Snakes on a Spaceship 2026

Coding for Operations – Emily Morgan, presented by Leslie Lamarche

Slide 1

Okay, so coding for operations is the tutorial this year. And this was supposed to be given by Emily Morgan, who is a NRL scientist and is highly qualified to give this presentation. And she very much wishes she could be here. But you get me instead.

There's a lot of ground on this topic. She's kind of going over what just scratching the surface and the most important things. I think this tutorial could easily be three times longer.

Slide 2

The priorities of this: write code that is helpful, not hostile. We've all been there, given a style of code, maintained for years and years that's evolved beyond its original purpose. Authors change, conventions change. It was translated into different languages. But now it's your problem.

Trying to trace this back is a Sisyphean effort. And then this presentation is really about how do we avoid that from the get-go? And how do you develop coding practices that were useful to stay away from that, particularly if you know you're developing code that will eventually be transitioned into an operational sense. So away from you, the research scientist, and into people who want it to work and be used and not think about it beyond that.

Slide 3

So, for our example, please meet Elle, the elephant.

One of the things that Emily really wanted to emphasize in this is that we could run through all the different programming sins that we all know we're not supposed to do, and we all do anyway. But instead, kind of to more collectively think about how we do things and how we can improve what we do with the capacity of what we do with the capacity that we have.

Consider Elle is our end user - our operational transitioner partner. For our projects, we want to help her to ease her understanding utility of the code we give her. One thing in particular to note is that operations can be stressful and solutions are usually demanded under duress and likely involve multiple people who are agitated. You're never given enough time, you're never working with an advantage, and others, like people more senior than you, want a problem fixed three hours ago. So our job as the people originally developing the code is to try and make Elle's job easier, trying to solve these problems once you are outside the loop.

So she's a scientist, probably with expertise in a related field but not always. Monitor, assess, validate tool and its output, troubleshoot failure. And then another thing too that's important to realize is your end user probably has limited options - some limitations both in their knowledge

of the field, the knowledge of the code base, and possibly even the tools available to them. Often IDEs perhaps are not available on the system they're working on and other things. So our hypothetical code is called CfO: Coding for Operations.

Slide 4

This is kind of how days in operations go: you come in first thing in the morning, before you even have your coffee, have an email:

Message below is read aloud

> Elle, there was a failure overnight with CfO.
> The message was "error in calc_avg".
> Restarting gave the same error. We're skipping future instances until this is resolved.
>
> V/r
> OPS

This could be an email from a coworker, could be from a script. Assume they're probably giving you all the information they have. One of the questions at this point is, you know, is this helpful? What could be more helpful?

Asks audience for responses

- Error logs, okay, specifically what in error logs? The type of error, good.
- The last thing that executed successfully, good.

Slide 5

One of the things that really is very useful that people apparently often leave out is the time of the error. And where it occurred. And I believe when you're doing operational things, that can be very useful and very helpful for tracking down what happened? So better:

Message below is read aloud

> Elle, there was a failure with CfO at 0345Z in task "process_temps_global". The message was "error in calc_avg". Restarting gave the same error. We're skipping future instances until this is resolved.
>
> V/r
> OPS

Ideally, this is automatic email, it's not always possible. There are more and more, obviously more things that could be done.

Slide 6

Error logs: so now what makes a good error log? This is some code provided from something Emily is currently working on that is getting an error, an hour into the task being run that just says "error here" is very frustrating.

Reads code aloud

```
1 for i in range(0,3):
2     if i in segs:
3         if prev < thresh:
4             val = prev + dp*i
5             return
6
7 # If we are here this is a problem
8 sys.exit('Error in pthresh')
```

Problems with the original code has actual multiple instances. So you can't see this, but I guess it had multiple instances of the pthresh function in the source code. And it is unclear if the issue is on line 2 or line 3, which makes it very confusing to troubleshoot.

Slide 7

Instead, this is kind of a slightly better rendition of it. And the additions add distinction between each logic step and their associated errors, provide actual values for the variables being tested. So you can see if something was a bad value. Provide values at the threshold in case that is improperly set. So if you're testing if something's below the threshold and you say, okay, well, it errored because this is my value, and this is my threshold, it's actually important to know both, because who knows what threshold is set to. Well, you assume what threshold is set to, but you don't actually know.

Slide 8

Error logs. This is a simple function to calculate the average.

Reads code aloud

```
files = glob.glob(os.join(here,'*dailytemp.txt'))
for days in files:
    try:
        temps = read_from_file(tempsfile)
        avg = sum(temps)/len(temps)
    except Exception:
        sys.exit('Error in calc_avg')
```

Who sees problems? Who has ideas?

Asks audience for responses

- Bring in logging module
- Raise a different type of error, like a different type of error class

Slide 9

So yes, I think those are both very good points. Add breakpoints, if there are files to read, so possibly the globbing files is a very good place for errors to occur.

Specify the exception, try/except block, prefer instance, very good. Add files of interest and suggest potential hazards, likely as a result of wisdom gained from developer testing. And then script/task name for easy reference in the problem if the problem is less straightforward. So again, instead of just error exactly, more information about what the error is and where it related to.

Slide 10

What makes a good error log? First, user doesn't have to open the source code to figure out what the problem is.

Second, tell the user exactly where the process occurred and give them a decent idea why.

Three, provide enough context for the user to give the user the solution or a possible solution.

The very important thing is your end user, Elle the elephant, does not want to gain intimate knowledge of the entire code base. Her job is to run the code and then when it breaks, fix it quickly and keep running it. You absolutely do not want your code designed in such a way that the only way to fix a problem, especially, you know, and sometimes it will be a higher-level problem that needs more debugging, but you want that not to be a standard thing. That things fail, and they can be fixed and restarted without someone needing to have as much information as the developer does about how the code is designed.

Slide 11

This is kind of a sidebar on IDEs, which is again going into some considerations of how your end user might not have all the tools to their disposal that you do. And while they're trying to bug fix and troubleshoot a stressful situation, they may be writing this on a remote machine. They may have information, classification, and/or control issues, and a variety of other things that result in, they cannot just pop open Spyder and have things work. And maybe they can, but maybe they can't.

So you need to be able to reproduce the error if not OPS. I'm actually a little confused about what these mean.

It can be difficult, and impossible because of data retention, and time consuming. You're the expert: it's much more efficient if you do the thinking. The developer does the thinking than the user, then the end user. Basically, as the developer, you'd be the one to sit through thinking about things. Don't expect when you pass this off, the end user to want to sit and debug. To the extent you can do the preliminary thinking and debugging and troubleshooting yourself.

Slide 12

Code comments. Code comments are good.

So we figured out where in the code the problem came. Now we need to figure out why. And if you're needing to figure out why, and as we previously discussed, you do not have an intimate knowledge of every single thing that was done in the code. Comments become very useful.

We have the bad, the ugly, and the good.

The bad, vague, doesn't provide helpful information. Could be removed without changing the user's understanding. One of the things that apparently is a pet peeve is "it should be obvious from reading the code". No, it is not obvious from reading the code. Write what your functions do. Do not expect someone to have to understand code to understand it.

Slide 13

Next is ugly. You can just as much as not enough comments, have too many comments. So each line has an entire paragraph dedicated to it, that describes in great detail. "Here we will loop over each segment of the whole, defined earlier in 'set_slices', which are integer values, and compare with the constant value 'thresh', defined either in the user config file run_style.cfg or defined as one of several default values".

That is going to be messy and onerous to read through quickly. So, resist the urge to overexplain, focus on solutions, not explanations.

Slide 14

And then finally, a better example kind of loop over segment. So just short but consistent and precise explanations of what your code is doing.

Any other thoughts, ideas? Who has feelings? Because I know everyone has looked at commented code and poorly commented code. Who has their own pet peeves?

Asks audience for responses

- Yes, okay, so no placeholders. Don't publish your code with the placeholders put in, which definitely has happened.

Yes, we've all been there. I know there was a code when I was in graduate school that I fortunately didn't have to work with, but apparently the comments were in three different languages because of generations of different graduate students who had worked on it. And it was rather confusing.

Slide 15

Organizing your modular programming. That's a thing we've all heard about. We all kind of feel like we should be doing - which you should be doing - but there are many ways to do things in a modular setting, and so you kind of have to choose a way that you do, and there is nuance to it. So you should do it. But remember, there is some nuance to how you do so it's not just, yes, I'm doing it and it's done.

Some hazards to watch out for when you're modularizing: abstraction of information makes it difficult to trace variables. If you're having to like backtrace a variable through ten different functions to figure out where the error is - I've done it - it's not the worst, but it can get confusing.

And then error messages can lack specifics. It only knows what scope you give it. So if it's an error in a function, that's an abstraction of something else that only has two of the parameters involved, sometimes it's hard to give it all the information to properly trace back where the error actually came from.

Slide 16

For instance, so instead of organizing scrips solely based on data types, so here are your satellite data, here's your in situ data, here's your model data. Perhaps it's more useful to separate post-processing steps into their own groups.

So here's the preprocessing module and then the post-processing module where everything happens together. This is post-processing example because it's often a step that encounters a lot of problems for routinely run code.

The main point is to be thoughtful in your design process. Don't be afraid to restructure the code if it's outgrown its previous organization structure. And I've definitely seen this in practice, where code was developed for one thing, and then as people apply it to different things, it becomes more and more of an effort to shove it into the original modularization structure and eventually you kind of have to decide, okay, wait, we thought we were going to use it for this, but now we're much more frequently using it for that and a different structure of modularization would make things much easier.

Slide 17

Okay, your code communicates to the user. In logs, error messages, documentation tell the user what happened, where it happened, why it happened. This is kind of one of the main points of this talk.

I think I'm a little slow on time, so I'm going to skip a few here.

Skips several slides

Slide 20

Communication with end users. Again, think about who your end user is. Think about how you interact with them. Sometimes you will not know them at all, especially if it's open source code. Sometimes you might know them, but you don't interact with them. Sometimes you're going to interact with them regularly. And depending on that situation, that definitely changes the approach you take with all of these things. And the importance of giving them the power to correct things themselves.

And one thing specific, encourage all of us to try and be good and get better about receiving constructive and destructive criticism. So it is okay if someone comes back and says “your code does not work or is frustrating” or “I hate it for X, Y, and Z reasons”. Hopefully they will be nicer than that, but fundamentally, yes, it is an ongoing development process, and we should all try and not take it personally when we worked hard on something and someone still finds flaws with it.

Slide 21
This is final payout of this. Hostile versus helpful code documentation is worth the effort, pay it forward. And this is something that I feel I personally feel very strongly about because half the time the end user is yourself in the future. So when you document things, it makes things so, so, so much easier to do it again in two years. Because otherwise I have no idea what I did and I have to rewrite it.

Find some conventions that work for you and your use case. There's lots of things about conventions, what should be done. But it really is all user specific and application specific. Don't feel like you have to be tied to specific ones. Modify them so they work for you.

pyIntensityFeatures – Angeline G. Burrell, presented by Alanah Cardenas-O'Toole

Slide 1
Hello, my name is Alana Cardenas-O'Toole. I'm a graduate student at the University of Michigan, and I will be presenting this work on behalf of Angeline Burrell, who's at the Naval Research Laboratory. She could not be here today, but she is online to answer questions, and I encourage you to talk to her or read her paper about this topic of pyIntensityFeatures.

Slide 2
The high latitude magnetosphere/ionosphere coupling varies based on the region. We can see in this image of the Earth with the magnetic field lines, the open and closed field lines, the open in red, that when you have open field lines, you're inside the polar cap.

Then for the closed field lines, the aurora oval are higher L-shells, and then going down from there, we have lower L-shells outside the aurora oval. So, there's different mechanisms working. There's different physics happening.

For statistical and modeling results, it's important to grid relative to the polar cap, so that open/closed field line boundary and the auroral oval.

The oral boundary detections from space-based imagers are needed to improve these efforts and existing boundary information may not be suitable for scientific use.

Slide 3
Here is an example of image SI12, SI13, and WIC that measured the Ly-α, oxygen, and LBH emission bands in the Northern Hemisphere.

Chisham et al. (2022) found the polar and equatorward auroral luminosity boundaries, or ALBs, using the Longdon et al. (2010 method. They used the Newell boundary DMSP boundaries to determine the offsets between the aurora luminosity boundary and the precipitation boundaries, and they fit the offset boundaries well with the convection reversal boundary, providing confidence in the ALBs.

They used an IDL code based developed by Longdon as a base, and they generalized it to allow for any number of Gaussian fits, added adaptive limits, and added a test for the magnetic local time consistency.

Slide 4

For this method, for the pyIntensityFeatures, first, it's recommended to clean the data based on the instrument team. Then it identifies a scan of intensity observations that contain the aurora oval in one hemisphere. Then you separate the scan into magnetic local time or MLT slices. For each slice, you fit the intensity, you identify the poleward and equatorward boundaries, and you calculate the boundary uncertainty, and then you evaluate the boundaries and select the best ones. Once you've done that, you can then scan for outliers and remove.

Slide 5

Here is an example again of the overall luminosity boundary in one image. You look for a scan - that's what this is -and you use pyIntensityFeatures, and it provides an array. You provide an array of time, latitude, longitude, and intensity data. And then you can specify a start time and a mask to clean the data. And then you could also provide the hemisphere. The default is north, and a minimum co-latitude.

Once you do that, it operates in geodetic coordinates, and it can process the auroral scan to get MLT slices. So it creates a grid in the magnetic coordinates. We see from this area here to here *(gestures between leftmost and center plots)* we have a different grid and then you get a mean and standard deviation of intensity for each magnetic latitude. This shows that as well.

And to determine the magnetic latitude limits of the slice by finding the highest and lowest latitudes with data at all MLT.

Slide 6

Here is going a little bit further into that. You take a band; here we have 00:15 MLT, and that is this region here going up. And you can see, based on I, a double peak structure in the auroral luminosity band, and that's what we're seeing here as well, this double peak. But we need our algorithm to find that.

What Angeline did was she identified the peaks by using Gaussian fits. So first you find the peak, and then you put a Gaussian to it and then within 2σ is the region of interest or region of influence. And then you go to the next one, you find after you subtract out that first Gaussian fit, you go find another one with the next peak. And if it's inside the region of influence, you would discard it, but if it's outside, it's another peak. For here we have two peaks and then a third one is inside the region of influence, so is not considered.

So this is to initialize the peaks and then work on future fitting.

Slide 7
Here is the future fitting. After you've identified the number of peaks, you can fit one, two, and three Gaussian peaks with a quadratic background. You can see the quadratic background pretty well here, and that is to account for day glow and noise.

Slide 8
Once we do that, we can find the poleward and equatorward boundaries for each fit, and that is done using those initial Gaussian fits. And the boundary is the location 2σ from the most poleward and equatorward peak, which you can see in the example here, for each one of the fits, we can see the different edges.

Slide 9
Next, you choose the best poleward and equatorward boundaries based on the fits, Pearson correlation coefficient and the boundary uncertainty. The boundary uncertainty is calculated using the standard deviation of the Gaussian, peaks, as well as the covariance of the fitting, and it's recommended for the user to account for the grid size in the final boundary uncertainties. And then the boundaries can be rejected based on the background intensity, boundary uncertainty, location relative to a latitude limit.

Slide 10
So this is some examples for one, two, and three fits. Here we have one fit, where we see that one peak is the best, we have some wider uncertainties, shown in gray here. Then when we get two fits, this one having the second one having a closer, smaller uncertainty, which, is aided in the fact that the two Gaussian fits are very similar, one and two peaks.

And then we have the third one where we have some larger uncertainties. But we are able to still fit pretty well into three Gaussian fits. Then there are times when you cannot detect one and based on I, we don't detect one and it doesn't detect it in the algorithm, which is great.

Slide 11
Once you do that, you can see here that we can take the boundary as shown by the X's and plus signs, and you can look for outliers. So this one here, the blue plus sign down at the bottom here is an outlier for the rest of these. So using five-hour MLT bins, you can remove the outliers and using the inner quartile range. And for full images such as a polar image, you can consider using the entire MLT range, which will give you a better idea of the outliers.

Slide 12
pyIntensityFeatures uses a class-based interface that combines all steps needed to perform ALB identification and output data into a netcdf file. The function access in submodules, allowing users to experiment with the best constraints for their data set. And this is a very nice flow chart and a bit about the code here.

Slide 13
In the future, ground-based observations like all sky imagers may also be worth incorporating. And other similar algorithms may be used for identifying other features using new classes

allowed for vetting identification algorithms. So you could potentially add traveling ionospheric disturbances, equatorial ionization anomaly or equatorial plasma bubbles.

Slide 14

pyIntensityFeatures is on GitHub. The link is shown here. The documentation is also available at this link, and the publication is shown here as well. So I highly recommend reading it. It's a great read. Thank you.

PyRayHF – Victoriya Forsythe et al., presented by Henry Valentine

Slide 1

Hello everybody. My name is Henry Valentine. I am a postdoc at the Naval Research Lab. And today I'm going to be presenting on PyRayHF, which is an open-source ray tracer for ionospheric data assimilation. This is Victoria Forsythe's brainchild. She is the main author of this project. I helped a little bit with development and validation, but I'm presenting this on her behalf as she's off to better things, as well as the on behalf of the other co-authors listed here.

Slide 2

Right, so quick overview. What is PyRayHF? PyRayHF is an open-source Python framework for ionospheric ray tracing with a particular emphasis on forward operators for data assimilation. So the kind of main functions, the main things that it provides are these forward operators for both vertical ray tracing. So like simulating an ionosonde measurement and creating vertical ionogram or doing two-dimensional ray tracing through a given ionosphere and maybe creating oblique ionograms out of that. We provide this vertical forward operator on the left, and then four distinct 2-dimensional ray tracers, each with varying levels of physical fidelity and computational performance, depending on the application that you need it for. Sometimes you need something really fast at the expense of accuracy. Sometimes you want something a little better, and so you can afford more time. There are obviously other ray tracers out there. PHaRLAP and various 3D magnetoionic Jones-Stevenson's codes.

PyRayHF is intended to fit a very specific niche of being fast, accessible, open source, and fitting into kind of NRL's broader Python framework of data assimilation tools. And so that's kind of the goal of this and why it exists.

Slide 3

So, talking a little bit about the vertical forward operator specifically. Like I said, this is used to simulate vertical ionosonde measurements or ionograms. The inputs are shown here on the left for a single location, you start with an electron density profile as well as magnetic field parameters. So the intensity as well as the inclination. And then you combine all those, and the user specifies whatever frequencies you want to run it at, and then it will produce an O-mode and an X-mode ionogram based on those inputs.

The vertical forward operator uses an adaptive gridding system, which is a very clever thing that Victoria thought up, where it increases the resolution near the reflection point of each ray, and that allows us to use the same number of grid points for each frequency and make sure we have really, really good resolution near the turning point for each ray. And because we have the same number of grid points, we can utilize NumPy matrix operations and do it very efficiently and solve and integrate the refractive index to find the virtual height for each frequency simultaneously. So you can get full ionogram traces in just a couple milliseconds, and it's really quick.

Slide 4

In addition to the vertical forward operator, we have multiple 2D ray tracing functions, as I mentioned. These come in different flavors, like I said, with different levels of physical fidelity.

We have these kind of Snell's law, very approximate versions that are on the left here, and these assume a horizontally isotropic ionosphere, no horizontal variation. And then you treat each kind of vertical altitude layer as an interface, and you compute Snell's law, and then that allows you to do a quick ray trace. We have it in both Cartesian and spherical coordinates. Spherical is a little bit slower, but then it accounts for the curvature of the ionosphere, so it's got just that little bit additional accuracy there.

And then in addition to the Snell's Law versions, we have these gradient methods, which solve the Eikonal equations, which is much more reminiscent of actual ray tracers. And these allow for horizontal gradients and are a lot more accurate, but they run more slowly. And then again, these are available for Cartesian and spherical coordinates as well.

Slide 5

This kind of begs the question, though, if we have all four of these different methods, are the really simple ones good enough for data similar applications assimilation and how do they compare to each other. We did this very quick validation test where we had a grid of transmitters and using all four of those different methods, we've traced rays of multiple elevation angles and frequencies and compared these to the output of a really highly accurate Jones-Stevenson 3D magnetoionic ray tracer that we have internal to NRL. So we use that as the ground truth, and then compared our tools against it, and then the results of that comparison are shown in the bottom and the right plots.

The bottom plot shows the kind of horizontal geographic distribution of the errors between those. And then the right plot shows the distribution of errors between them. And you'll note that it's pretty decent. As expected, the spherical gradient method, the slow but accurate one performs the best, but the kind of quick Snell's law ones actually perform pretty well as well, having a standard deviation errors within like 5 to 10%, which is fantastic for data assimilation. Granted, there's some limitations with this validation process because we used a very smooth IRI climatological ionosphere to test this on, and that may not be very representative of real life where you have more horizontal variation, and then the Snell's law one might break down a little more. So, there's some future work there to do some additional validation with more accurate ionosphere backgrounds.

Slide 6
I'm going to skip that one.

Slide 7
We presented the forward operators that PyRayHF offers. And so now let's talk about how these get incorporated into a data assimilation cycle. The goal here is to extract ionosphere profiles like FoF2, hmF2, B0, etc., from oblique ionograms or vertical anagrams using these forward operators that PyRayHF has. So the process looks a little bit like this, as represented in the flowchart: you start with a model state, so a model of your electron density and initial guess, and then you apply these forward operators which will give you a vertical ionogram or an oblique ionogram or something, and then you can compare those ionograms against actual data. And by taking the difference between the actual data and the simulated forward operator data, you look at the difference of those, and then that can help us inform how we need to adjust the model. So we adjust our parameters based on that, feed it back into the model state, and then iteratively apply that process until we refine it and get a model state that is consistent with real life observations.

That's the process of it. And the question is how doable is that? And for the 2D operators, it really depends on which ray tracer you're using. The Snell's law ones are super, super quick. And to assimilate a full oblique with 50 iterations, 20 elevation angles, and 20 frequencies, it would take less than a minute for the Snell's Law ones. But doing the same thing with the Cartesian gradient ones currently as it is in PyRayHF, those would take much, much, much longer. So for something like real-time data assimilation those aren't very useful. Of course, though, the vertical forward operator is much, much quicker. And so there's also the question of can we use the vertical forward operator, make some assumptions, and then apply it to an oblique case?

Slide 8
So enter stage left, the midpoint approximation: this midpoint approximation is a technique that has been used for decades to interpret oblique measurements in a kind of very simplified way. If you have a horizontally isotropic ionosphere and a flat ionosphere, you can do a little bit of geometry kung fu and then find a scaling factors that allow you to translate your oblique ionogram into a vertical ionogram at the midpoint of the transmitter and the receiver. So that works perfectly if everything's perfectly uniform. Of course, those criteria are hard to come by in the actual ionosphere, but you can still do the same transformation, and it gives you an approximate vertical ionogram that approximates what it would be for the oblique at the midpoint there.

Slide 9
And so using that particular transformation, we can plug that into our previously shown data assimilation loop. And PyRayHF provides tools to do just this.

Going through the same process we went through earlier, we start with an oblique ionogram on the left side. We do this midpoint transformation that I just mentioned, and it turns it into an equivalent vertical ionogram. And now that vertical ionogram is much, much easier to compare with our model using the vertical forward operator. That's really, really fast and efficient.

As shown here, you take the midpoint vertical and compare it to the actual data and then compare it to the forward operator from the model. And then you use that to inform the changes to your electron density, iterate that until you get the final result here. In this particular case, we did a test where you have this blue initial guess for the ionosphere, red is the true value, and then through this iterative process, we're able to converge on a pretty accurate solution here.

Slide 10

Then we did a little bit of validation and performance assessment of that specific tool. So all of that is done with just a single function in PyRayHF; it's just one function call. And we took a subset of the validation cases that we showed earlier and then did the same parameter minimization approach using the midpoint approximation to see how effective it is. And the answer is: it's pretty good.

The histograms down below are for the various different ionosphere parameters that we fit. The left is foF2, and in essentially all cases we were within about 5% of the true value for that fitting algorithm, and then the results for hmF2 and B0 are shown as well. The blue is the initial guess background for various different cases, and then the red is the results, so showing solid improvement in each of those.

Again, the process is a little limited because we're using a smooth climatological ionosphere, but we're doing more validation work from there.

Slide 11

In summary, PyRayHF is an open-source Python framework that provides ray tracing forward operators and tools to aid in the process of data assimilation. Source code is on GitHub. It's also on PyPI. So you can just do pip install PyRayHF, and it's really easy to get. Yeah, thank you.

asispectralinversion – Alexander Mule et al.

Slide 1

Hi, I'm Alex Mule. I'm here from Dartmouth, and I'm going to talk about the library asispectralinversion.

Slide 2

Brief introduction as to what it does: the routine is designed to invert calibrated all-sky imagery to electron precipitation spectral parameters. It was designed to work with the Poker digital all-sky camera at Poker Flat, Alaska, but it can and has been adapted to other cameras in Scandinavia, I believe, and one or two others, I believe.

It has a flexible choice of spectral shapes. The idea is that there's only so much information you can get from 3 color imagery. But since this routine is driven with the GLOW model, which I'll go into more detail about, you do have some freedom in the spectral assumptions that you make that you then choose to invert. In particular, we restrict ourselves to spectra that are parameterized by two parameters, one of which is just going to be the total energy flux, but the other of which is up to you.

So the classic example would be, if you assume that electron precipitation is Maxwellian, you would have one parameter, Q, the total energy flux, and then a second parameter E_0, the characteristic energy. But if you decided some other spectrum maybe fit your data better, you could parameterize it a different way. And in principle, it's completely adaptable.

It's designed to work with three color imagery. So that's 558 nm is the green line, 428 nm is the blue line, and then it can either work with 630 nm, the red line, or 8446 the near infrared. And they work in very similar ways.

It can also work without the blue line, so without 428 nm, though a little bit less effectively. And I will get into that.

One more thing before I go on: this sort of thing has been done many, many times, as some of you are probably familiar. This particular implementation is kind of from the heritage of the UAF group and was shared to me from Don Hampton, and we've made some changes to make it kind of more user-friendly, added some additional features like this inversion without the blue line. But yeah, that's kind of the heritage of this routine.

Slide 3

I have a little flowchart on how this works. Ultimately, it all relies on the GLOW model. The way GLOW works is GLOW is a physics-based model that will take a bunch of background drivers. So, for example, F10.7, Ap, a location, a time. And then it takes a spectrum of electron precipitation, which you can tell it, give it whatever you want. And then it spits out volumetric emission rates for spectral lines. The ones we care about in this case are the ones I mentioned. But you can specify whatever spectrum you want, or you could do what we do, which is specify a whole swath of spectra.

So let's assume a spectral shape, again, kind of Maxwellian is the classical example. You could imagine a swath of total energy fluxes and a swath of characteristic energies kind of filling out a 2D space. You could run glow with every single combination of a choice of Q and a choice of E_0, and then you can sort of tabulate these volumetric emission rates. And that's exactly what we do, and we generate what we call a lookup table for short, which is just this kind of tabulated result of a bunch of GLOW runs that can all be run parallel to each other. And it's not super computationally intensive.

Slide 4

Where does asispectralinversion come in? Well, once these lookup tables are generated, asispectralinversion inverts them. So GLOW does all this physics driven work to go from spectral parameters to emission lines, and then asispectralinversion uses the lookup tables to work backwards. Given some all-sky imagery, given some calibrated auroral imagery, it works backward to give you spectral parameters.

Slide 5

A rough outline of how it works: again, those of you who are familiar with imagery inversion have probably seen something like this many times. But long story short, the blue line roughly -

again, this is in the case of Maxwellian spectra but its relatively universal - mostly depends on total energy flux. Whereas the red to green ratio or green to red ratio mostly depends on characteristic energy.

Of course, it's not quite that simple, and we just kind of use an iterative step, iterative process that very quickly converges to actually find the value of spectral parameters that best corresponds to the intensities. But in principle, that's mostly what we're doing. And that approximation makes up the initial guess.

Slide 6

In terms of kind of additional features that we've added, one of them that we found to be very useful is this wavelet transform-based denoising; for time reasons, I'm not going to talk about how it works. In much detail, but feel free to ask me about it. But the idea is that we can denoise images in a way that doesn't suppress high frequency features the way a standard kind of low-pass filter-based denoising works. And in this picture, you can see so this is the blue line image, which tends to be very, very noisy.

There's some structure here that we can kind of verify is real in the green imagery that is, not only maintained, but in fact, is easier to see with the naked eye than it was in the noisy version.

Slide 7

A little comment on additional spectral shapes: in our lab, we've made use of accelerated Maxwellians, which are just Maxwellians with a shift. The idea is when you're generating your swath of spectra, instead of just varying a whole bunch Maxwellians, from really cold to really hot, you pick a source temperature and let them fall through an inverted “V” of a potential of your choice. And that cuts off the long tail of high energy particles associated with the hot Maxwellian that we believe sometimes leads to unphysically high ionization rates very deep into the ionosphere.

That's one reason you might want to use a non-Maxwellian spectrum.

Slide 8

Can you get electron density and conductances out of this? Sort of.

GLOW relies on IRI, which you can only use in certain conditions and at certain altitudes. So we don't really advertise electron density and conductances as data products. You can get them.

Just as another example of kind of how this routine can be used, we often use it as a topside driver for the GEMINI ionospheric model. And in that way, GEMINI has its own way of generating ionospheric densities that does not rely on IRI, where it kind of preheats an ionosphere, you give it background driving conditions, and it'll find an equilibrium state over a 24-hour run and uses that. And that kind of gets around some of the issues with IRI. But in principle, GLOW is very flexible.

So if you have your own reference ionosphere that you like better, you could feed it in, don't ask me how, but it can be done. But yeah, if anyone else has suggestions or ideas, feel free to contribute them.

Slide 9

Here's another feature that we added in, which is inversion just from the green and red lines, because in principle, all the information needed for this kind of two-parameter inversion is contained in the red and green lines.

Now, it's sketchier than using the blue line, and I will explain why. But I guess just to summarize what I said on the slide, the green line moves around based on precipitation, precipitating energy. It can move up or down, and the red line is very broad, and that means that any sort of inversion that's really relying on the relative values of those two is subject to geometric error.

Slide 10

And if the blue line is available using this method, you can kind of assess consistency of the data with the GLOW simulation. So just here's a little picture of that. You can see if you're looking in any direction other than straight up a magnetic field line, you're going to have some geometric error.

Other limitations of the regime of the routine: of course, you have geometric error. You also have the requirement of this kind of properly calibrated three-color imagery, limitations with IRI, etc.

Slide 11

As for the user interface, we currently have two versions that are going to be merged together very soon. There's a tutorial style Jupyter notebook that calls some functions in a more scriptable version. They call the same functions, but yeah, maybe not a best practice, but we're getting there. Soon to be merged together.

Slide 12

For the time being, I have QR codes for both of them.

As a summary, ASI spectral inversion allows two parameter inversion for an arbitrary set of spectral shapes of precipitating electron spectra from calibrated 3-color auroral imagery data.

It can be useful at supplying mesoscale precipitation information to auroral models or as context for experiments, data assimilation, etc. And yeah, that's all. Thank you.

pyValEIA – Alanah Cardenas-O’Toole et al.

Slide 1

Hello, I'm back. Once again, my name is Alana Cardenas-O'Toole. I'm a student at the University of Michigan, and last summer I was an intern at the Naval Research Laboratory through the

NREIP program, and I worked with Angeline Burrell and Dustin Hickey on pyValEIA, which I'll talk about now.

Slide 2

pyValEIA is a Python package that we developed to characterize the 2D shape of the low latitude ionosphere for the plasma density of the ionosphere for model-to-model comparison and model observation comparison.

This image is an example from Madrigal of the density, what the density structure regularly looks like in the ionosphere at low latitudes. We see a double peak, a double band. We see the yellow higher density, and between the two yellow stripes is a lower density region, and that's often called the equatorial ionization anomaly, or EIA. And this is a result of the $\mathbf{E}\times\mathbf{B}$ drift and plasma diffusion acting together with hemispheric asymmetries that often result from neutral winds

The goal of pyValEIA is to diagnose the physics-based weaknesses of a model using the presence or absence of the EIA. And in this way, we can understand the ionosphere a bit better and fix our models where we need to.

Slide 3

For the study, we used a geomagnetically active period - January 2014 - and a geomagnetically quiet period - April 2020. We used Swarm A, B, and C for the in situ plasma density measurements, the Next-Generation Ionosphere Model for Operations, or NIMO, was one of the models that we're comparing the data to, and this is a global data assimilated model. Then we have a PyIRI, which is a pure Python implementation of IRI, which is what we compared NIMO to. And then we also used the TEC measurements from Madigral.

Slide 4

In order to determine the shape of the ionosphere, we had to give them names. So we categorized it by four categories based on if there was an EIA or not.

We have Trough as the first one, which is labeled there. So we have the lower density region at the magnetic equator, but we don't have that double peak structure that we see with the EIA.

Next, we have Flat. So flat can be three different orientations: it could be north, south, or symmetric. Here's an example of flat north where we have an increase in density from the southern hemisphere to the northern hemisphere with no really defined peaks.

Next, we have Peak, which is what it sounds like. We have an increase in density right around the magnetic equator and the orientation is based on whether on where the peak is. In this case, it's symmetric because it's right on 0 degrees

Next, we have our EIA categories. We divided these into three different classes because you have more peaks, you have more ways that things can look generally. We have Classic first; we saw the image in the first slide. This is what that might look like with a latitudinal slice or a

longitudinal slice going through. We have an increase, decrease, increase what we'd expect. And this is the orientation is based off of which peak is higher.

Next, we have EIA Saddle. It's similar, we have that double peak structure, but we don't have that same trough in the middle, that minimum.

And then finally, we have a new designation that we found as we were going through the swarm data, that flies through around 462 km, so a bit above the regular F region, where you generally the F region peak. It is a triple or double peak structure. In this case, it is a triple peak structure, and its orientation is based off of which side is higher. So it looks like a little sheet ghost. If you put eyes on it, it really does. Whichever arm is higher is the orientation.

We also have a Ghost Peak, which is where you just have one arm and a peak at the center. So it's important to designate there's a peak at the center and then the two sides or one side.

And this is probably a mix between peak and EIA, so maybe a transition between the two is what we are thinking is going on here. So those are our designations.

Slide 5

And pyValEIA, we start by processing the density. This can be TEC, plasma density, or electron density. And this is done by interpolating, detrending, and filtering. And then from there, we have this nice flow chart.

So we go down, we scale the density. And that scaling equation is shown here. And this allows us to put the TEC and electron density in the same system.

Additionally, we calculate the gradient of the density. And you find the zero lats, and the zero lats are where that gradient equals 0. So we have those transition points. Then you count the number of zero lats and you determine if they are maxima or minima. And from there, you find the number of crests.

Ideally for zero crests, we would have a flat or trough. For one crest, ideally, we'd have peak, and for two or more crests, it would be EIA. But sometimes one crest isn't necessarily as important as the other ones. So, two or more crests can be any of the categories, and one crest could also be flat or trough, and hopefully this will get a bit more clear as I go through.

Slide 6

Here's an example of PyValEIA in action. In this case, there are three zero lats, and we connect them to what we call an anchor point at the edges of the slice, the latitudinal slice. And you find the slopes between them, shown here. The density is scaled so it's on the same scale as the magnetic latitude, and they have the same unit, so those slopes are unitless.

we then determine if the zero lats are primary or secondary maxima or minima. So this one is a primary maxima because we have a slope, an increasing slope above 0.5 and a decreasing slope. The absolute value above 0.5.

And then our secondary is where we have one of the slopes is less than 0.5, and the other one's negative, and the one in between is our minima.

So, these two crests are evaluated based on their distance, width, and slopes to the minima, resulting in a EIA Saddle South designation, which you can see at the double peak structure, but less of a density depletion in between.

Slide 7

Here's another example with Swarm but also with NIMO and PyRI plotted as well. So there are bubbles shown in the swarm density data, which shows why it's important to detrend so that we can get a smooth trend over top rather than seeing the depletions as another, different trough for different peaks.

NIMO is the data that we chose corresponds directly to the Swarm pass, including the longitude, altitude, and the closest time. And that's plotted here, which you can see that the magnitude matches pretty well and actually the type matches well as well. They both say EIA South, which is perfect.

PyRI also shows an EIA category, but it is slightly lower in density, and it has a Saddle Peak instead of the Classic shape.

Slide 8

We can also do this with the total electron content.

Madrigal TEC measurements are sparse in certain regions, so we chose two different sectors: we have the American sector, and we have the Asian sector, which is between the solid lines.
The TEC data was binned for every 15 minutes by 5 degrees longitude, which is shown here.
And then we linearly interpolate it to trend it and smooth it, which is the yellow line over top.
And we also have NIMO TEC here, which is limited by the same parameters as the observations.

In this case, we have both of them showing an EIA orientation. One of them EIA category, one of them as a Saddle and one of them as Classic, and then for this one here both say Peak, but the different orientations. So we can see how Nimo is working against the observations.

Slide 9

PyValEIA provides an apples-to-apples comparison between models. And this is just one example of how you can do that. This is something you could create with the code; here it is simplified, but you can go as deep as you want.

This is just an observation model table where if we have an EIA, if both agree, it would be either a hit or a correct negative, and if they disagree, it would be their miss or a false positive, and this is between the observations and the model. So you can see how the observations compare to one model, and how they compare to another to find the weaknesses and strengths of your models.

Slide 10

In conclusion, PyValEIA is designed to characterize the low-latitude ionosphere state based on the presence or absence of the EIA. It provides the EIA state, in addition to the crest latitudes and densities, and it's useful for statistical studies, diagnosing weaknesses in models, and performing model-to model-comparison.

We demonstrated the use in total electron content and electron density. And we can also use it to compare the two models, showing the weaknesses and strengths in both.

It is in active development on GitHub, and the link is there. Upcoming releases will include bug fixes and package enhancements, and we encourage you to test it and provide feedback. Thank you.

PlasmaCalcs – Samuel Evans et al., presented by Save Koontaweepunya

Slide 1

Hello, everyone. My name Save Koontaweepunya. I'm presenting this talk on behalf of Sam Evans, who couldn't be with us today, but I think he's on Zoom. So he can answer questions. I can answer questions as well.

Slide 2

PlasmaCalcs is built to solve a specific problem, and the problem is plasma physics is complicated. There are lots of data sources we can take data from, including simulations, observations, blah, blah, blah. There are also lots of quantities to look at, lots of operations. And the point of this Python package is to make this easier.

Slide 3

And how do we do that? One thing is we provide a consistent interface across all data sources. We can take data from multiple simulations as well as observations, including GAMERA, REMIX, Bifrost, EPPIC, and these are like mostly simulations, but there's no reason for observational data to not be included here.

There are some already set up calculators for these specific models, so after you have the model hooked up with the package, you can use some operations on them. And, for example, you want to look at the number density of some species, you can just type “n”, and “n” is the number density, and that will communicate, that will be the same function across all of these simulations or observations.

And you can even do “log10_n” like is seen here. you can get outputs like these as well.

Slide 4

For plasma cells, the dimensions and coordinates are attached directly to the data. And we make use of this other package called Xarray to store most of these arrays.

If we load this “log10_n”, we'll get this Xarray with multiple dimensions, including the time steps, well, it depends on whichever simulation, we are using.

So here, for REMIX, you have snap, you have hemisphere, you have theta, radius, and this can be easily plotted by just calling “pc.subplots()” and it would make plots for you. You can make movies as well. And this is the movie.

There are default settings for these plots, so that if you just want to look at data quickly, you can just maybe use Jupyter Notebook or something and yeah, you can just make these plots.

Slide 5

There’s also slicing for our data array. You can slice things in here with arguments, you can add arguments after when we are loading these. For example, if we want to look at the eighth time step, you can say “snap=8”. If you don't want to look at the whole array, you can like do some slicing of these dimensions. For example, theta 40 to whatever the maximum is.

If you look at the plot on the right, you can see that these got sliced out. Like the radius, you can even slice in fractional indexing. And this is scary to me. For me, this slice object is its own thing, but apparently Sam did something fancy to it

Yeah, if you say slice 0.2 to 0.7, that means it's gonna cut out the 20% out, and then it goes up to 70% of the radius.

An interesting thing is that this slicing happens before loading. So you don't have to load the whole array and then slice things, PlasmaCalcs will slice it for you. Just load the sliced array.

Slide 6

It is also easy to combine lots of operations and customize behavior. For example, these are some like regular expression magic. Like, if you want to know what is “(n/mean_n)-1”, you can just type in that line and you get that value instead of “first I need to do n equals this and then calculate the mean of n”.

But here you can just type it in as a string of n divided by mean of n minus one.

Same thing for log and absolute. You can even do Fourier transform for this.

There's also these plots that you can make with default settings, or you can also specify what settings you need.

Slide 7

There's also multiple dimensions; **U** is a vector. You can also have an Xarray of U_x or U_y, U_Z. And you can plot these easily as well with the default settings for the plots.

Here you have the movie on the right that has the U_x on the top, U_y on the bottom, and also have multiple species. So electrons, protons, carbon there, in each column. And this is all easily configurable from for the user as well by putting it as in the arguments in this subplots function.

Slide 8
And the underlying grid doesn't need to be rectilinear.

Here we have simulation data from GAMERA and the grid there is like an earth-centric grid, where this is the earth, and you go out radially out of the earth, and so it's not rectilinear. Yeah, it's like an egg-shaped out, and you can plot them with PlasmaCalcs as well.

Slide 9
And the internal implementation looks like physical equation, so it's easy to read and has custom health systems with you can do “pc.help” of anything and there will be doc strings come out as well as some custom helping lines.

Slide 10
It is most helpful if you spend more time writing code than running code because it is great at quickly exploring and visualizing data and comparing across different data sources. And it's built to grow and support many more kinds of inputs in the future.

Slide 11
You can find PlasmaCalcs on PyPI. You can do “pip install PlasmaCalcs” and it will install. It's on GitLab as well. This is the QR code for GitLab, and you can reach out to us or me after the talk, and you can join the PlasmaCalcs Slack as well. Ask for link. Okay, thank you for listening.

Discussion Notes:

- Riding between different languages
- Emerging languages and tools
- Institutional barriers to FOSS
- Citing code in papers
- What to do as a reviewer
- Dependency management
- 3D Python plotting
- Do you need a package or just better docs?

How do you appropriately cite packages?
- Do you also cite dependencies?
- Citing actual software (in addition to paper) is usually required/desired.
- Also need to cite data, but this can be modified with conversations with editors when needed.
- For python modules specifically, if you are ascribing to the "top-level citations only", pipdeptree is a handy tool to outline your dependencies

Dependency management, people should do it!
- You should tell people what dependencies you need
- pyproject.toml is the current standard for managing dependencies

- what about optional functionality? There are standards for this and is common for hard-to-install packages

Do you need a package or better documentation
- Self-describing data is the best
- Meta-data is essential, but documentation can sometimes be more accessible