Snakes on a Spaceship 2025

Snakes on a Spaceship Tutorial Production Software and Version Control – Leslie Lamarche

Slide 2

Good code for science. It's important. Why is it important?

It helps us share our results, improve reproducibility, and let people share their code, because we like open social code. But people still sometimes have trouble with publishing, both because there's a lot of steps to do it, to do it well, and also a lot of people still worry about misuse of code stealing of results, not being able to get every credit for it.

That's not necessarily the topic we're going to go over today, but I know it's something that people think about.

Slide 3

What I specifically was talking about is making the transition from the type of code you write to do a paper, to what I call production code. So, what is paper code? Paper is the code you have to write to do the analysis for a paper you publish. And, usually you're the only one who looks at it or you and maybe one other of your collaborator, it needs to function for exactly the period of time it takes for you to write the paper, do the revisions of the paper and have it submitted and published. Even if you are following good open code practices and share it, it's the likelihood of other people looking at it ever are on the lower side. It's something that you're working at constantly.

And production caught on the other is something that you're expecting to be used over and over and over and over again, for the period of possibly years.

And the main difference between those two is you have one thing where you can really get away with probably some sloppy coding practices, even if they're not something that's ideal. And then the other you really don't want to.

Slide 4

So why do you need to worry about doing protection code?

And this is actually important because it does take time to do things correctly. And even if we strive to do things most like, how correctly you need to do things, that's something to consider. So really to me, main criteria is this something that either lots of other people will use or I myself will have to go back and use repeatedly at lots of times distance in the future when I've forgotten how it works and that usually comes if there's a, you know, a model that a lot of people write a front processing data that's happening routinely, toolkits that are widely used.

So really anything that is shared widely enough or is going to be used long enough in the future that you're not going to be able to sit down at a table and talk through all the potentials users of it in detail and train them all.

Slide 5

So here are just some general things for moving towards production code. The first thing I love Jupyter Notebooks for arrived research, they are fantastic. You cannot set them up to run automatically though. Or even if you can, it's probably not very efficient. So move from Jupyter Notebooks to more scripts that are written that way for things that are being run routinely.

Packaging codes, which there's entire sessions or tutorials on how to do that, but it's something to look at, avoid hard coding things. So, if I'm doing a script for my research paper, I will absolutely hard code the file path to this data file on my computer. As soon as that turns into something that needs to be used more routinely for data processing, that won't not do. It has to be something that's an input to a function or fed in through a config file or something like that. You definitely don't want anything that, oh yeah, in order to make the code act like this, you need to go comment out this section or add that section or go to line 90 and change this number to that.

Document things, best practices are things, and then finally, version control is very, very useful when it comes to this. And so the rest of this, I'm going to talk about what version control is, just at a very high level because that's something I frequently get questions about.

Slide 6

So it's software that keeps track of changes in the code you makes. It maintains a what I call a semi-permanent record, which is it's not going, it's hard to change but it's not, it's hard to delete it but it's not impossible.

So, but you do have to try kind of hard to do. Git is a popular version, so you've probably heard of Git. GitHub, that's all that, what this version control software is, but there are others such as SVN.

So just as a pull of hands, who has used GIT here? OK, mostly everybody. Does anyone feel the need to go through the rest of this or shall we skip to the next slide? Go through-- OK, we'll go ahead and keep going.

Slide 7

Yeah, so again, why do I want to use it? Because it keeps the record and you can kind of take the state of code you're in and go back to any previous state in the history of the project, what I find that's really useful for is it emboldens you to make dramatic changes to your code, which is very handy.

So if I'm working on something, I have something that works. If you're using this, developing this with version control, you're a lot less afraid to go break it all by developing something new. And you then can do your development in a lot less cautious way. Because you know, if I completely screw everything up, I can go back to the last checkpoint and still have a working version of my software. Which is extra important when you're doing this production code because if the data still needs to be processed and I just broke the entire data pipeline by

trying to implement a new feature, well, okay, that's fine, we can go back and still process the data with the old version.

And the other thing is it allows me and my coworker and my other coworker to develop multiple features simultaneously and then bring them together, which is also very efficient.

Slide 8

So this is roughly just a high level schematic of how these things work. You have these different paths and each point is a checkpoint that you might be able to go back to in your code. So you start developing your code and then you kind of do what's called a branch to do certain features, and then that can merge back into the main one. And you can have multiple branches for different features. And that lets you kind of sink back in. And there is a lot more you can learn about this. It's kind of just a high level overview of why this is useful. As again, it lets you lurk on blue. You know, blue is your main thing. Then you can work on simultaneously green feature, yellow feature and red feature. And also, if red feature doesn't work, you forget about it. Although red is marked as bug fixes, but you can also do that. These are bug fixes and then we integrate it back.

Slide 9

Okay, this is an important point I like to bring because there's a lot of confusion about this with new people I work with. Git is a software that lets you do version control. You can operate it completely locally. GitHub is kind of an online platform for hosting Git-controlled projects. So you can use Git locally without ever touching GitHub. And in fact, I kind of recommend that as a good way to learn how to use it and see for yourself what it's useful before you start trying to develop code simultaneously with other people.

GitHub is a fantastic, amazing tool for when you are simultaneously with other people. And it's not that hard to use. In fact, they make it pretty easy to use. They have a lot of tutorials on it. But it is a service that is just one way to use Git in collaboration and provide a hosting platform for your code in a shared space, rather than having to set that up locally.

Slide 10

And then just the general workflow that you would follow. So for the two different ones, forget you modify a file, you do the staging file, which is a Git add, and then commit it. And that's what creates those little checkpoints, which were the dots in that previous schematic.

The process for GitHub is exactly the same, except at the end of it, and possibly you push your changes to the Git repo. And then at the beginning of this, you also, I should probably put this, you pull any changes from the repo too, so you can sync with other users or other developers.

Yes, so again, this is a very high-level overview. If you have not used Git before, this absolutely seems like way more work, and why would you ever do this, and this is a pain. And all I can tell you is I promise if you are doing development-type work and production-type work with lots of people and a long, ongoing project, it is absolutely worth it, and it's a useful skill to know.

So yeah, that is it and I will hopefully get things moving along. But I am happy to answer any questions if there are any for the group or just come find me afterwards.

GCMprocpy – Nikhil Rao

Slide 1

Good evening, everyone. I'm Nikhil. I'm a software engineer at the High Altitude Observatory at NCAR. And today I'm talking about GCMprocpy.

Slide 2

So GCMprocpy is a Python package for post-processing and visualizing outputs of TIE-GCM and WACCM-X. What basically means is to build plots and analyze the data.

Slide 3

Initially, we've had tools like tiegcmidl and tiegcmprocf90**.** These tools were built in Fortran. New researchers don't really like Fortran. It's difficult to integrate into modern data workflows. GCMprocpy uses Python's extensive scientific libraries and it's a lot more user friendly compared to these tools.

Slide 4

GCMprocpy was initially called TIEGCMpy. It was limited to analyzing outputs of TIE-GCM but we renamed it to GCMprocpy and added the capability to analyze WACCM-X outputs as well. We also adopted a modular design that I'll talk about in a later slide and added capabilities for a GUI.

Slide 5

This is just the timeline of a release. Our first release was like a year and a half ago. October 23, we renamed it GCMprocpy. Last year October and last release out of the GUI April of this year.

Slide 6

There are three main ways to access GCMprocpy. The first is a GUI, then there are function calls, API, and we also have a command line interface.

Slide 7

We've designed GCMprocpy in a way that it's the one-stop solution to understand the outputs of these models. So you have the data exploration functions that help researchers understand what the data is. Things like time steps that are listed, the variables, the data structure, then we have data manipulation functions. So, TIE-GCM does this thing where it outputs time steps in day of year, hour, minute, second, which is not ideal. So things like that come under data manipulation where we put them in numpy data 64, arrays, and so on. Data array generators are basically array generation functions that, for example, a user wants data for a variable at the particular latitude or longitude throughout the pressure levels. Data array generators give that to you.

Emissions calculators. Emissions functions basically functions that use multiple variables to calculate and give you data arrays for it. Then we have plotting routines that create static images and movie routines that use the plotting routines to create a time-lapse for animations.

Slide 8

These are just the plot types that we have available. A lat/lon plot, a pressure level variable.

Slide 9

Pressure level longitude, pressure level latitude.

Slide 10

Pressure level time, latitude versus time.

Slide 11

GCMprocpy also uses AutoDoc. It is a Sphinx extension that automatically helps us generate documentation. So what that means is under a function, you have Docstrings that you add that basically explains what the function is, its inputs, its outputs, in a particular format. GCMprocpy uses Google Style Docs strings. So it basically automates the process of documentation in a way where you just edit this part and your function as you update it, edit or add it. You push it into your master, and we're using ReadTheDocs that basically, once you push it into master it automatically updates the website and gives you a beautiful HTML with updated documentation.

Slide 12

This basically goes over what I was talking about earlier, the modular structure of GCMprocpy. Let's take an example where a developer or researcher wants to add a new emissions function. So all they would need to do is create the new emissions function and tell the emissions controller function of what variables the emissions function uses. They will not need to edit any other part of GCMprocpy. This just makes it a lot easier for developers and researchers to add different new ways to calculate emissions or other plotting routines.

Slide 13

Future development: we still don't have a good testing framework. Need to develop on that. We're working on adding movie routines to the GUI, adding polar plots and a lot more types of plots and always adding additional emissions calculations from the tiegcmprocf90 and a lot more performance optimizations.

Slide 14

That was mainly it. We have a Google Collab tutorial that will help you guide through how GCMprocpy works. We have a GitHub link and read the docs documentation. We have a MAGE tutorial on Thursday that we will be using GCMprocpy on. So do attend that.

Slide 15

Thank you.

rbinvariantslib: Radiation Belt Invariants Library – Daniel da Silva

Slide 1

Okay, hi, I'm Daniel da Silva, I work at Goddard. I'm going to talk today about rbinvariantslib, this Python package for some tools to calculate the adiabatic invariance in Python from MHD and empirical models that work at some magnetic field.

Slide 2

Yep, so this tool provides -- this package provides tools for radiation-developed researchers to calculate particular parameters that are useful for modeling particle trajectories. These trap particle trajectories. These parameters are K and L*, and the computed from magnetic field models of the Earth's magnetosphere and some information about the particle state. So if you don't really know what this means, I'll have a couple of slides and I'll explain it. But what this practice does that is previously not done is that it's in Python, and it provides tools for both empirical and MHD models of Earth's magnetic field coming from like space where they're modeling framework or the second Tsyganenko models. And we have a peer-reviewed publication that covers the detailed numerics of how we compute these parameters.

Slide 3

So I promised to explain what the package computes and how that works. So the radiation builds its three periodic motion of the trap particles. They gyrate around in a helix. They bounce along field lines and they drift around the earth. And each one of these periodic motions has a conserved quantity as the magnetic field of the earth changes. And it's interesting for analyzing spacecraft data to compute these conserved quantities of motion called the adiabatic invariance.

Slide 4

We can skip this slide.

Slide 5

Yeah, so the model API, the package API is pretty simple. It has a models package where you can load instances of a magnetic field model class. Here you load a space where they're modeling framework CDF file from this path right here. And then you call a function called calculate_LStar with the magnetic field model and the starting conditions of the particle and you get back a value.

Slide 6

So it uses some algorithms that will go over in the paper but calculation of the first adiabatic invariant M is very easy, it's a direct analytic equation. Calculation of the K invariant, we trace a bounce path along the field line, and do numerical integration. That's a standard method of doing it. The drift invariant L* is the hardest one of all the three quantities that it

calculates, and we use the Roederer method that was prescribed in the 1970 paper. It's an optimization problem that if you want to avoid doing it yourself, you can. It takes some optimization and exploits some symmetry in the problem to do it more computationally efficient.

Slide 7

We support the empirical models on the grid. So the second Tsyganenko 2005 model, the T96, second Tsyganenko 1996 model, and loading data from MHD models, based on space weather framework and LFM, hopefully, GAMERA soon, and data from arbitrary structured grids if you just pass arrays and construct the magnetic field model directly.

Slide 8

So that's it. That's a plug for the package. It provides Python tools for calculating adiabatic invariance, radiation, and other research. It's open source and it has a Sphinx documentation site on readthedocs.io. It's on GitHub, it's under the Apache license. And if you have any questions or end up using it, it's very useful for people of multiple open source projects to hear back from people. It helps prioritize what's getting used and what people are excited about. So if you do end up using it or have any questions, please reach out. Thank you.

Kaipy – Michael Wiltberger

Slide 1

I'm Michael Wiltberger. I am at NSAF NCAR and also a member of the Center for Geospace Storms team. The people that have really been helping out with me on this are Nikhil Rao, who you just saw earlier on talking about GCMProcPy and Eric Winter, who's on John's team at APL, and has been helping out with the Python tools for it.

Slide 2

So, Kaipy is a Python package that we've created to help with the analysis and visualization of MAGE simulation results. We've been working really hard in past month or so to get our first, or not in our first, but a major update for Kaipy out the door. That includes spending time getting a nice logo. So we're very pleased that we've made it into the realm of real packages 'cause we have a logo.

And we've also been doing some modernization efforts. We've pushed forward. We had been doing a lot of work on older versions of Python. So now we're supporting Python 3.10 to 3.12, and it can probably work beyond that. We just haven't tested in that space at the moment but we've gotten out of 3.8 which was a bottleneck for various reasons.

Slide 3

A very similar GCMprocpy, we've got it set up to use both an API and a command line interface. It is available today now for you. We are on PyPi as a pip installable package, you

can just do pip install, Kaipy and you should be good to go after you've gotten, of course, your favorite environment setup for Python.

The QR code there works for the the PyPi package and very much following along the path methodology that Nikhil educated me in in setting their package up we are using AutoDock strings and Sphinx to create the documentation and we've got that on ReadTheDocs available for what we have there for both our command line scripts and the API itself.

Nikhil has been working really hard to educate a poor dumb scientist like me about the proper ways to do these things. And we've made all of the command line scripts part of the package so that when you do the pip install, the command line scripts are available to you. You can just find them without having to do path updates and work easily for what we have going on in the system itself.

Slide 4

An example of the command line scripts is say you've gotten some runs that you did at the CCMC or somewhere and you want to be able to create a visualization of them. We've got a simple function called msphpic that has a nice little help string to it there that has various options to it. Yes, we've got too many options, but we're scientists and we need to be able to tweak and adjust everything in the system there. But we have that there you run that script you can see a little snippet of the output of the script running over there. It's processing through the file reading in the various time steps and actually in this case it's reading in three different models to create this plot that's coming into play there.

So it's reading the GAMERA data files to get that great bulk of the magnetosphere. The inset in the lower left is the output of the inner magnetosphere model. And the insets on the right hand side are the field aligned currents from the ionospheric coupler solver pieces of it.

So this one script is drawing upon the breadth of the routines that are present in the API to give you one nice little summary. And in fact, one of the things you probably can't read in the little ICandy screen there is that there's a minus VID option to the msphpic pick, and it will actually, in parallel, create all of the process, all of the steps in the files that exist, and roll it up into an MPEG-4 movie for you so you can get a summary of the results from it from that side.

So the command line provides nice little easy interface for it, but if you are your own scientist and you don't like the way we've customized or the values that we provide on end to it, you want to use the API.

Slide 5

As an example of the API, just a quick little example here, we have a function, there's a series of pipes that we've created so that you can create a pathway into getting the data. So this is what you need, is the routine here, right? You basically specify the directory and the base name of the file that you want, then you run this pipe object and it's going to return the ability to give you NumPy arrays for the data set that you want. And then you can go run your radiation belt and variant on it.

Slide 6

You can go ahead and turn those arrays into a into a plot using the same essential routines that we used in the command line script to do that and go forward with it from there. But the idea behind the API, you fundamentally, is help you make plots but also just bring that data in from all of the different models that are part of the MAGE system into an NumPy array and then go crazy do what you need to do to do the analysis that you want for it from that side of things.

Slide 7

I'm sorry this got a little shifted. We are trying to do some best practices for it. One of the things that was very helpful for us in moving from Python 3.8 all the way up to 3.10 and 3.12 was the implementation of unit tests. We're using Pytest as the package to support that. A little bit of the example of the output of the tests that we've run through here. You see that there's a bunch of tests that are being skipped. Those are the ones that are actually part of my testing for CDAS-Web and it takes a long time to send the query to CEDO Web for each of the satellites and to pull those back. So I'm not routinely doing that, but there's a minus run slow option in the testing routine, so you can run those tests and get that going from it from there.

The other thing that was on here is that you see that I am not a very good Python programmer and I need help. So I use GitHub Copilot to help me out with that. And there was a lot of help in the development of the unit tests and getting the things set up from Copilot on that. I find it to be very useful. I don't know what other folks' value is on that, but let the AI help you out when you can on those kinds of things certainly helped us get this testing stuff moved along and move forward to that side of things.

Slide 8

As Nikhil alluded to at the meeting here today, we are this week we're announcing the major open-source release of the MAGE model, which is the underlying model support that both GCMProcPy and Kaipy are using for that. The GitHub site for the repository is live, as is the documentation. And we have, I think it's supposed to happen today, or maybe in the next day or two, the CCMC folks are supposed to turn the switch on on the slightly earlier version of the MAGE model that has the Rice Convection model as the inter-magnetosphere as opposed to our new completely rewritten RAIJU model.

But Thursday, 10 a.m. I don't know the room, but Thursday, 10 am, in that in that first breakout session will be our hands-on tutorial where we'll go through the installation process for the model and get you guys using Kaipy and GCMProcpy going forward with that. Happy to take any questions.

Accelerating SuperDARN File I/O with Rust – Remington Rohel

Slide 1

Hi, yeah, so this one will be a little bit different focus. I'm with SuperDARN Canada. I've been in Saskatoon and I took on a project to pick up the reading and writing of SuperDARN

files. It was really bad, slow, and so this talk is going to be a little bit structured more on how I used Rust to make it faster. Kind of my philosophy is it's a tool, not a solution for every situation necessarily. And I just kind of wanted to highlight some of the things I liked about it and some of the ways you can use it. And if you find that useful for your own purposes, great. We can chat about it if you're interested.

Slide 2

I'll go through the motivation a little bit about writing Rust code for Python and then some of the testing and benchmarking that I did.

Slide 3

So pyDARNio, it's a Python package for reading and writing SuperDarn files. It's used by Python, which is the community's developed software for reading SuperDarn files. It's used by pyDARN, which is the community's developed software for plotting SuperDarn data. So most scientific stuff you see will use pyDARN. It's really slow though. It's a pure Python implementation. It usually takes tens of seconds to read one day of data, which is 100 to 600 megabytes. And often people want to read months or years of data. So it can be a big bottleneck. So the goal was just to speed it up, essentially.

Slide 4

So the basic usage, just import pyDARNio, get your string path to the file, and then throw that into this SDARNRead class, and then you can select your specific file type. So in this case, “fitacf” is the name of the SuperDarn file structure, and just read that in, and then very similarly for writing. So it's a pretty kind of one-shot sort of sort of usage.

Slide 5

The problem is that SuperDarn data files are bespoke binary data form in that. So kind of the basic structure is it's just composed of a concatenated list of records essentially. Each record has some metadata about it including how many scalar and vector fields are stored in the record, and then the scalar and vector fields are also shown here where they're just a name, a type, and then an array of data. And then with the vectors, there's also, you know, how many dimensions there are that you can interpret that array and its shape. And yeah, so essentially you have to read through the file byte by byte. There's no hierarchical structure to it. And that's kind of what leads to the bottleneck.

Slide 6

So Rust is another programming language. It's a compiled language. It gives you speed similar to C. But it's got a few more guardrails, like enforced memory safety and stuff like that. It's kind of composed of four main types. There's primitives like integers, booleans, et cetera, and then structs that can have associated methods with them, enums, similarly you can have associated methods and then traits which are kind of meta classes that you can inherit from. And it essentially enforces you to have certain methods if you're implementing that trait. So it's pretty easy to create Python bindings for Rust projects. There's a really good couple of packages that make the process simple. And then as a developer, there's kind of two ways somebody can install your rust package for Python if you set it up correctly. They can either clone your project and build it locally for their computer, or as a developer you can do that process for them and generate a Python wheel file that you can upload to PyPI and then

it's just a simple pip install. And there's actually a few other tools to use GitHub Actions to do that automatically so that it's even easier and you don't have to hassle with that. And in that case, then you just have a Python API and your user installs with PIP and then uses it and is none the wiser that it was even written in Rust. So there's that usefulness to it.

Slide 7

So I'll just kind of try and skip over this stuff pretty quick, but essentially the Rust code is structured somewhat like this: standard imports at the top, and then your doc strings just with triple backslashes before your--or forward-slashes, I guess, before your functions. There's a special decorator to specify that this is something you want to be exposed to the pipe on side. And then a special function that you have to generate just once, like the root of your project in order to expose all of these functions to Python.

Slide 8

And then when you're actually making these functions, again, similarly, there's a decorator. And then there's a few more kind of levers you can pull for specifying the name of the function when it's exposed to Python or the text signature for variable hinting or whatever. And then you just write your Rust code. And so it has to return these special Py-result types. And then that'll automatically get converted to your Python types when you're using it.

Slide 9

It's really easy to parallelize. There's a really nice package written in Rust. So kind of in the middle of this code blob here, there's two blocks that do the same thing. The first one is single threaded where you just, I have set up an array and you're just iterating through the array and calling some function on the elements and storing the result. And all you have to do is prepend your functions that you're using with par for parallel and it will automatically distribute the load over the cores of your CPU. With very little hassle, you don't know how many CPU cores you have, like with multi-processing and Python. So I found that really, really handy and convenient.

Slide 10

And then to actually get it to Python, you also have to specify a few fields in the top level toml files, pyproject.toml will add in some specifications in there. And then there's a tool called maturin that you can use to compile the code and install it in your virtual environment that has a corresponding GitHub action as well to automate building and generating and uploading wheel files for Linux, Mac, Windows 32-bit, 64-bit, etc. And for the SuperDarn specific package, this is done in this project called DMAP, the bottom link there, which you can go check out if you're interested.

Slide 11

On the Python side, the user side, you just import this, so DMAP, the name of the package. And then, you know, you can see with the Dunder Doc, the top level doc string that I created for the module or for the specific functions. Those are all accessible. So it really just handles that conversion for you. And with this package, so I've done the work in a REST project and then I'm re-exporting those functions within pyDARNio so that the wider community that only knows about pyDARNio doesn't really feel any ripples, but they just get the benefits.

Slide 12

Just quickly, through-testing is really nice. You rate your functions and just decorate them with a test macro, and that'll just automatically be detected when you run your test. That's kind of the standard rust way. You can put tests in the same file right next to where functions are defined. That's really great, it's local. You can even embed tests in your docstrings. So you'll know right away if you've changed your functionality and didn't update the docstring, then you'll get a failing test. One thing to note though, only the functions that you expose to Python can be tested in Python. Everything else you have to test in Rust if you're not making it something that you can use from Python, or that's hidden away.

Slide 13

And then for benchmarking, I found this really fun: there's a nice package called Criterion, and essentially you just set up some functions and a bench group, and then you can run simple command, cargo bench, and that'll run the benchmarking and generate these nice HTML plots, and you can compare over multiple invocations and see how your speed has changed and compare.

And then there's also another external tool I found really useful called Hyperfine. That's just like a command line, one shot, call benchmark this function, and then you can add two or three, and it'll compare and see which one was the fastest and generate some statistics for that.

Slide 14

And I guess this is kind of some of the highlights, some of the results from my work with the SuperDARN IO for a couple of two different types of files and two different sizes of file. So the top is a small file. Left side is kind of a lower level data product that's bigger but less complex and then the right side is a higher level data product that's got more fields, so it's more complex. And essentially, when moving to a backend that used Rust, we got 10 or 20 times speed up, even if it was single threaded. So it was really, really great.

And some of the other SuperDARN groups have started using that in their workflows, like for their website, especially doing user plotting but this was their biggest bottleneck. And they've really appreciated it.

Slide 15

Yeah, so the Rust code - very fast. The one hang up is that it's slower to pass the data. You have to cross the boundary from Rust to Python. So that is, I guess one downside from this, but yeah, it's been really good.

Slide 16

So yeah: summarize. I found it to be a useful tool. I really liked the results that it gave me, and if anybody thinks they might like to use it, yeah, come and chat. Thanks.

PyAuroraX, and more – Darren Chaddock

Slide 1

Okay, hi everybody. I'm Darren Chaddok from the University of Calgary. I'm going to be talking about PyAuroraX and a bit more.

Slide 2

So I'm just going to first introduce a little bit of the scope of things. Our group operates 121 instruments right now: Canada, Alaska, Greenland, and newly in Antarctica. We run in five different data centers, primarily into, but we've also expanded for a few tools. But we have a petabyte of data now, 100 terabytes every year. So it kind of just keeps racking up.

Slide 3

Out of everything that it takes to operate these instruments, this is a graph from Emma Spanswick that shows all the bits and pieces. PyAuroraX is, it falls into this, into the red box on the bottom right. It's our data tooling and our utilization aspect of our platform.

Slide 4

Yeah, so where it fits in here, it's a component of the AuroraX Data Platform that provides data access and analysis support for All-Sky Imager data. We have a few arrays that we operate, THEMIS ASI, might be the one that everybody is familiar with. TREx RGB be the one that everybody is familiar with. TREx RGB is another one tha's quite widely used. Newly just today, we released an update for PyAuroraX that also includes support for the new SMILE ASI cameras, which are the full color replacement for THEMIS ASI. So that data is now public.

We also have a few more, but the AuroraX Data Platform also includes doing conjunction searches and ephemeris searches, which also have some machine learning integrations into them where you can find, you know, times where certain, maybe the clouds are filtered out more if we're looking at a certain type of Aurora, you can do these types of things. We also have web visualizations and inside of all of the AuroraX platform it's PyAuroraX and there's an equivalent in IDL as well.

Slide 5

So it can read data and download data, L0 and L1+, we have a couple of different summary products. It provides some common analysis functions. One of the important things that we do because we expose the raw data. Once it's read in, you can feed that raw data into the routines for analysis, but you can also go off and do whatever you want with it. We know everybody likes to do different things, and we don't want to box people into corners to process the data in the way that we usually process it. So we want to expose that raw data and let people do what they would like.

You can also do conjunction searches and ephemeris searches that would interact with the AuroraX API.

And again, filtering using cloud and auroral type machine learning models.

And you can also use the TREx AuroralTransport model that's developed by Jun Liang in the audience here. So you can interact with that model as well using the tool.

Slide 6

So I'm just going to run through a few different examples of things. This is a keogram - I think it was a 2D time lapse. So you take a single slice through each image and stack it horizontally. So on the bottom axis, this is now worth of imagery data from Lucky Lake, which is in Saskatchewan. And this is actually from the brand new SMILE ASI cameras. So you can make keograms.

Animation 1

You can look at the raw images just as, I just want to see this picture in time.

Animation 2

You can make keograms as well using magnetic latitudes or geographic latitudes. This is for the TREx-RGB systems.

Animation 3

Montages: You can show a series of single ASI images all stacked together like this.

Animation 4

You can overlay boundaries, latitude and longitude lines on top of the all-sky images. This is in magnetic coordinates.

Animation 5

And you can also take a region in space, and from maybe an hour of hour worth of data, this is what this example is, and you can say, okay, what is this red box? I want to see the statistics of that area.

This is, I think this is a mean, like an average of that of luminosity in that box. I think that's what this example is.

Slide 7

You can also put things onto maps. This is using Cartopy - as we've talked about a bit. So this is on the left side there. That's TREx RGB. And that is five different imagers. Six different imagers, sorry.

And on the right side there, that's two of the SMILE ASI cameras. So you can stitch these together and get a more global picture of the scene on that point in time.

Animation 1

This one is a little bit of a rendition of that, where it's multi-ASI networks. So you have RGB on the left side and the further right side. And in the middle there, you'll see it's black and white - that's THEMIS.

And then on the top right, that's red line that's our REGO camera. So you can stitch multi-network data together onto the one map.

Animation 2

You can also throw in satellite data, the position of it. You can pull it from a couple different sources. Our example shows it pulling from the AuroraX database, which actually does CDAS WebData--or sorry, SSC WebData, and we add some more metadata.

Animation 3

This is an example of a mosaic with RGB and then the TREx spectrographs laid over.

Animation 4

And then this one is a new functionality that we just released, being able to make field-of-view plots, showing not just the data but where the sites are. And we're going to bake in some availability into this so you can say, okay, I want to just show the site map for this point in time. And it'll show you whatever data is available. That'll come later.

Slide 8

This is an example of our grid files. So these are more for the modeling community that we've been working on, where making mosaics can be a bit of a labor intensive, or computationally intensive rather, making these mosaics with the raw data. So we've been, we've pre-generate them.

A bit of a lower resolution, something that's been, been seeming to work alright. And so on the top is the high resolution and on the bottom is those grid files.

So our tooling, you can read in these. They're quite a bit smaller than the raw data. And you can just be off to the races looking at those.

Slide 9

This is an example of TREx ATM. So on the left side, those are all the outputs of one of the functions. And on the right side, this is a conductance map that Liang made. So you can do these types of things with the ATM.

Slide 10

We have also released PyUCRio. This is another library in our platform that provides same kind of, similar types of access and tooling for our riometers and our hyper spectral riometers. We have a Python library, we also have an IDL library, they have the same functionality.

This is more time series data. It's a little bit easier to work with than the images, but they'll have their problems.

Slide 11

So to learn about how to use PyAuroraX and all of our other tooling, I always recommend that people go to data.phys, our open data platform, open science platform. And on this page, there's a couple different things that you can look at.

Animation 1 & 2

So you can look at the data set documentation. That page has information on DOIs, all of the image or instrument arrays, and information about what they are, where they are, who you can contact for questions.

Animation 3 & 4

Working with our data, so this page has lots of examples. That's where our big example gallery is. And so you can go into there and you can scroll down to the CribSheets section and there's tons of examples.

Animation 5

This is a list here of all the other resources we have. You can look at our data portal. There's a link for that on data.phys. SwarmAurora, AuroraX platform are tooling, and you can also use SPEDAS for the THEMIS and REGO data. That’s been there for a while.

Slide 12

So I’m going to end on this one, which is kind of where we’re going. So Emma Spanswick has a new project: GDC ground. It's 134 instruments across 27 sites. It's a lot of RGB cameras, GNSS instruments, magnetometers, 16 red line cameras and eight spectrographs, lots of riometers and some Fabray Perot’s.

So we'll be, I think the money started flowing for this in November last year. We've already started development on the red line cameras, and we should be deploying instruments in the next year or two as we kind of pick along at them. So all of the tooling that we've been showing here for PyAuroraX, all these new instruments will get baked into that. So it'll just continue to operate and get new data all the time.

Slide 13

All right, that's it.