Heliophysics Data Environment Enhancements
Abstracts of Selected Proposals
(NNH20ZDA001N-HDEE)

Below are the abstracts of proposals selected for funding for the Heliophysics Data Environment Enhancements program. Principal Investigator (PI) name, institution, and proposal title are also included. 16 proposals were received in response to this opportunity. On October 29, 2020, 8 proposals were selected for funding.

Bryan Harter/University Of Colorado, Boulder
Integrating the python XArray library and the CDF file format

Background
XArray is an open source python library that provides multi-dimensional data storage with their custom DataArray and DataSet data structures. In addition to storing the data values themselves, XArray objects also store coordinate information and allow the user to interface with the data by referencing coordinate names. These data structures can also handle arbitrary metadata alongside the data in the form of the "attribute" dictionary which is attached to each object by default.
Because of the inherent ability to store multi-dimensional data alongside metadata, XArray has gained adoption in the python in heliophysics community. Of particular note is that XArrays are the primary data storage object in the PySPEDAS and PyTplot libraries. Other heliophysics libraries, such as pysat, use XArray objects in portions of their code, and have the ability to set the output of some functions as XArrays. However, even without officially supporting XArray input, the data inside of the XArray objects can still be used in the majority of typical data processing routines, since the data inside of XArray objects can be exported to Pandas DataFrame objects or simply NumPy matrices.
XArray had its origins in the field of geophysics and has typically been coupled with netCDF files, which are prevalent in that field. Because of this coupling, the DataArray and DataSet data structures have the inherent ability to receive netCDF files as input, as well as output their data structures into netCDF files. While netCDF is used in the field of heliophysics too, CDF files are also quite prevalent throughout the field.
"Cdflib" is an open source pure python CDF file reader/writer that interfaces with the CDF files directly rather than going through the C libraries. As such, there is no need for the user to install extra libraries to read in CDF data. It contains 3 primary modules 1) a CDF reading class, 2) a CDF writing class and 3) A time conversion class for converting CDF time types (EPOCH/EPOCH16/T2000) into python data structures (datetime, Astropy epochs, etc.). Cdflib is currently being used as the CDF file reader in the PySPEDAS and Heliopy libraries.

Proposal
Because of the widespread usage of both CDF files and the XArray library in the field of heliophysics, we propose that modules be added to the cdflib library that enables the conversion of ISTP compliant CDF files to XArray DataSets objects. The module would also allow the conversion of DataSet objects back into ISTP compliant CDF files.
This would enable researchers to quickly load the data from CDF files into a python data structure that integrates well with the python heliophysics tools. Processing can be done on the values retrieved from the CDF file using the suite of heliophysics python tools, and exported back again to a CDF format. This would enable data processing workflows found in the IDL SPEDAS library.

Methods
Unit tests will be developed for all major functions, striving to achieve complete code coverage. These unit tests would run when new code changes are pushed to the GitHub repository using the "GitHub Actions" feature. Additionally, the new module can be tested with data from a variety of missions (MMS, MAVEN, THEMIS, etc.) to ensure that the module works as expected given a variety of real heliophysics data.
Documentation of the module will be added to cdflib.readthedocs.io.
Because the XArray library already contains functions for converting between DataSets and netCDF files, these functions can be looked to for reference when designing the CDF to XArray module. The interface and implementation of XArray<->netCDF and XArray<->CDF modules should be kept similar in order to maintain consistency for current users of the XArray library.

Jonathan Niehof/University of New Hampshire, Durham
dbprocessing: Space Science Data Processing Controller in Python

We propose to provide the Python in Heliophysics Community (PyHC) with a tool to automate routine data processing needs arising in every Heliophysics mission. We will adapt an existing mature codebase which manages complex dependencies from inputs to high-level products and tracks provenance. Heliophysics science teams and operations centers would save substantial development time with this developer-friendly architecture.

Processing Heliophysics data to scientifically useful products requires not only code to produce a given output file from a given input file (e.g. processing a telemetry file into a counts-based file), but tracking relationships between multiple data types and updating data products for new input data. This process becomes intractable to perform by hand.

To address this problem on the Radiation Belt Storm Probes-Energetic particle, Composition, and Thermal plasma suite (RBSP-ECT), the proposing team developed a system called dbprocessing, a database-driven processing controller which uses the arrival of new data to automatically execute relevant processing codes, providing updated files for all possible data products. dbprocessing controlled the processing of all data from the RBSP-ECT suite, used in hundreds of peer-reviewed science publications

studying the radiation belts. The data similarly processed for Parker Solar Probe/Integrated Science Investigation of the Sun (PSP/ISOIS) have already resulted in over a dozen peer-reviewed publications investigating energetic particle dynamics in the inner heliosphere. It similarly supports data processing for >20 national security satellites at Los Alamos National Laboratory.

Several complementary tasks will transform dbprocessing into a tool suitable for community use and enhancement. These include transition to a common public repository, modernization of the code base, and development of unified documentation for users and developers.

A dbprocessing-based system can go beyond processing telemetry from a single mission. The use of higher-level input is proven by the ISOIS and RBSP-ECT use of level 2 magnetic field inputs from other instruments. dbprocessing could also support a study combining products from several missions, ground sources, and even model output. All publicly-available Heliophysics data are potentially exploitable: if data files can be placed in a location where dbprocessing has access, they can serve as inputs to such a system.

The proposed work will transform dbprocessing from an internal project, suitable only for its developers, to one ready for use by the broader community, fully engaged with the PyHC and its standards.

Joseph Plowman/Association Of Universities For Research In Astronomy, Inc.
EMToolkit: Computing and Visualizing Differential Emissions Measures with a Standardized Framework

Science goals: Provide a standardized set of tools for computing, viewing, and analyzing the thermal structure of coronal plasmas

Emission from the solar corona is characterized by a `Differential Emission Measure' (DEM), which is a distribution of squared density along the line of sight as a function of temperature. The intensity of most coronal spectral lines are completely determined by these functions, so they encompasses everything these line intensities can tell us from a single point of view. This project would provide a standardized set of tools for computing, viewing, and analyzing these functions.

Methodology:

We have developed a new code for computing these DEMs, which is under publication. As part of this project, we would port our new DEM code to Python, as well as the another widely accepted DEM code, that of Cheung 2015. We would provide these codes with a standard interface so that other DEM codes could be seamlessly integrated with the rest of the framework. This would also include capability for estimating uncertainties in the DEMs and noise reduction in the inputs (e.g., by rebinning).

We would also extend the codes so they can perform inversions from SDO, XRT, and EIS simultaneously. This will be implemented in such a way that any other new data source that specifies a observing time, viewport, and temperature (or spectral?) response function (e.g., from Solar Orbiter or SUVI) can be seamlessly integrated.

The results of computing DEMs from a data sequence is a 4-D (x, y, t, T) data cube. We provide various ways of visualizing this data using the 2 spatial dimensions, 1 time dimension, and 3 color channels available on a computer screen (the `viewer' space). Examples include:

1. Providing basic temperature map images, both static time snapshots and movies made from image sequences.
2. Providing color images made from individual temperature components of the DEM, or from `synthetic' temperature response functions integrated against the DEM.
3. As part of the above, we will also provide capability for producing colorbars which related perceived hue, intensity, and saturation to temperature, emission measure, and DEM width. These can also be used to visualize multichannel image data (e.g., from SDO/AIA) directly.
4. Slicing the cube, such as a slice in the t, T plane (i.e., showing the time evolution of the DEM at a particular point in space);
5. The DEM along a linear feature, such as a coronal loop. A background subtraction capability and framework would be supplied to enhance visualization of such features.

These are all examples of a class of mapping and projection operators from the 4-D (x,y,t,T) emission measure space to the 3-D (x,y,t) viewer space. The backend framework would include a generalized implementation of these operators with the capability to define any desired instance procedurally as well as via an interactive use interface, and to translate between the two.

Ashton Reimer/SRI International

Enhancements, Updates, and PyHC Compliance for the Reproducible Software Environment Platform

The reproducibility of an experimental result is a fundamental requirement of the scientific method. In the digital age, there has been an increasing focus on the challenge of demonstrating the reproducibility of analyses performed with software on computers [e.g., Teytelman (2018)].
The Reproducible Software Environment (Resen) was developed as a means of facilitating reproducibility of analyses for the geospace and heliophysics research communities. Resen is a novel, open-source Python package that provides a JupyterLab interface to containerized (e.g., Docker) computational environments called buckets [Bhatt et al. (2020)]. A bucket contains the user s software and the data required to perform research. Buckets can be trivially shared with other researchers or published to

online citable repositories. The containerization ensures computational reproducibility. Additionally, Resen provides ready-to-go images (referred to as cores) that contain pre-installed, commonly used, community Python tools.

We developed a working version of Resen under two National Science Foundation (NSF) awards: EarthCube and Cyber Infrastructure for Sustained Innovation. We presented a pre-release of Resen to the CEDAR community at the 2019 CEDAR Workshop and a full release of Resen at the 2019 AGU Fall Meeting. In April 2020, we delivered three webinars that generated significant CEDAR and NASA community interest in Resen. Heliophysics scientists, including some working on NASA-funded research projects, are eager to use online and downloadable versions of Resen, especially to facilitate access to and use of mission data.
To meet the broader requirements of the Python in Heliophysics Community (PyHC), we propose some value-added software development for Resen that includes code refactoring, unit testing, and compliance with the PyHC Standards [see Annex (2018)]. We propose that a team of three developers complete the following tasks to improve Resen, bring it into compliance with the PyHC Standards, and ensure better compatibility with the wide range of applications used in Heliophysics research:
1) Refactor the codebase to remove duplicate code,
2) Ensure codebase compatibility with Python 3.8,
3) Implement unit testing via Travis-CI (or similar),
4) Improve documentation in user guides/tutorials, inline code, and docstrings,
5) Post contribution guidelines,
6) Adopt a code of conduct, and
7) Prepare a lessons-learned document for the PyHC.

Resen was developed with several software development best practices in mind (e.g., version control, minimization of dependencies, use of Python Enhancement Proposal (PEP)-8 coding style, adequate documentation). Thus, we anticipate Tasks 3 and 4 will require approximately half the total development time, and all other tasks will be completed in the remaining time. Improvements to documentation will include a guide on how advanced users can create their own Resen core , which will enable Resen to be used by researchers, such as modelers, who do not exclusively use Python. Additionally, we will prepare a lessons-learned document at the end of the project that will detail the development process used to bring pre-existing software into compliance with the PyHC Standard. This document will provide useful information for the PyHC and may be used as a reference for making other pre-existing software packages PyHC-compliant.

References:
Annex (2018), doi:10.5281/zenodo.2529131
Bhatt (2020), doi:10.1051/swsc/2020011
Docker (2020), see https://docs.docker.com/install/
Teytelman (2018), doi:10.1038/d41586-018-06008-w

Daniel Ryan/American University
Solar X-ray Spectroscopy Analysis Tools for the Heliophysics Python Ecosystem

We propose to develop new Python-based analysis tools for solar X-ray spectroscopy. X-rays are the most direct signature of impulsive energy release, accelerated electrons, and the hottest plasmas in the solar corona and provide an important probe of the physical processes in solar flares. To infer the properties of the X-ray-emitting electron distribution, parametric models must be fit to the observed X-ray spectra. These spectra may consist of several components, e.g. excitation lines, thermal and/or non-thermal continua. In many cases, multiple models must be fit both to time- and energy-binned spectra, and account for instrument calibration. Such capabilities are crucial for making new discoveries in solar flare research and require a robust software analysis toolkit specifically designed for solar X-ray spectroscopy.

For many years OSPEX, an IDL-based toolkit, has provided the necessary tools for performing the above tasks. However equivalent functionality does not currently exist in Python despite strong and increasing demand from the solar X-ray community. While some X-ray spectroscopy packages do exist in Python, they do not provide solar-specific models, the ability to handle non-diagonal instrument response matrices, or are not optimized for time-sequenced spectral fitting, all crucial for solar X-ray spectroscopy. The absence of such tools in Python raises concerns about the long-term maintenance and accessibility of solar X-ray data analysis as the solar physics community moves away from proprietary software languages like IDL. It is thus a risk to maximizing the science output of past and future instruments.

We will address these concerns by developing a new Python package, sunxspex, specifically designed for solar X-ray spectroscopy. sunxspex will provide the critical parameterized solar-specific X-ray models categorized in terms of emission mechanisms, e.g., bremsstrahlung, radiative recombination, and line emission. These models are complex and go beyond simple parametric power-laws and exponentials as they often involve solar-specific approximations and optimized numerical integrations for efficient use of extensive atomic databases. They will be constructed such that they leverage pre-existing data fitting infrastructures, e.g. astropy.modeling. The models will be tested with simulated data and benchmarked against outputs from OSPEX. Moreover, sunxspex will enable time-sequenced spectral fitting. In addition to OSPEX-equivalent functionalities, sunxspex will enable simultaneous fitting of a single parameterized model to spectra from multiple datasets, e.g. from independent detectors within one or across multiple instruments. This has long been a need of the solar physics community but has so far not been adequately filled.

Finally, we will implement a data class specifically built for count-based data that can be binned both in energy and time. We will collaborate with previously funded HDEE projects (e.g., ndcube, SunPy) to ensure this class is suitable to as wide a range of use-cases as possible and is interoperable with existing data classes within the PyHC environment. The class will facilitate conversion between counts and photons, even when the mapping requires convolution with non-diagonal response matrices. It will be

instrument-agnostic and thus render sunxspex suitable for multi-instrument analysis including those onboard satellites, cubesats, sounding rockets and balloons, e.g. NuSTAR, Fermi/GBM, RHESSI, MinXSS, FOXSI, GRIPS, etc. In fact, this proposal is particularly timely given the new data that will soon become available from SolO/STIX.

This work will be done in coordination and collaboration with PyHC and adhere to the PyHC standards. All development will be performed openly on GitHub (https://github.com/sunpy/sunxspex), maintain continuous integration and unit testing, and include detailed documentation and examples.

Grant Stephens/Johns Hopkins University
EmmPy: A Python Library for Empirical Magnetic Field Models

Science goals and objectives: A global 3D description of the magnetic field and its time evolution is a critical component for a comprehensive picture of the magnetosphere system. It is often crucial to have a global picture of the magnetic field geometry and topology in order to contextualize and interpret in-situ spacecraft observations. For instance, empirical magnetic field models are used to map spacecraft locations to the ionosphere, so that in-situ measurements can be compared to auroral signatures, or for determining magnetic field line conjunctions of spacecraft observations. These models are often the only means in which to compute the adiabatic invariants for particle observations, and are also useful for tracing particle trajectories within the magnetosphere. Furthermore, the latest generation of terrestrial empirical magnetic field models utilize powerful data-mining algorithms to dynamically bin data, resulting in magnetic field models with an unprecedented ability to reconstruct the global dynamics of geomagnetic storms and substorms. This has elevated the status of empirical magnetic field models from being a tool in which to contextualize in-situ observations to being a scientific analysis resource in their own right. However, most of these models are written in FORTRAN using the standards and practices from the 1970s. As such, they are often impenetrable to many in the community, in particular to young researchers and graduate students. The proposed effort is to rewrite the latest data-mining based geomagnetic field model (TS07D) in Python using modern best standards and practices of software design, thus engendering these models to the upcoming generation of scientists and programmers in the heliophysics community.
Methodology: Despite the ubiquity in the use of empirical magnetic field models in magnetospheric physics, the source code is still largely impenetrable to many in the community. This is largely because the codes have been stuck in the 1970s. These models have been written using FORTRAN with the standards and practices that were common at the time. It is not feasible within the allocated funds to translate all these numerous models, so the effort will focus on just one, the latest data-mining based terrestrial model TS07D. The proposal team has much experience in updating these FORTRAN codes into a modern object oriented framework, as it has recently created a Java library including the T89, T96, TS07D, SST19, KT17, and a Saturn empirical magnetic field models. These efforts have not simply been translations, but rather a complete reengineering of the codes using modern programming principles from the outset. The team will leverage

this experience to architect the EmmPy library, which will be incorporated into the PyHC framework. Although only the TS07D model will be present at completion of the effort, the infrastructure will be in place so that additionally models can readily be added to the library. Given the long history and use of these models, there are now numerous iterations as they have grown increasingly complex over time (e.g. T89, T01, T96, TS05, TS07D, SST17) and for the variety of magnetized planets (e.g. the KT15 and KT17 model for Mercury, and the Khurana models for Saturn). These models share many of the same components, but as there is no centralized library, the revision control process involves pasting and copying the components from one model to the next. The proposed effort will not only deliver a functional coding of the TS07D model, but will establish a centralized library of commonly used components and a shared API for these types of models. This will allow the library to expand and incorporate more models over time, and will introduce a new generation of upcoming researchers to these codes.

Jon Vandegriff/Johns Hopkins University
Enhancing Heliophysics Python Library Interoperability by Adapting Common Data Models

We propose to create adapters to convert between the common data models used within low level Python Heliophysics libraries: SpacePy, ndcube/SunPy, and HAPI. Each of these libraries provides a fundamental data access capability, yet they each use a different internal representation for data structures. We will create Python adapters to seamlessly go between each data structure. The proposed adapters will allow these Python libraries to interoperate, i.e., data read from one library can be analyzed or visualized by any of the other libraries. For example, our proposed converters will allow SpacePy objects to be converted to ndcube objects. This will also encourage other library developers to either use these same data structures, or to provide adapters to them. Because these libraries cover a significant fraction of the science domains within Heliophysics, seamless data access between the libraries will support cross-disciplinary science, a major emphasis within the Heliophysics Data Environment. We will also create a report on recommendations towards a common Python data representation, the creation of which is beyond the scope of a small effort such as this one.

Robert Weigel/George Mason University
3-D Heliosphere Data and Model Visualization using Python

Physical processes in heliophysics are three-dimensional, but our analysis of measurements and simulation output is typically done using two-dimensional projections. The reason for this is often due to the difficulty in creating scientifically meaningful and useful three-dimensional renderings of measurements and simulation output. We propose to develop a suite of Python tools that allow scientists to easily view and explore heliophysics data in 3-D. The software development side of this project will involve (1) creating a set of Python functions that use existing SpacePy and HAPI libraries to read in measurements and simulation output and use the Python/ParaView API to render the

data; (2) creating a library of functions that produce 3-D renderings of data in a way that is useful for domain scientists; and (3) developing a set of functions that allow for common and basic 3-D data reduction and manipulation that can be used for exploring 3-D data renderings. Our initial target for the types of simulation visualizations that can be created will be magnetosphere simulation output in the CCMC CDF format, native SWMF simulation output, and Enlil simulation output. The software will also have the ability to read in spacecraft measurements from CDAWeb and SSCWeb using an existing HAPI library and overlay the spacecraft time series data on a 3-D rendering of simulation output. The second part of this project will involve the development of a set of recommendations and conventions for 3-D visualization tools in Python in collaboration with members of the PyHC community.
