instrumentation that is neither portable nor low-cost. Demonstrations of optical detection systems often invoke synchronous detection to isolate the signal from environmental noise sources using a digital signal processing (DSP) lock-in amplifier, like the SR810 from Stanford Research Systems (SRS) [21]. The SRS810, with a weight of $10\,\mathrm{kg}$ and dimensions of $43\,\mathrm{cm}$ x $13\,\mathrm{cm}$ x $50\,\mathrm{cm}$ [22], is not appropriate for deployable POC instruments; smaller instruments for lock-in detection are needed.

Miniaturized commercial off-the-shelf integrated circuits that implement synchronous detection, such as the ADA2200 synchronous demodulator [23]–[25] and the AD8333 I/Q demodulator [26], have been demonstrated in laboratory settings. The ADA2200 is a promising key component of a lock-in amplifier for certain applications due to the following specifications: 1) low power consumption of $1.30\,\mathrm{mW}$; 2) miniature 16-pin package; 3) performance specified over the industrial temperature range; and 4) simple system integration. Other work has created custom DSP lock-in amplifiers using miniature micro-controllers [27], [28]. Here we present reproducible methods to evaluate lock-in amplifiers that support the efforts to develop lock-in amplifiers appropriate for portable instrumentation.

Next, we evaluate and compare the performance of the SRS SR810 and the ADA2200 using Bluesky. The motivations are:

1) To demonstrate the use and benefits of the Bluesky and instrbuilder software suite in a small laboratory.
2) To present a side-by-side benchmarking of the SR810 and the ADA2200.
3) To document a standardized test procedure for custom lock-in amplifier evaluation.

A. Lock-in Amplifier Description

A lock-in amplifier mixes the input, $V_{sig}(t) = A\cos(2\pi f_{sig}t + \phi)$, with the reference frequency, $V_{ref}(t) = A\cos(2\pi f_{ref}t)$, to create an output at frequencies that are the sum and the difference of the signal and reference frequencies. This output, referred to as the I (in-phase) component, is:

$$V_{diff}(t) = A\cos(2\pi(f_{sig} - f_{ref})t + \phi)$$
$$V_{sum}(t) = A\cos(2\pi(f_{sig} + f_{ref})t + \phi).$$

This mixer output is subsequently low-pass filtered to eliminate the frequency sum component and leave only the frequency difference component (V_{diff}). An ideal low-pass filter would output a DC value proportional to the input signal amplitude when the input is coherent with the reference ($f_{sig} = f_{ref}$) and otherwise output zero. The bandwidth of the low-pass filter determines the dependence of the output magnitude upon the deviation of the signal frequency from the reference frequency. The phase difference (ϕ) between the signal and reference contributes to the magnitude of the output signal. To eliminate the need for phase adjustment, lock-in amplifiers are designed with two mixers: the first, measures the in-phase I component as described above; the second, mixes the signal with a 90° phase shifted replica of the reference to produce a quadrature component, Q [29]. With a two phase system the magnitude and phase of the input signal are readily calculated as:

$$A = \sqrt{I^2 + Q^2}$$
$$\phi = \arctan(Q/I)$$

The SR810 internally computes the magnitude and phase of the input signal. The ADA2200 outputs either the I component or the Q component (selected by a register bit). This limitation of the ADA2200 means that the relative phase of the input signal and the reference signal must remain constant to unambiguously determine the input amplitude. In practice, this can be overcome by deriving the signal from the reference output of the ADA2200 so that the signal and reference are phase locked.

B. Experimental Setup

The instruments used for lock-in amplifier evaluation and controlled entirely by Bluesky/instrbuilder include a waveform generator (Keysight 33500B), an oscilloscope (Keysight MSOX-3012A), a digital multimeter (Keysight 34465A), a power supply (Rigol DP832), and an I2C/SPI controller (Total Phase Aardvark). Spreadsheets of SCPI commands for each of these instruments are provided in the instrbuilder repository. A basic experimental setup for SR810 evaluation is shown in Figure 1.

The ADA2200 is mounted on the ADA2200-EVALZ development board (Analog Devices) with a supply voltage of $V_{DD} = 3.3\,\mathrm{V}$ and configured through its serial peripheral interface (SPI). The analog output from the ADA2200 is digitized by a Keysight 34465A Multimeter with the ADA2200 reference clock (RCLK) triggering a sequence of eight readings (see Figure 2). The offset corrected mean of these eight samples gives either the in-phase component, I, or the quadrature component, Q, depending upon the PHASE90 bit. The Bluesky experimental engine saves the multimeter data to a file. Statistics that summarize the returned array (i.e. mean) are calculated and available to the Bluesky live callbacks, such as a LiveTable.

Typical usage of the ADA2200 includes an analog low-pass between the ADA2200 output and an ADC (and/or a digital low-pass filter within a microcontroller). These digital filters, in effect, extract the eight sample mean (cycle average) in an RCLK period. Here Bluesky allows more flexibility. We capture unfiltered data and attach digital filter prototypes (constructed with the Python function *scipy.signal.iirfilter*) to the data stream, which allows investigations of optimal filter parameters.

C. Frequency and Phase Resolution

The first measurement evaluates the frequency resolution of the SR810 versus the slope of the low-pass filter and demonstrates a Bluesky `grid_scan`. To do so, Bluesky sets the internal reference frequency (f_{ref}) of the lock-in amplifier to $1220.70\,\mathrm{Hz}$ and configures the waveform generator to input a zero offset, $2\,\mathrm{V}$ pk-pk sinusoid. The Bluesky plan `grid_scan` steps the function generator signal frequency (f_{sig}) through 60 linearly spaced points from $f_{ref} - 60.0\,\mathrm{Hz}$ to $f_{ref} + 60.0\,\mathrm{Hz}$ and steps the filter slope of the lock-in amplifier through 6,12,18, and $24\,\mathrm{dB/oct}$. A delay attribute for each test parameter is present in Bluesky; we use this delay parameter to ensure that the lock-in amplifier result settles to $> 9\tau$ (τ is the filter time-constant) after each step of the frequency input. At each step, after settling is complete, the input signal amplitude ($A = \sqrt{I^2 + Q^2}$) is read from the lock-in amplifier. As the measurement proceeds the Bluesky *LiveTable* callback (shown in Listing 1) updates in real-time.

```
In [1]: %run Bluesky_demo/Bluesky_demo.py
+-----------+------------+------------+----------+
|   seq_num |       time |    freq    |    amp   |
+-----------+------------+------------+----------+
|         1 | 16:26:00.5 | 4997.00000 | 0.000244 |
|         2 | 16:26:00.8 | 4998.14285 | 0.000244 |
|         3 | 16:26:01.0 | 4999.28571 | 0.000915 |
+-----------+------------+------------+----------+
generator scan ['286692ea'] (scan num: 1)
```

Listing 1: A Bluesky LiveTable as seen in the IPython console that shows the lock-in amplifier measured signal amplitude (*amp*) as the function generator frequency (*freq*) is stepped. Table formatting has been slightly modified to fit one column.

Figure 3 displays the SR810 measured signal amplitude versus signal frequency. The results match expectations and demonstrate the trade-off of bandwidth and frequency resolution. For example, with a filter configuration of slope = $6\,\mathrm{dB/oct}$ and $\tau = 30\,\mathrm{ms}$ ($f_c = 5.30\,\mathrm{Hz}$) a signal at a frequency three octaves from the cutoff frequency ($\pm 42.4\,\mathrm{Hz}$ from the reference frequency) is expected to be attenuated by the filter by $18\,\mathrm{dB}$; an attenuation of $18.18\,\mathrm{dB}$ is measured. Similarly, for a filter slope of $24\,\mathrm{dB/oct}$ we anticipate an attenuation of $48\,\mathrm{dB}$ two octaves beyond the cutoff frequency and measure an attenuation of $49.19\,\mathrm{dB}$. Figure 3, and all

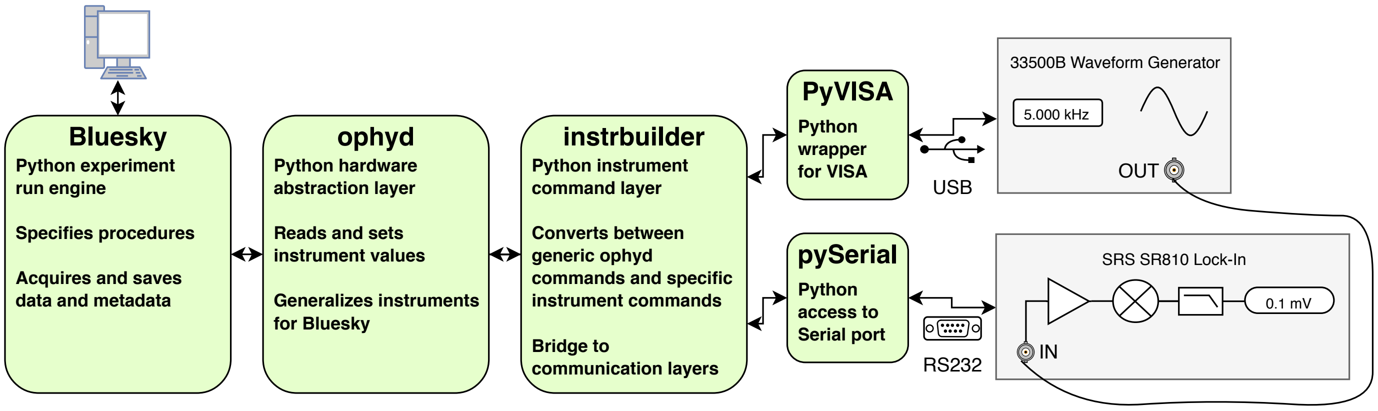
Bluesky
Python experiment run engine
Specifies procedures
Acquires and saves data and metadata
ophyd
Python hardware abstraction layer
Reads and sets instrument values
Generalizes instruments for Bluesky
instrbuilder
Python instrument command layer
Converts between generic ophyd commands and specific instrument commands
Bridge to communication layers
PyVISA
Python wrapper for VISA
pySerial
Python access to Serial port
USB
RS232
33500B Waveform Generator
5.000 kHz
OUT
SRS SR810 Lock-In
0.1 mV
IN

Fig. 1. A block diagram of the experimental setup to evaluate the frequency resolution of the SR810. From left to right, the figure shows the Python software packages (green), the computer to instrument communication methods (USB, RS232), and the instruments (gray). The computer represents the point at which the user interacts with the software suite.

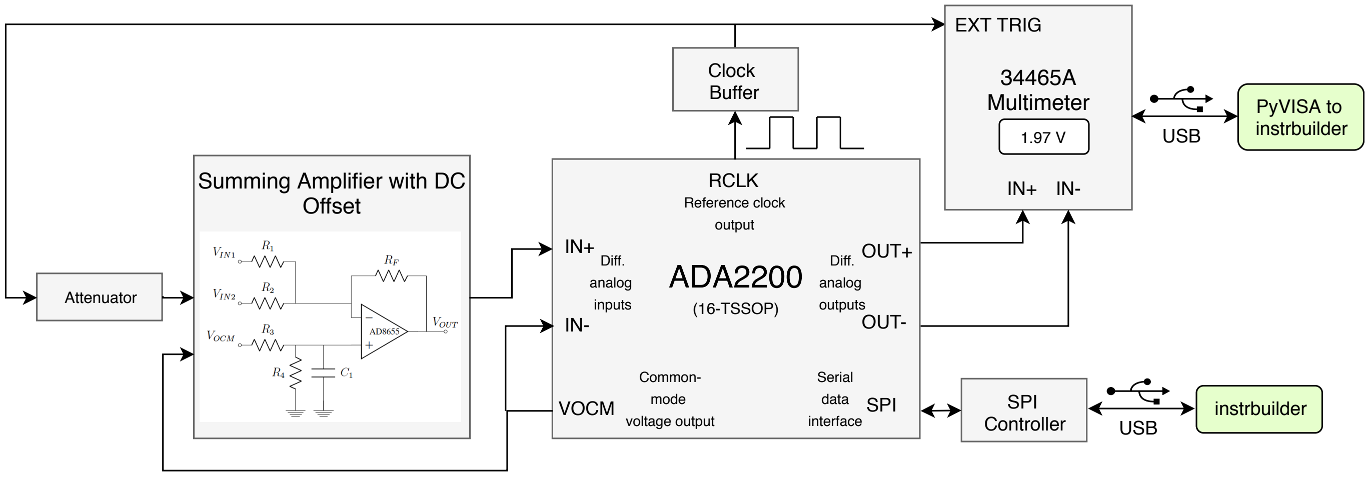

Fig. 2. A block diagram of the experimental setup used to evaluate the signal-to-noise ratio of ADA2200. The ADA2200 reference clock output (RCLK) is phase and frequency locked to the on-chip demodulation. The RCLK signal is used as a signal input to the ADA2200 and a trigger to the multimeter. The summing amplifier has two signal inputs (V_{IN2}, V_{IN1}) to, respectively, inject a coherent signal and an incoherent interferer for evaluation of interferer rejection. For SNR measurements V_{IN1} is left open. The multimeter is read and configured over USB by the Bluesky suite. The ADA2200 configuration registers are written and read over SPI through the Bluesky suite.

subsequent data figures in this paper, are tagged in the caption with the first six characters of the run UID. This tagging provides traceability from the manuscript figures to the data and analysis code. The analysis code to create each figure is available at GitHub (within the directory instrbuilder/bluesky_demo/lockin_analysis) [16] and the data and metadata are available at figshare [30].

The next experiment evaluates the phase resolution of the ADA2200. The ADA2200 outputs only I or only Q at a single time, making the evaluation of the frequency resolution a challenge. This is because when $f_{sig} \neq f_{ref}$ the input signal magnitude is ambiguous as the relative phase of the two signals continually drifts. Instead, the measurement we present quantifies the dependence of the output magnitude upon the phase between the input signal and the frequency reference when $f_{sig} = f_{ref}$.

Figure 4 shows the ADA2200 cycle average output voltage versus the relative phase between the input signal and the ADA2200 reference clock. Bluesky was used to set the waveform generator to input a sinusoid and to step the phase in 60 linearly spaced steps from 0° to 360° using a plan `scan`. The waveform generator sinusoid was configured to an amplitude of 0.707 V RMS and a frequency of 1220.680 518 Hz. This frequency was experimentally determined to minimize the phase drift between the input signal and the ADA2200 reference clock. At each phase step, the average of the digital multimeter burst reads was recorded as the input signal magnitude. The measurement of the phase difference between the input signal and the ADA2200 *RCLK* was captured by the oscilloscope and recorded by Bluesky. This oscilloscope measurement of phase difference is the parameter *Phase* displayed in Figure 4. From this experiment we extracted a phase of positive zero-crossing of 91.8° and a conversion gain of 1.064 V/(V RMS) as compared to datasheet specifications of 83° and 1.055 V/(V RMS).

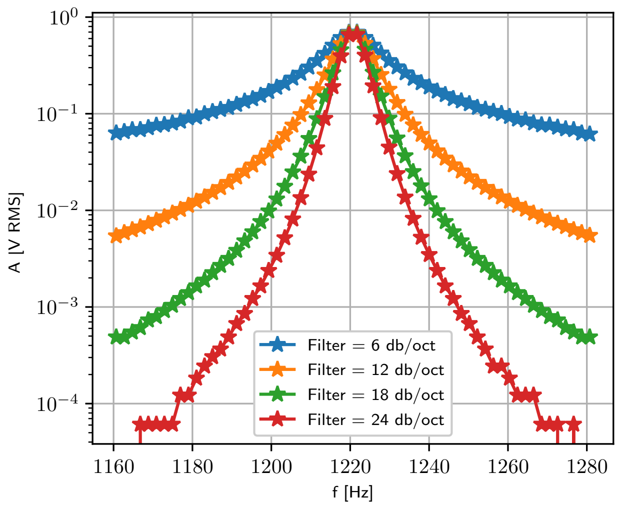

Fig. 3. Evaluation of the frequency resolution of the SR810 evaluated versus filter slope with a fixed time-constant of 30 ms. The Bluesky experimental engine produces a unique identifier (UID) for each experimental run. In this article we tag each figure with the first six characters of the experiment UID. UID='3de221'.

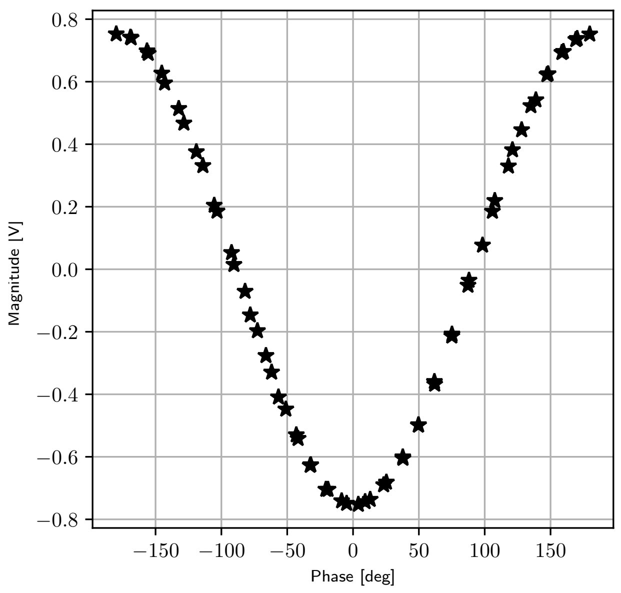

Fig. 4. Evaluation of the phase transfer function of the ADA2200 with a constant frequency 0.707 V RMS input sinusoid. Phase [deg] is the measured phase difference between the input signal and the reference clock. UID='b270dad3'.

D. Signal-to-Noise Ratio

The input signal-to-noise ratio (SNR) was evaluated for the SR810 and ADA2200 using fixed broadband attenuators (Mini-Circuits VAT and HAT) placed between the waveform generator output (SR810) or the RCLK output (ADA2200) and the lock-in amplifier input. A Bluesky `list_scan` stepped the attenuation through a list of values (0, 6, 10, 20, 30, 50, 60, 70, 80, 90, 100, and 110 dB). This measurement tested a single full-scale output sensitivity of 1 V RMS; of course, if the SR810 sensitivity is tuned based on the amplitude of the input signal, a wider dynamic range is achieved. Similarly, for the ADA2200 a preamplifier is not used, but applications with low voltage input signals would implement a preamplifier before the input of the chip for optimal performance.

SNR measurements require multiple samples (i.e. an array) at a single input amplitude and then a calculation of the mean and standard deviation of the captured array. To do so, Bluesky configures the SR810 to buffer a sequence of measurements to internal memory at a fixed sample rate. Once the buffer is full, Bluesky reads the values. An instrbuilder/ophyd component `SCPISignalFileSave` was designed for these array-like return values. This component saves an array to a user-selected file format and yields to Bluesky the filename (composed of a unique id and measurement index number); the filename, unlike the entire data array, is compatible with the final Bluesky data record and callbacks for live viewing. An ophyd signal `StatCalculator` comprises an ophyd device `BasicStatistics`, which can be attached to the component returning an array. Standard functions from the Python package NumPy such as, sum, mean, standard deviation, minimum, maximum, and length are calculated and available in real-time to Bluesky [31]. Using these components, for each attenuation level of the SNR measurement, Bluesky saved to disk a data array of multiple measurements and recorded the filename, mean, and standard deviation of the array. Investigative data post-processing can retrieve the original data arrays using the filenames logged by Bluesky during the experiment.

This SNR evaluation procedure requires an operator to manually switch attenuators between each measurement. To track the attenuation in the experimental record, a Bluesky component with `get` and `set` methods not attached to an underlying instrument control system was created (`ManualDevice` within *ee_instruments.py*). This demonstrates the simplicity of creating a custom Bluesky instrument automatically tracked in the experimental data records (four lines of Python code).

Similarly, the ADA2200 SNR was evaluated using a buffered version of the reference clock (RCLK) chip output as the input signal. The ADA2200 requires an input common-mode of 1.65 V; this common-mode voltage is provided as a chip output at the VOCM pin. Since the attenuators suppress the DC component a summing amplifier with DC offset adjust, shown in Figure 7, was used after the attenuation to add the required common-mode level. The differential output of the ADA2200 was digitized by the digital multimeter with the RCLK signal serving as an external trigger. The multimeter sampled the output level eight times per trigger with captures timed to measure each of the ADA2200 eight output sequences. The input signal was measured to be 2.71 V pk-pk with a 1.6205 V common-mode at a frequency of 390.58 Hz. Figure 6 shows SNR results for a filter time-constant of 10 ms and a filter-slope of 24 dB/oct. The noise at low input levels was measured as 80 μV RMS. The SNR (Out/Noise) is $>$10.8 bits with an input signal of 0.271 V pk-pk, falls to 8.0 bits with an input signal of 0.0271 V pk-pk and is reduced to 3.1 bits at an input signal of 0.856 mV pk-pk.

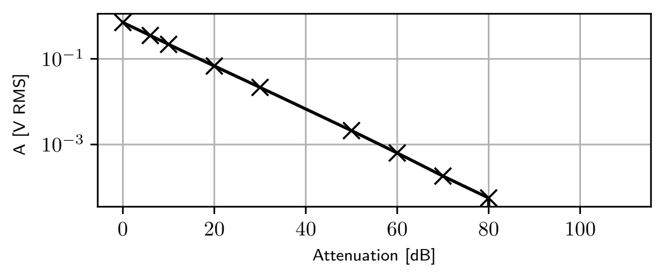
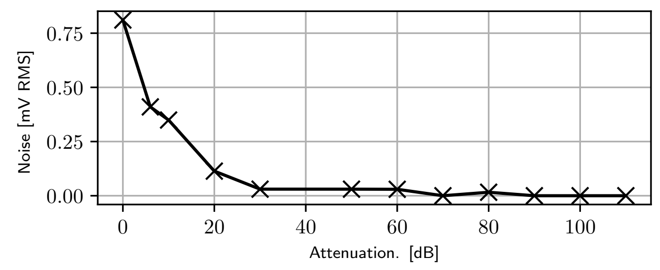

Fig. 5. Evaluation of the SNR of the SR810. At an attenuation of 90 dB or greater the measured noise is 0. A is the value returned by the SR810 and is the magnitude of the in-phase and quadrature outputs: $A = \sqrt{I^2 + Q^2}$. UID='6c22a3cf'.

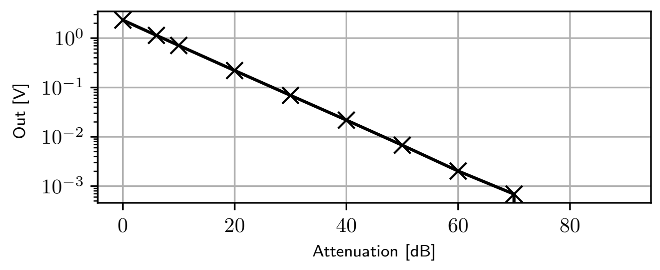
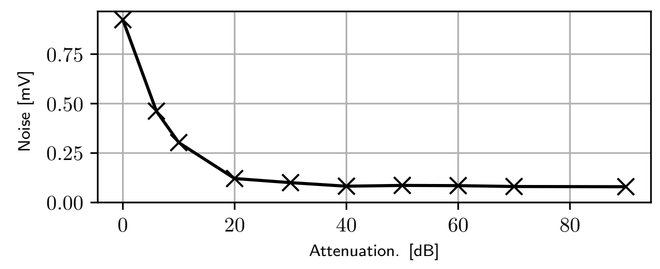

Fig. 6. Evaluation of the SNR of the ADA2200 using a low-pass filter slope of 24 dB/oct and time-constant of 10 ms. At large attenuation the noise plateaus to 80 μV. UID='f6e88efb'.

E. Dynamic Reserve

The dynamic reserve of a lock-in amplifier quantifies the rejection of interfering input signals (f_{int}) at frequencies away from the reference frequency. How much greater in amplitude can an interferer at $f \neq f_{ref}$ be than the input signal at f_{ref} before a 5% error is caused? This specification of lock-in amplifiers is termed 'reserve' since an input range is held 'in reserve' to accommodate noise and interference [29]. To measure the dynamic reserve, the summing amplifier in Figure 7 combines an incoherent interferer, V_{int}, and an attenuated coherent signal, V_{sig}. This measurement demonstrates a custom Bluesky plan that is adaptive and searches for the maximum amplitude of incoherent interferer that produces a change of less than 5% in the result measured in the absence of an interfering component (see the function *adaptive_sweep* within dynamic_reserve_sr810.py at GitHub [16]). The search is repeated over a range of interferer frequencies. Figure 8 shows the dynamic reserve $(20 \log 10(V_{int}/V_{sig}))$ of the SR810 versus frequency when configured in 'normal' reserve mode at a sensitivity of 20 μV, a filter slope of 24 dB/oct and a time-constant of 100 ms. The measured peak value of 86 dB, outperforms the value of 74 dB specified in the SR810 manual for the given sensitivity and reserve setting. Of particular interest is the reserve at the harmonics of the input signal. With an interferer frequency of $fs/2$ the reserve decreases to 54 dB.

A similar metric, termed interferer rejection, was measured for the ADA2200 using the process described above with one exception. The ADA2200 does not have options for a non-unity gain from the input to output. Because of this, an input signal that produces a full-scale output is at the limits of the input voltage range (set by the power supply) and any superimposed interferer signal would saturate the input amplifier. In order to produce a meaningful measurement of the ability to reject out-of-band interference, a low amplitude input signal of 6.1 mV pk-pk was input and the Bluesky algorithm searched for the largest interferer amplitude that produced an output change of less than 5%. Since the input signal did not produce an output at 90% of full-scale this measurement is not precisely quantifying dynamic reserve, however, this method does accurately communicate the level of interferer that can be rejected while still correctly measuring a coherent input. Figure 9 shows the interferer rejection, the ratio of the maximum interferer signal over the coherent signal, of the ADA2200 for a 10 ms time-constant low-pass filter with a 6 dB/oct slope (the filter was implemented in Python SciPy). The rejection is nearly 54 dB at most frequencies away from the odd harmonics ($\times 3$, $\times 1/3$, ...) of the input frequency (input and interferer are both square waves).

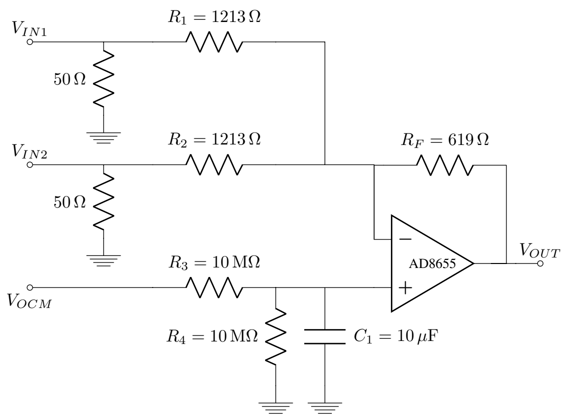

Fig. 7. Summing amplifier (with optional DC offset) for dynamic reserve evaluation and signal-to-noise ratio studies of the lock-in amplifiers. When testing the ADA2200 the common-mode output from the ADA2200 was connected to the V_{OCM} input. The 50 Ω terminating resistors shown are either an actual resistor on the board or the termination of inline attenuators. The AD8655 is powered with split supplies, ±2.5 V, or a single supply, 5.0 V, when testing the SR810 and the ADA2200, respectively.

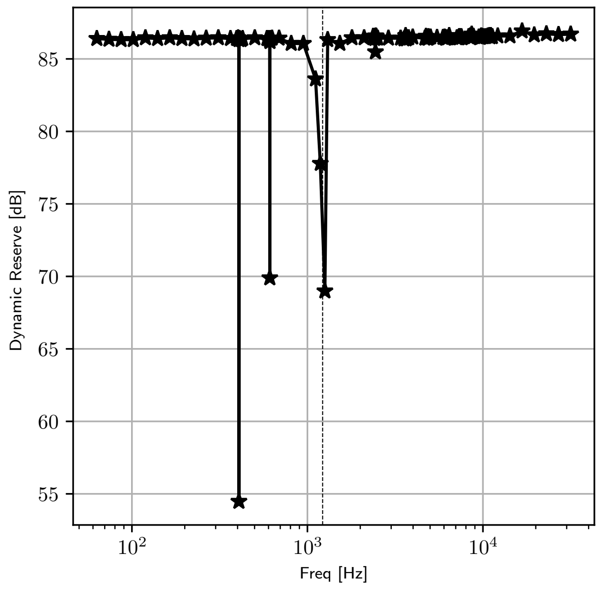

Fig. 8. Evaluation of the dynamic reserve of the SR810 in 'Normal' reserve mode, with a time-constant of $20\,\mu$s and a $24\,\mathrm{dB/oct}$ filter slope. The signal frequency of $1.22\,\mathrm{kHz}$ is indicated with a dashed line. UID='ffcd20b8'.

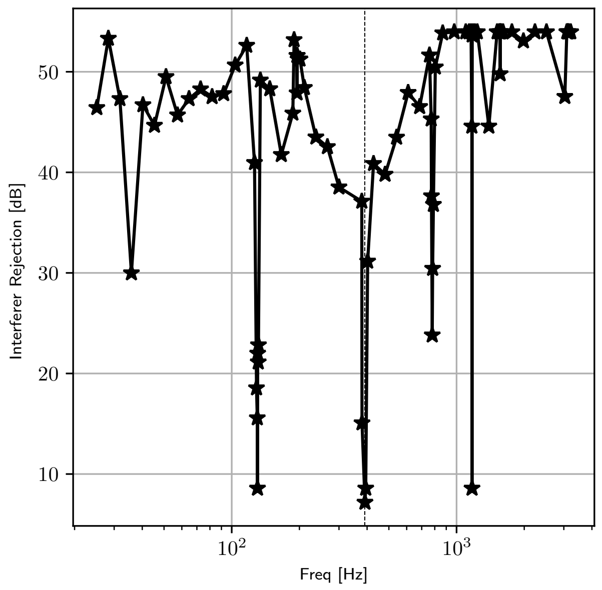

Fig. 9. Evaluation of the dynamic reserve of the ADA2200. The signal frequency of $390.5\,\mathrm{Hz}$ is indicated with a dashed line. UID='470bc279'.

V. Data, Metadata, Analysis, and Configuration Equivalence Checking

The Python package databroker is the interface to the analysis of data captured by Bluesky. The document organization of Bluesky data is used by databroker to: 1) access experimental metadata, 2) provide additional methods attached to the metadata which summarize available information, 3) search on metadata by keyword, and 4) lazily load data. To analyze lock-in data we track the experimental run using the UID, review the experiment configuration programmatically by inspection of the metadata, and utilize the analysis capabilities provided by Python pandas and NumPy to process the data once loaded by databroker.

The experimental data described in this paper are archived at figshare [30]. The archive includes: 1) event data 2) run start, run stop, and event descriptors metadata and 3) a databroker configuration file. The instrbuilder GitHub repository provides the analysis code for each of the data figures in this paper and a script that runs the entirety of the analysis (see *run_all_analysis.py* in the directory instrbuilder/bluesky_demo/lockin_analysis/) [16]). As part of the data processing, examples of experimental run metadata, such as the duration of the experiment and time of the run, are printed to the analysis console.

A. Instrument Configuration Equivalence Checking

The instrbuilder `Command` class includes a property which indicates whether a readable value should be considered a configuration parameter. These configuration values are not expected to change over the course of the run, however are a critical component of experiment reproducibility. An API `read_configuration()` is provided by Bluesky that reads all components of a device which are specified as configuration parameters. As a part post-processing, the equivalence of every configuration parameter for experiments that are nominally expected to be the same can be checked. This automated verification of equivalent configuration is an essential contribution of the Bluesky/instrbuilder suite to experiment reproducibility.

As an example of metadata equivalence checking we present processing code that inspects the SR810 signal-to-noise measurements. This example first searches for experimental runs with the purpose 'snr_SR810'. Next, the code compares configuration values from the metadata of two distinct runs found in the search. In the process, the programmatic comparison reveals 43 configuration keys of the lock-in amplifier and detects two differences, which are printed. Listing 2 shows the IPython console output from this metadata processing example (*metadata_examples.py* in the directory instrbuilder/bluesky_utils/ [16]). This automated recording of all configuration data and organization into Python dictionaries is facilitated by the hierarchical approach to instrument construction as a collection of signals.

B. Software Setup and Getting Started

The instrbuilder package is hosted at PyPI, references the NSLS-II Bluesky suite dependencies, and can be installed using *pip*. Complete setup instructions for use in a small laboratory are available at the README on GitHub [16]. Most

```
In [1]: %run metadata_example.py
Found 24 runs with the purpose of snr_SR810
Comparing:
  UID = 5b9c4bf7-55c6-45ce-99c3-e717be0d1ef3 to
  UID = a71988ee-2da4-486f-9fd2-f97551306695d
Located 43 configuration keys to compare:
 Found configuration difference:
  Run 1: lockin_sample_rate = 32.0
  Run 2: lockin_sample_rate = 1.0
 Found configuration difference:
  Run 1: lockin_tau = 0.01
  Run 2: lockin_tau = 0.1
```

Listing 2: An example of configuration equivalence checking using the Bluesky metadata within an IPython console. The analysis code first searches for experimental runs tagged with the purpose of 'snr_SR810' and then compares the configuration of two of these runs. These two runs have 43 configuration keys in common which are compared – two differences are found.

laboratories do not have a lock-in amplifier to exercise the examples in this paper. As such, we created a Bluesky/instrbuilder demonstration using a function generator and an oscilloscope [16]. In addition, Bluesky tutorials that do not require hardware but instead exercise simulated hardware from ophyd are available (see http://nsls-ii.github.io/bluesky/tutorial.html).

VI. Discussion

The ADA2200 has potential as a key component of a lock-in amplifier in power and volume constrained applications. The analog sampling of the ADA2200 reduces the sampling rate needed of a downstream ADC and limits the complexity of subsequent digital signal processing algorithms (i.e. an IIR lowpass filter); a small and low-power microcontroller could be used in conjunction with the ADA2200 to create a lock-in amplifier. In terms of noise floor, $80\,\mu\mathrm{V}$ (at $24\,\mathrm{dB/oct}$ filter), and interference rejection, $54\,\mathrm{dB}$, the ADA2200 can extract millivolt level coherent signals superimposed among a full-scale ($3.3\,\mathrm{V}$) incoherent background. With an input signal of $6.1\,\mathrm{mV}$ pk-pk out-of-band interferers (at frequencies away from harmonics of the input signal) reached $3.02\,\mathrm{V}$ pk-pk without causing a greater than 5% deviation to the output signal. Larger input amplitudes were not tested due to the specifications of the minimum and maximum allowable input signal voltages of $0.3\,\mathrm{V}$ and $V_{DD} - 0.3$ V, respectively. Optimization of noise, interferer rejection, and transient response is possible for a specific application by modifications of the low-pass filter parameters.

Since the ADA2200 outputs only one of I or Q at a time some flexibility in experimental design is sacrificed. In most cases, this can be handled by designing an instrument that buffers the RCLK output and uses this as the sample stimulus. On the other hand, the SR810 can output measured phase, measured magnitude, or I; and the SR810 can lock to a replica of the sample stimulus signal. These capabilities and the wide range of configurable sensitivities ($2\,\mathrm{nV}$ to $1\,\mathrm{V}$) increases experimental flexibility and convenience of the SR810. However, the ADA2200, with a power consumption of $\sim 1\,\mathrm{mW}$ at a supply voltage of $3.3\,\mathrm{V}$ and an analog performance that spans $\sim 54\,\mathrm{dB}$, is an intriguing building block for lock-in detection in portable instrumentation.

VII. Conclusion

Experiment reproducibility starts at data collection. The methods and computer code used for data collection should be clear and general so that others can reproduce a research result starting with the experiment. Or, in the classifications of reproducible computational research described by Stodden [32], the experiment should be *replicable*. The data itself should be archived so that it is portable, self-contained, and contains complete details of the configuration of the experiment, or per Stodden, *auditable* [32]. The software package described above accomplishes these goals. The hierarchical approach to instrument construction of the Bluesky suite facilitates recording of self-contained data records that describe instrument configurations; the command generation approach of instrbuilder abstracts instrument specific commands to a generic language so that experimental code can be replicated by other laboratories.

This paper introduces the Bluesky and instrbuilder combination as a capable software suite to control experiments. We acknowledge that many capabilities of Bluesky were left out. Exciting future developments that may be contributed by a user community include integration with laboratory cameras by leveraging the Bluesky/ophyd interface to x-ray area detectors, asynchronous acquisition using the Bluesky/ophyd ”flyers” interface, and live image viewing that utilizes real-time video encoding.

Acknowledgment

L. Koerner acknowledges laboratory start-up funding from the University of St. Thomas School of Engineering. This research used resources of the National Synchrotron Light Source II, a U.S. Department of Energy (DOE) Office of Science User Facility operated for the DOE Office of Science by Brookhaven National Laboratory under Contract No. DE-SC0012704.

References

[1] Y. Xie, “knitr: a comprehensive tool for reproducible research in R,” *Implement Reprod Res*, vol. 1, p. 20, 2014.
[2] B. Baumer, M. Cetinkaya-Rundel, A. Bray, L. Loi, and N. J. Horton, “R markdown: Integrating a reproducible analysis tool into introductory statistics,” *arXiv preprint arXiv:1402.1894*, 2014.
[3] M. Pastell, “Pweave,” 2016. [Online]. Available: http://mpastell.com/pweave/index.html
[4] M. Ragan-Kelley, F. Perez, B. Granger, T. Kluyver, P. Ivanov, J. Frederic, and M. Bussonnier, “The Jupyter/IPython architecture: a unified view of computational research, from interactive exploration to communication and publication.” *AGU Fall Meeting Abstracts*, pp. H44D–07, Dec. 2014.
[5] J. M. Ellis, “Application specific automated test equipment system for testing integrated circuit devices in a native environment,” U.S. Patent 6,324,485, Nov. 27, 2001.
[6] E. C. Hayden, “The automated lab,” *Nature*, vol. 516, no. 7529, pp. 131–132, Dec. 2014.
[7] A. Arkilic, D. B. Allan, T. Caswell, L. Li, K. Lauer, and S. Abeykoon, “Towards integrated facility-wide data acquisition and analysis at NSLS-II,” *Synchrotron Radiation News*, vol. 30, no. 2, pp. 44–45, Mar. 2017.
[8] NSLS, “NSLS-II software documentation.” [Online]. Available: http://nsls-ii.github.io/
[9] J. L. Johnson, H. tom Wörden, and K. van Wijk, “PLACE,” *Journal of Laboratory Automation*, vol. 20, no. 1, pp. 10–16, Feb 2015.
[10] K. Pernstich, “Instrument control (ic) - an open-source software to automate test equipment,” *Journal of Research of the National Institute of Standards and Technology*, vol. 117, 2012.

[11] A. Arkilic, D. Allan, T. Caswell, L. Dalesio, and W. Lewis, "Metadatastore: A Primary Data Store for NSLS-2 Beamlines," in *Proceedings, 15th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS 2015)*, 2015, p. WEPGF043. [Online]. Available: http://inspirehep.net/record/1481665/files/wepgf043.pdf

[12] W. McKinney *et al.*, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, vol. 445, 2010, pp. 51–56.

[13] A. Kozubal, L. Dalesio, J. Hill, and D. Kerstiens, "Experimental physics and industrial control system," in *Proceedings, International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS89)*.

[14] M. Newville, "PyEpics: Python epics channel access," CARS, University of Chicago, Tech. Rep., 2016.

[15] L. Dalesio, R. Chesnut, H. Shoaee, and J. Hill, "Epics directions to accomodate large projects and incorporate new technology," in *International Conference on Accelerator and Large Experimental Physics Control Systems*, 1999.

[16] L. Koerner, "instrbuilder," *GitHub repository*, 2019. [Online]. Available: https://github.com/lucask07/instrbuilder

[17] S. Consortium *et al.*, *Standard Commands for Programmable Instruments (SCPI), Volume 1: Syntax and Style*, 1999.

[18] C. Liechti, "pySerial documentation," 2017. [Online]. Available: http://pyserial.readthedocs.io/en/latest/

[19] "PyVISA documentation," 2018. [Online]. Available: https://pyvisa.readthedocs.io/en/stable/

[20] F. B. Myers and L. P. Lee, "Innovations in optical microfluidic technologies for point-of-care diagnostics," *Lab on a Chip*, vol. 8, no. 12, p. 2015, 2008.

[21] A. E. Herr, D. J. Throckmorton, A. A. Davenport, and A. K. Singh, "On-chip native gel electrophoresis-based immunoassays for tetanus antibody and toxin," *Analytical Chemistry*, vol. 77, no. 2, pp. 585–590, 2005, pMID: 15649057.

[22] Stanford Research Systems, "SR810 & SR830 — 100 kHz lock-in amplifier datasheet." [Online]. Available: http://www.thinksrs.com/downloads/pdfs/catalog/SR810830c.pdf

[23] L. Orozco, "Synchronous detectors facilitate precision, low-level measurements," Analog Devices, Tech. Rep., 2014. [Online]. Available: http://www.analog.com/media/en/analog-dialogue/volume-48/number-4/articles/synchronous-detectors-facilitate-precision.pdf

[24] O. Sushynskyi, M. Vistak, and V. Dmytrah, "The sensitive element of primary transducer of protein optical sensor," in *2016 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET)*. IEEE, 2016, pp. 418–421.

[25] E. Ortega-Robles, A. Cruz-Orea, and D. Elías-Viñas, "Simple and portable low frequency lock-in amplifier designed for photoacoustic measurements and its application to thermal effusivity determination in liquids," *Review of Scientific Instruments*, vol. 89, no. 3, p. 034904, 2018.

[26] B. G. Saar, C. W. Freudiger, J. Reichman, C. M. Stanley, G. R. Holtom, and X. S. Xie, "Video-rate molecular imaging in vivo with stimulated raman scattering," *Science*, vol. 330, no. 6009, pp. 1368–1370, Dec. 2010.

[27] L. E. Bengtsson, "A microcontroller-based lock-in amplifier for sub-milliohm resistance measurements," *Review of Scientific Instruments*, vol. 83, no. 7, p. 075103, 2012.

[28] G. Li, M. Zhou, F. He, and L. Lin, "A novel algorithm combining oversampling and digital lock-in amplifier of high speed and precision," *Review of Scientific Instruments*, vol. 82, no. 9, p. 095106, 2011.

[29] M. L. Meade, "Advances in lock-in amplifiers," *Journal of Physics E: Scientific Instruments*, vol. 15, no. 4, pp. 395–403, Apr 1982.

[30] L. Koerner, "Lock-in amplifier data collected using Python instrument control suite of Bluesky and instrbuilder," Feb 2019. [Online]. Available: 10.6084/m9.figshare.7768352.v1

[31] T. Oliphant, "NumPy: A guide to NumPy," USA: Trelgol Publishing, 2006–. [Online]. Available: http://www.numpy.org/

[32] V. Stodden, J. Borwein, and D. H. Bailey, "Setting the default to reproducible," *Computational Science Research. SIAM News*, vol. 46, no. 5, pp. 4–6, 2013.

Lucas J. Koerner received B.A. degrees in Integrated Science (Hons), Physics, and Mathematics from Northwestern University in Chicago, IL, USA. He then received the degree of Ph.D. in Physics from Cornell University in Ithaca, NY, USA. Since 2018 he is an Assistant Professor of Electrical Engineering at the University of St. Thomas, St. Paul, MN, USA. His research interests include electrical instrumentation development, software for reproducible research, and image sensors.

Thomas A. Caswell received B.S. degrees in Physics and Mathematics from Cornell University, Ithaca, NY, USA, and the Ph.D. degree in Physics from University of Chicago, IL, USA. He is currently scientific staff at Brookhaven National Lab, Upton, NY, USA and is a co-lead developer of matplotlib.

Daniel B. Allan received B.S. degrees in Physics and Mathematics and B.A. in Music from the University of Rochester, NY, USA, and a Ph.D. degree in Physics from Johns Hopkins University, MD, USA. He is currently scientific staff at Brookhaven National Lab, Upton, NY, USA.

Stuart I. Campbell received his B.Sc. (Hons) and Ph.D. degrees in Physics from the University of Salford, UK. He has previously held scientific staff positions at Rutherford Appleton Laboratory and Diamond Light Source in Oxfordshire, UK and Oak Ridge National Laboratory, TN, USA. He is currently scientific staff and group leader for data acquisition, management and analysis within the NSLS-II at Brookhaven National Laboratory, Upton, NY, USA.