pyCRTM: A Python Interface for the Community Radiative Transfer Model

Bryan M. Karpowicz[a,b,c,1], Patrick G. Stegmann[d], Benjamin T. Johnson[d], Hui W. Christophersen[e], Edward J. Hyer[e], Andrew Lambert[e,f] and Eric Simon[e,g]

[a]*Global Modeling and Assimilation Office, NASA Goddard Space Flight Center, Greenbelt, MD, USA*

[b]*Goddard Earth Sciences Technology and Research II, Greenbelt, MD, USA*

[c]*University of Maryland Baltimore County, Baltimore, MD, USA*

[d]*Joint Center for Satellite Data Assimilation, NOAA Center for Weather and Climate Prediction, College Park, MD, USA*

[e]*U.S. Naval Research Laboratory, Monterey, CA, USA*

[f]*General Dynamics Information Technology, Monterey, CA, USA*

[g]*Science Applications International Corporation, Inc., Monterey, CA, USA*

Abstract

The Community Radiative Transfer Model (CRTM) is a powerful and versatile scalar radiative transfer model for satellite data assimilation and remote sensing applications. It is implemented as an object-oriented Fortran library, enabling flexible code development and optimal runtime performance on clusters. The downsides of the Fortran interface are a steep learning curve for students and the reduced productivity of users that is typical for static compiled languages, in contrast to dynamic interpreted languages like Python. pyCRTM is a new software framework that directly interfaces the CRTM Fortran data structures and procedures in Python, leveraging both the simplicity and ease of use of Python syntax as well as the flexibility arising from the vast contemporary Python ecosystem. The goal of pyCRTM is to lower the barrier of entry for university students to learn and use the CRTM and to boost the productivity of researchers seeking to create new methods in radiative transfer and data assimilation, or seeking to apply the CRTM to study atmospheric phenomena without having to go through the pre-existing complexity of the CRTM Fortran interface.

Keywords: radiative transfer; CRTM; Python; data assimilation; remote sensing;

1. Introduction

1.1. *Background*

The Community Radiative Transfer Model (CRTM) [1-6] has proven to be an invaluable fast scalar radiative transfer model which enables direct assimilation of radiance measurements for numerical weather prediction, calibration and validation studies of passive microwave and infrared satellite instruments, and planning for new instruments and platforms via Observation System Simulation Experiments (OSSEs). In the past, the CRTM was primarily utilized as a part of a larger framework typically a data assimilation system like the NASA Global Earth Observing System-Atmospheric Data Assimilation System (GEOS-ADAS), NOAA's Global Data Assimilation System (GDAS), or the US Naval Research Laboratory's (NRL) NRL Atmospheric Variational Data Assimilation System-Accelerated Representer (NAVDAS-AR). The CRTM has also been used in satellite retrievals such as the Microwave Integrated Retrieval System (MiRS), in the Global

[1] Corresponding author.
E-mail address: bryan.m.karpowicz@nasa.gov

Modeling and Assimilation Office's (GMAO) GMAO-OSSE system, and the operational calibration and validation (CalVal) at the NOAA NESDIS STAR line office. The CRTM is implemented in Fortran for reasons of parallel performance and optimization. While the Fortran interface is versatile in the hands of Fortran experts, there is a steep learning curve for students and new researchers unfamiliar with Fortran. There are also research applications with a rapid development cycle, or where a researcher just wants a quick plot of brightness temperature given a meteorological analysis or forecast where Fortran is not ideally suited. This particular performance metric is increasingly referred to as *time-to-plot* in scientific programming. A Python interface to CRTM known as pyCRTM was developed with students and researchers in mind who would prefer working with Python *in lieu* of Fortran, for applications where rapid prototyping is required, and other applications that could harness the power of the Python ecosystem.

1.2. *Previous Work on Python Interfaces for Radiative Transfer*

It should be noted that pyCRTM is not the first and only interface to a fast radiative transfer model as the Radiative Transfer Model for TOVS (RTTOV) has a Python interface known as pyRTTOV [7]. While pyCRTM is not the first interface to a fast radiative transfer model, it does allow researchers to simulate observations as the GEOS-ADAS, GDAS, and NAVDAS-AR simulate them versus how they would be simulated in the European Center for Medium-Range Weather Forecasts (ECMWF), UK Met Office, or Meteo France systems. Having a Python interface to both RTTOV and the CRTM lowers the barrier for researchers and students to perform intercomparison studies between the two systems.

1.3. *Purpose of this study*

The purpose of this study is to present the basic design, and implementation of pyCRTM. Along with presenting the interface itself, some examples that demonstrate the ease of use, and powerful applications that can be developed with pyCRTM are presented.

1.4. *Manuscript Overview*

In the following, the manuscript is divided into 5 remaining sections. Section 2 provides the design goals of pyCRTM. Section 3 briefly describes the implementation, and access to data structures. Section 4 describes connections to other frameworks that when combined with pyCRTM minimize the barriers to novice users, and novel applications. Section 5 discusses some of the powerful results that may be obtained using pyCRTM. Finally, in Section 6 concluding remarks are made.

2. Design Goals

The overarching design goal with pyCRTM is to maintain access to the many features available within the underlying Fortran model, while keeping the interface simple enough for users with only a basic knowledge of radiative transfer calculations and Python. Additionally, given there is an existing Python module for RTTOV, an effort was made to keep variable names and methods consistent between pyCRTM and pyRTTOV. The design essentially wraps two models available in CRTM: the Forward Model (accessed in pyCRTM as runDirect) which can be used to compute brightness temperature or radiance, along with transmission, and the CRTM K-Matrix Model (accessed in pyCRTM as runK) which performs the same computations along with Jacobians of atmospheric temperature, concentration of atmospheric

constituents, surface temperature, surface emissivity, surface reflectance, wind speed, and wind direction. Finally, the OpenMP capability of the underlying Fortran models is preserved allowing for parallel processing of many atmospheric profiles.

3. Conceptual Implementation

The CRTM is more fully described in other publications [1-6] and this paper describes only the interfaces and configurable options of the model. This section provides an overview of the implementation of the pyCRTM interface and the underlying concepts. The main class by which a Python user interacts with pyCRTM is through the pyCRTM class and profiles data structure. A UML diagram of the initialized pyCRTM class is shown in Fig. 1.

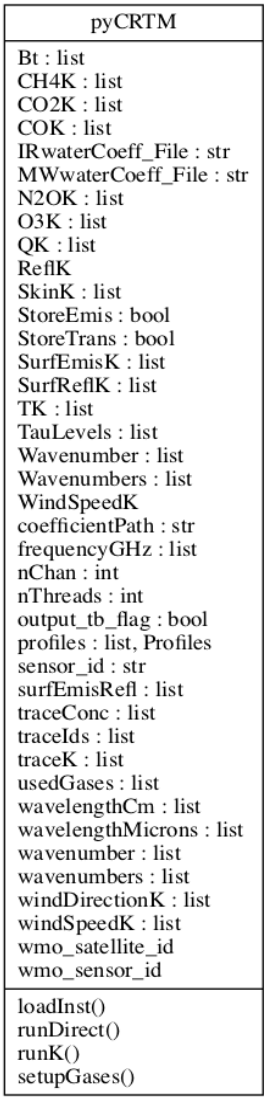

Figure 1: UML diagram of the pyCRTM core class.

Most of the variables in the pyCRTM class are output variables which are populated by the loadInst, runDirect, or runK methods. Once the pyCRTM class is initialized, the user must provide the profiles data structure, and the sensor_id. The sensor_id is a string which defines the simulated sensor which is the sensor name followed by an "_", then the observation satellite platform (e.g., 'atms_n20' would indicate the Advanced Technology Microwave Sounder on the NOAA 20 satellite). Next, the user must run the loadInst method in the pyCRTM class to populate metadata such as channel frequencies or wavelengths, along with setting the wmo_sensor_id and wmo_satellite_id for the underlying CRTM. The user can then choose to run either the runDirect function, or runK function.

If the user calls the runDirect function shown in Fig. 2, it will set required inputs to the CRTM based on the user's input profiles, run the Forward Model in the CRTM, populate brightness temperature or radiance (variable Bt in the pyCRTM class), along with the option to return transmission (TauLevels), surface emissivity and surface reflectance (surfEmisRefl). The dimensions of the brightness temperature variable (Bt) are profile number, followed by channel number. For the transmission variable (TauLevels) the dimensions are profile number, followed by channel number, along with layer number indexed by increasing pressure (top of atmosphere is index 0, bottom of atmosphere is the last index). The user can then take the variables from the pyCRTM class and use them in further numpy calculations, plot them using the aid of matplotlib, or include them in user defined Python applications.

Alternatively, the user may call the runK function shown in Fig. 3. It will set required inputs to the CRTM based on the user's input profiles, run the K-Matrix model in the CRTM, populate brightness temperature or radiance (in Bt), along with the option to return transmission (TauLevels), surface emissivity and surface reflectance (surfEmisRefl) as with the runDirect function. In addition runK will populate the Jacobians (any variable in Fig. 1 ending with a capital "K"). If the user does not provide a gas concentration in the *profiles* structure, it will not populate the associated trace gas Jacobian (e.g., N2OK will not be populated if the user does not include N_2O). The user must provide at minimum water vapor and ozone to the CRTM, therefore their associated Jacobians (QK, and O3K) will alway be populated. The dimensions of all Jacobian variables with a vertical dimension (e.g., QK,O3K,TK) have array dimensions of profile number, channel number, and layer number (increasing pressure). The skinK Jacobian array has dimensions of profile number, channel number, and surface index (0=Land Temperature, 1=Water Temperature, 2=Ice Temperature, 3=Snow Temperature). Jacobians for all other variables which do not have a vertical dimension (e.g. emisK, reflK, windSpeedK, windDirectionK) have array dimensions of profile number, and channel number. Again, the user may take the variables populated into further NumPy calculations, plot them using the aid of matplotlib, or include them into any desired Python application.

Most of the pathways triggered within runDirect or runK (see Figs. 2 and 3) are defined by the user provided profile data structure. The user initializes a profile data structure using the profilesCreate function providing the number of profiles, and number of layers in the profiles, along with optionally providing the number of aerosol types, number of cloud types, and a list of additional trace gases. The fields the user must specify in the profiles data structure are shown in Fig. 4. Required fields include layer pressures, layer temperatures, layer specific humidity, layer ozone concentration, level/layer interface pressure, angles (sensor zenith, sensor azimuth, solar zenith, solar azimuth, sensor scan angle), surface types, surface fractions, surface temperatures, wind speed at 10 meters, and wind direction at 10 meters. Additionally, the user may set desired inputs for aerosols and clouds present in the *profiles* data structure.

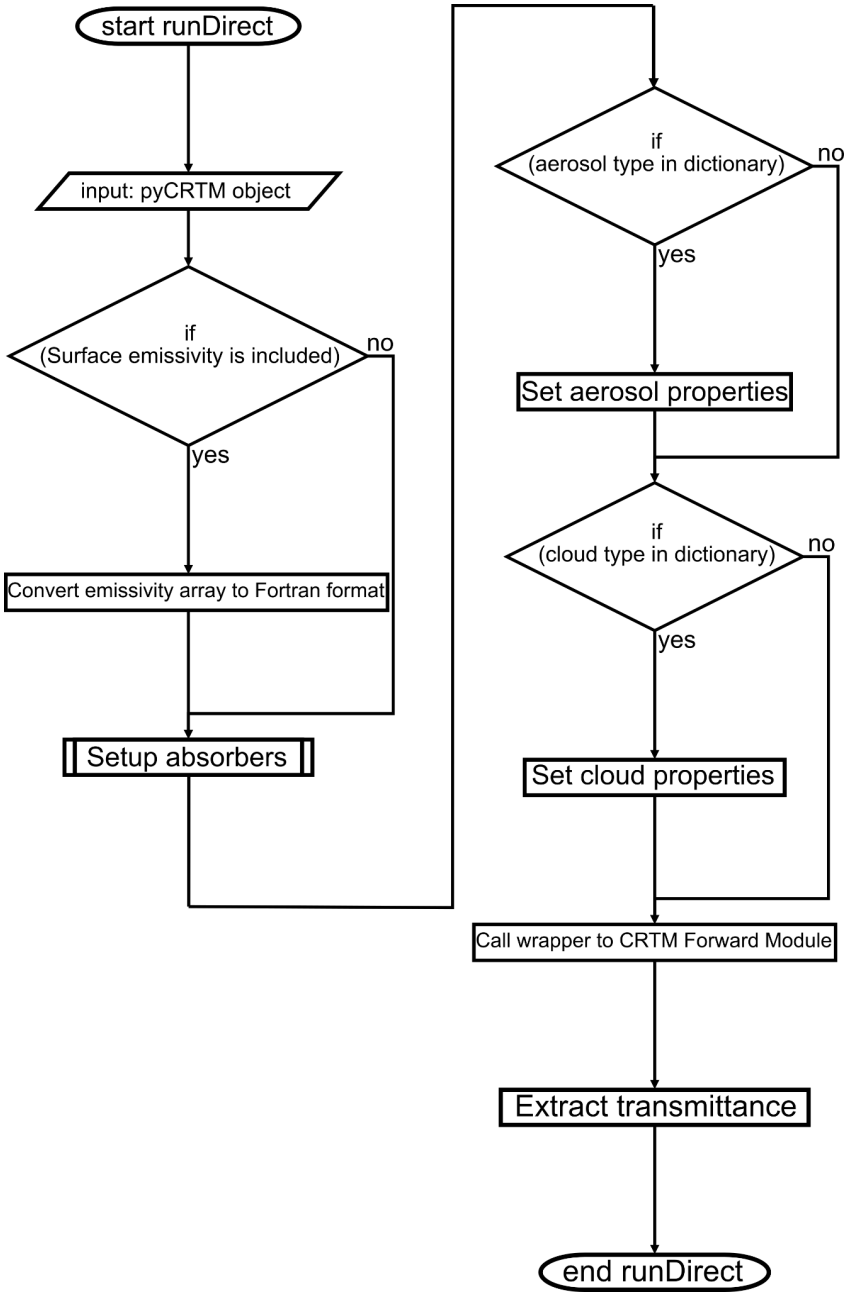

Figure 2: Flowchart of the forward operator method of the pyCRTM class.

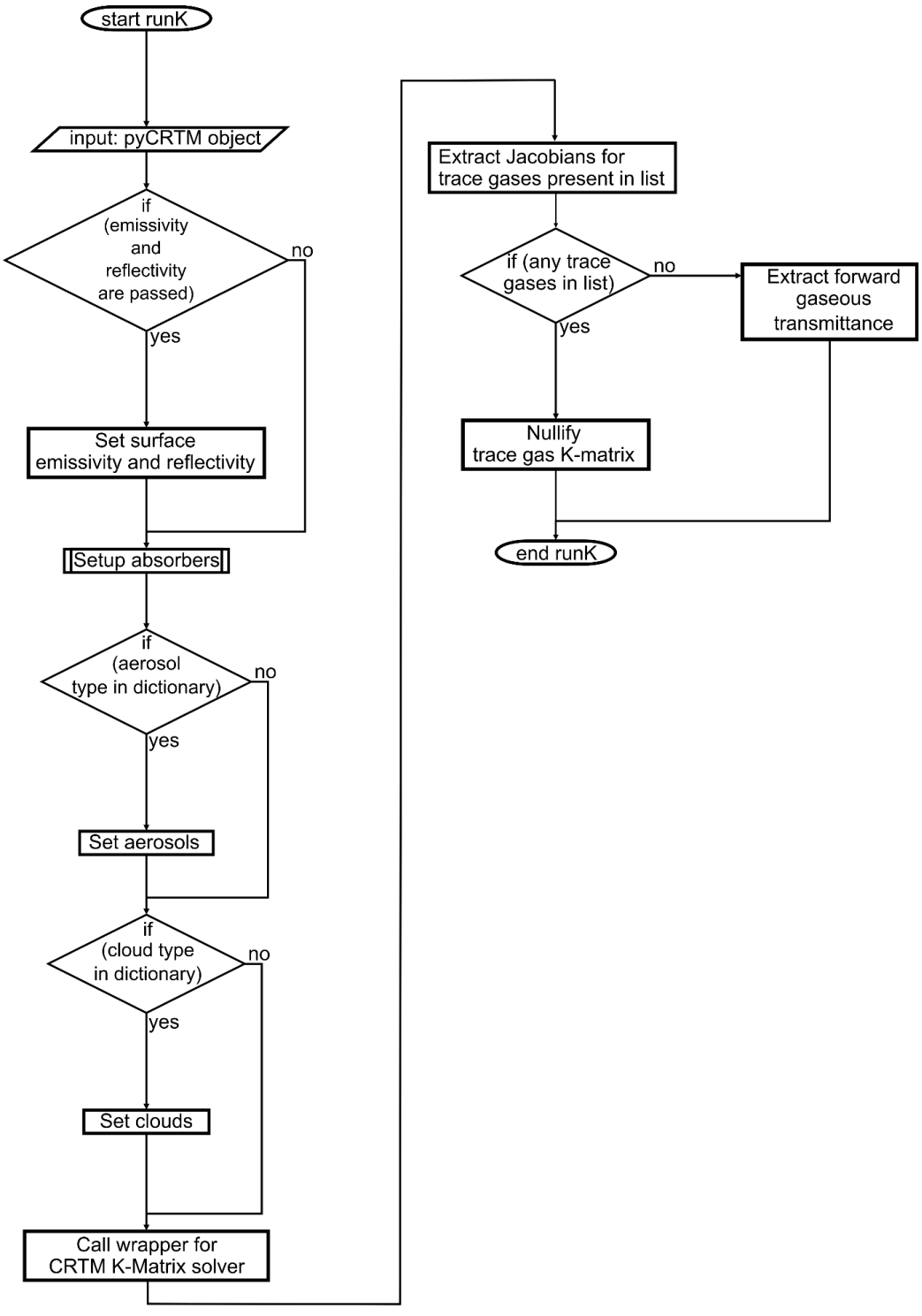

Figure 3: Flowchart of the Jacobian (K-Matrix) method of the pyCRTM class .

Profiles
+ P(n_profiles,n_layers) [required] : float array, layer pressure + T(n_profiles,n_layers) [required] : float array, layer temperature + O3(n_profiles,n_layers) [required] : float array, layer ozone ppmv + Pi(n_profiles,n_levels) [required] : float array, level/interface pressure + Angles(n_profiles,5) [required] : float array, zenith, azimuth, solar azimuth, solar zenith, scan angle + SurfeaceTypes(n_profiles,6) [required] : int array, land, soil, veg, water, snow, ice + SurfeaceFractions(n_profiles,6) [required] : float array, land, soil, veg, water, snow, ice + SurfeaceTemperatures(n_profiles,6) [required] : float array, land, soil, veg, water, snow, ice + windSpeed10m(n_profiles) [required] : float array, wind speed + windDirection10m(n_profiles) [required] : float array, wind direction + Salinity(n_profiles) : float array, ocean salinity PSU + surfaceGeom(n_profiles,3) : float array, latitude, longitude, elevation + DateTimes(n_profiles,3) : int array, year, month, day + aerosols(2,n_profiles, n_layers,n_aerosol_types) : float array, concentration, effective radius + aerosolType(n_profiles,n_aerosol_types) int array, aerosol type + clouds(2,n_profiles, n_layers,n_cloud_types) : float array, concentration, effective radius + cloudsType(n_profiles,n_cloud_types) int array, cloud type + cloudsFraction(n_profiles,n_layers) float array, cloud fraction + climatology(n_profiles) int array, climatology (1=tropical,2=Midlat summer, 3=Midlat witer, 4=subarctic summer, 5=subarctic winter, 6=US standard 1976)

Figure 4: UML of the profiles data structure

4. Connections to Other Frameworks

4.1. Interaction with other Python frameworks

A major advantage of encapsulating the CRTM into pyCRTM as a convenient Python interface is the direct interaction with the vast selection of other scientific Python packages. For radiative transfer, remote sensing, and data assimilation these include in particular:

- NumPy for numerics and linear algebra [8].

- Matplotlib for plotting and data visualization [9].

- MetPy for meteorological analysis [10].

- Cartopy for geographic maps [11].

Often the goal of a computational study is the analysis of some numerical data that is the result of a CRTM calculation. However, running the CRTM as a standalone Fortran program requires saving the calculation results in an intermediate format as an additional step. The output data are often further analyzed using NumPy, or visualized using Matplotlib. Using pyCRTM makes the aforementioned intermediate step unnecessary and the data can be processed directly in a single Python program. A more elaborate example of this functionality is given in subsection 5.1 on sensor simulation in the microwave region.

4.2. Jupyter Notebooks

Computational notebook programming was first introduced in the proprietary Mathematica software [12] as a form of literate programming [13] suitable for scientific research. A popular open-source alternative is the Jupyter framework [14] which, as its name implies, supports Python as one of its languages. pyCRTM, as a Python module, can easily be called from inside Jupyter notebooks. A screenshot of running pyCRTM within a Jupyter notebook is shown in Fig. 5.

Run the CRTM in Forward and Jacobian Mode

Run the Forward solver of the CRTM for the 4 test cases:

```
crtmOb.runDirect()
```

Extract instrument brightness temperatures from solution object:

```
forwardTb = crtmOb.Bt
```

Extract surface emissivity from solution object:

```
forwardEmissivity = crtmOb.surfEmisRefl[0,:]
```

Reset the surface emissivity for the K-Matrix run:

```
crtmOb.surfEmisRefl = []
```

Run the K-Matrix solver:

```
crtmOb.runK()
```

Extract the brightness temperature solution and the surface emissivity Jacobian:

```
kTb = crtmOb.Bt
kEmissivity = crtmOb.surfEmisRefl[0,:]
```

Figure 5: Screenshot of running pyCRTM inside of a Jupyter notebook..

This makes it particularly convenient for the type of exploratory analysis that Jupyter notebooks are well suited for. University courses may also benefit from using pyCRTM within Jupyter notebooks as a tool to teach radiative transfer, remote sensing, and data assimilation, further emphasizing the practical value of pyCRTM. pyCRTM Jupyter notebooks provide a dynamic environment with a low barrier-of-entry and can be used for coursework assignments that can automatically be graded using the nbgrader tool [15].

4.3. GeoIPS

The Geolocated Information Processing System (GeoIPS) [16] version 2.0 is a Python-based, efficient generalized processing system that will bridge the gap between satellite remote sensing and data assimilation and NWP modeling by providing modern modular scripts to facilitate rapid incorporation of satellite, model and other environmental data. GeoIPS is intended to replace Terascan software currently in operation at Fleet Numerical Meteorology and Oceanography Center (FNMOC). Expanding the capability of GeoIPS with augmentations of radiative transfer calculations and various data assimilation applications will increase its synergy of satellite processing/imaging, model output and other environmental data. NRL has incorporated CRTM into GeoIPS using pyCRTM to facilitate rapid incorporation of satellite, model and other environmental data. While GeoIPS is publically available, the pyCRTM functionality is available by request only.

5. Research and Real-Time Applications

5.1. Sensor Simulation in Microwave

pyCRTM provides a standalone Python interface for the forward calculations of CRTM, as discussed above. The Python interface provides more flexibility for applications such as calibration and validation (CalVal) of the satellite sensor data, inter-comparison between different radiative transfer models, different model background, and different static and dynamic ancillary inputs. NRL scientists have utilized pyCRTM to calibrate the simulated brightness temperatures for various existing and new microwave satellite sensors by using either the ECMWF or NAVGEM model outputs. We have established an efficient workflow to simulate brightness temperatures from the Advanced Technology Microwave Sounder (ATMS), Special Sensor Microwave Imager/Sounder (SSMIS), and WindSat. The sensor simulator is easily configured to incorporate new sensors such as the Compact Ocean Wind Vector Radiometer (COWVR) and the Temporal Experiment for Storms and Tropical Systems (TEMPEST). Such fast RTM simulations can be compared with the line-by-line RTM simulations to calibrate the satellite sensor performance, which then can be used to provide quality control statistics for NWP applications.

At NASA GMAO, evaluation of new microwave sensors has been done routinely via Observation System Simulation Experiments, and Observation System Experiments. However, these systems often don't have an easy method to quickly visualize vertical sensitivity via weighting functions. This is typically calculated using

$$WF = \frac{dTau}{dlog(P)} \qquad (1)$$

where Tau is the atmospheric transmission, and P is the atmospheric pressure. While pyCRTM does not output weighting functions directly, it may be computed by taking the transmission and taking the finite difference of transmission and log pressure. (e.g., using NumPy's diff function). An example of this is shown in Fig. 6 comparing water vapor sensitive channels from the Global Precipitation Mission Microwave Imager (GMI), and the Temporal Experiment for Storms and Tropical Systems - Demonstration (TEMPEST-D). Both are microwave instruments that sound the 183 GHz water vapor absorption line.

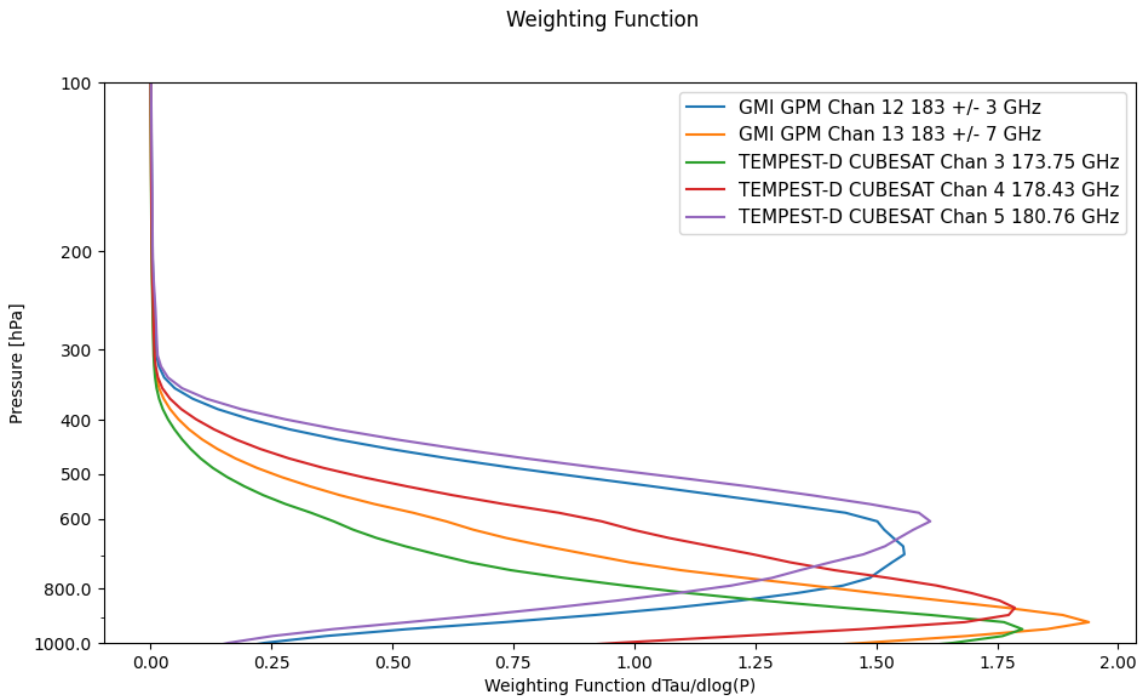

Figure 6: An example of weighting function computed utilizing pyCRTM for GMI and TEMPEST-D channels sensitive to water vapor.

Furthermore, pyCRTM retains the capability of the CRTM to compute the adjoint Jacobians for all possible instrument types, which is crucial for variational data assimilation applications and in-depth analysis of instruments. Similar to the weighting function, an example of the adjoint brightness temperature Jacobian w.r.t. the atmospheric input profile layer temperature is shown in Fig. 7 for the Advanced Microwave Sounding Unit-A (AMSU-A) instrument onboard the Meteorological Operational Satellite-A (MetOp-A).

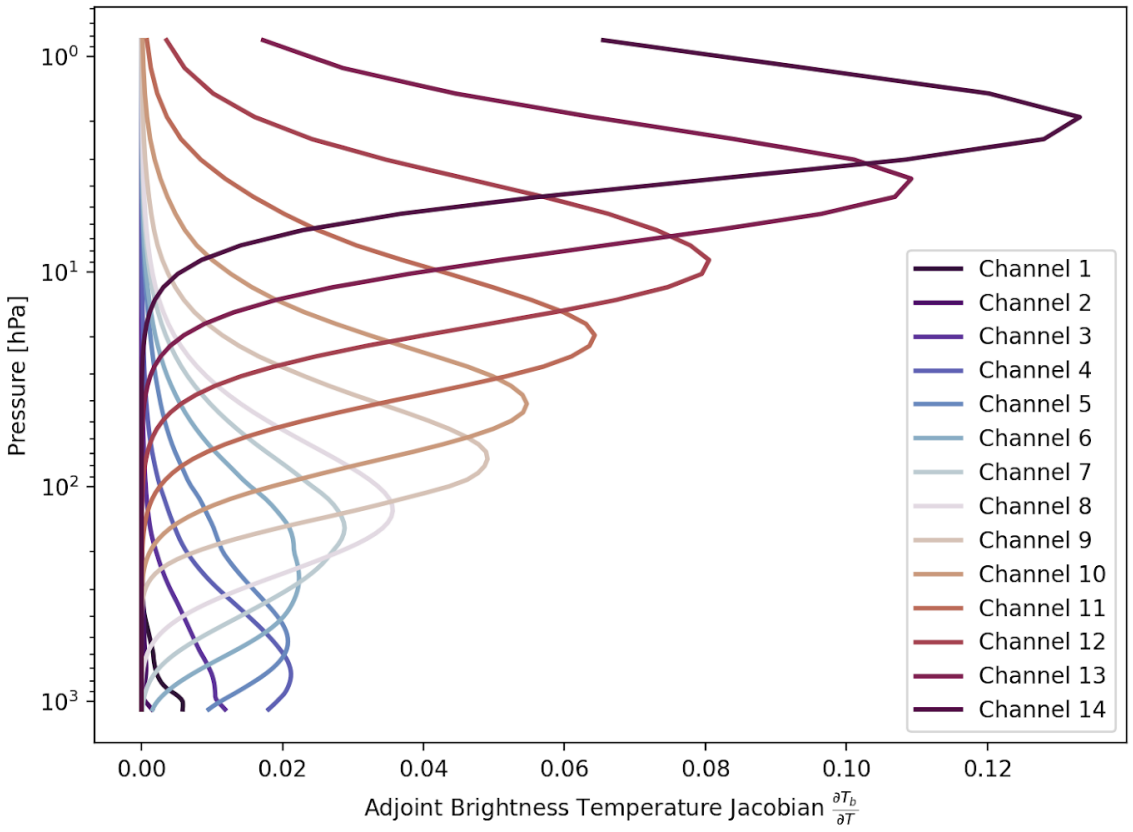

Figure 7: Adjoint brightness temperature Jacobian for channels 1 through 14 of AMSU-A MetOp-A.

This provides a further example for the interaction of pyCRTM with other frameworks in the Python ecosystem.
As the Jacobian is the linearized form (or K-Matrix) of the nonlinear pyCRTM Forward operator, an important first step in the analysis of an instrument is the singular value decomposition (SVD) of this K-Matrix, as it provides the basis for the identification of the row- and null space of a measurement, and the computation of the information content and Shannon entropy. NumPy here provides the facilities to store the K-Matrix conveniently in a matrix array and also to perform the SVD operation itself. The resulting left singular vectors and corresponding singular values for the AMSU-A case of Fig. 7 are shown in Fig. 8.

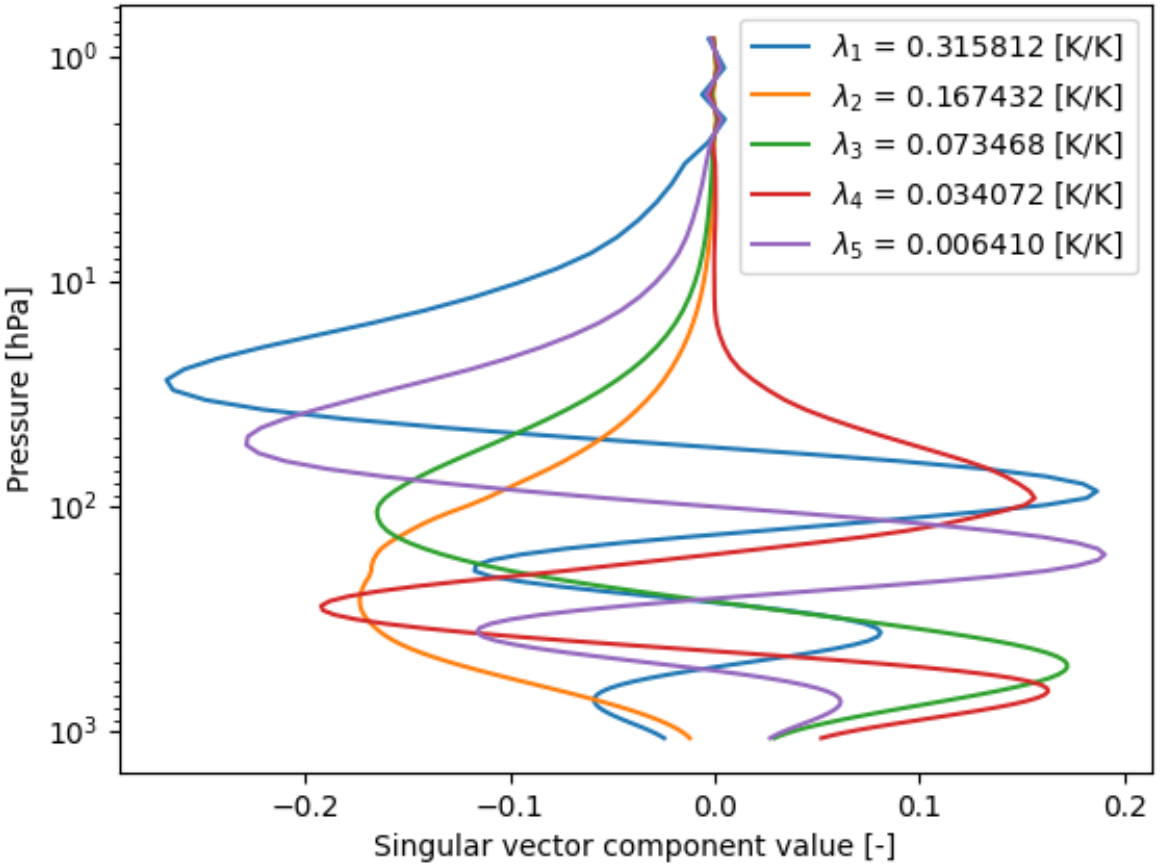

Figure 8: First 5 singular values and singular vectors of the K-Matrix shown in Fig. 7 for AMSU-A MetOp-A.

The listing of the corresponding Python code is given in Listing 1 and its clarity and brevity despite symbolizing such a complex operation is a good example for the principle to encapsulate and hide the complexity of a process from the user and for the conciseness of pyCRTM.

Listing 1: Seamless computation of a temperature Jacobian (K-Matrix) for AMSU-A MetOp-A and its SVD using pyCRTM and NumPy.

```
import numpy as np
...
crtmOb.runK()
kTb = crtmOb.Bt
u, s, vh = np.linalg.svd( kTb.T )
```

5.2. *Sensor Simulation and Visualization using GeoIPS*

An active area of development in numerical weather prediction is the use of model forecast data to produce simulated satellite imagery with a "look and feel" resembling the satellite data that many meteorologists use regularly (e.g. Bodas-Salcedo et al. [17]). Generation of "synthetic" microwave sensor imagery is a similar process to forward modeling involved in data assimilation, and also relies on models such as CRTM. Microwave sensor simulator capability for CalVal applications have been adapted and integrated with GeoIPS. Currently this capability supports typical microwave sensors such as SSMIS and AMSU-B, and model outputs from ECMWF, NAVGEM and COAMPS. Users can visualize the simulations by calling the script with input of satellite sensor and interested channels, which can be done either in batch or interactively using Jupyter notebooks or similar frameworks. Figs. 9 and 10 provide an example of global and regional visualization of synthetic SSMIS 37 GHz and 89 GHz brightness temperatures computed by CRTM based on NAVGEM model outputs.

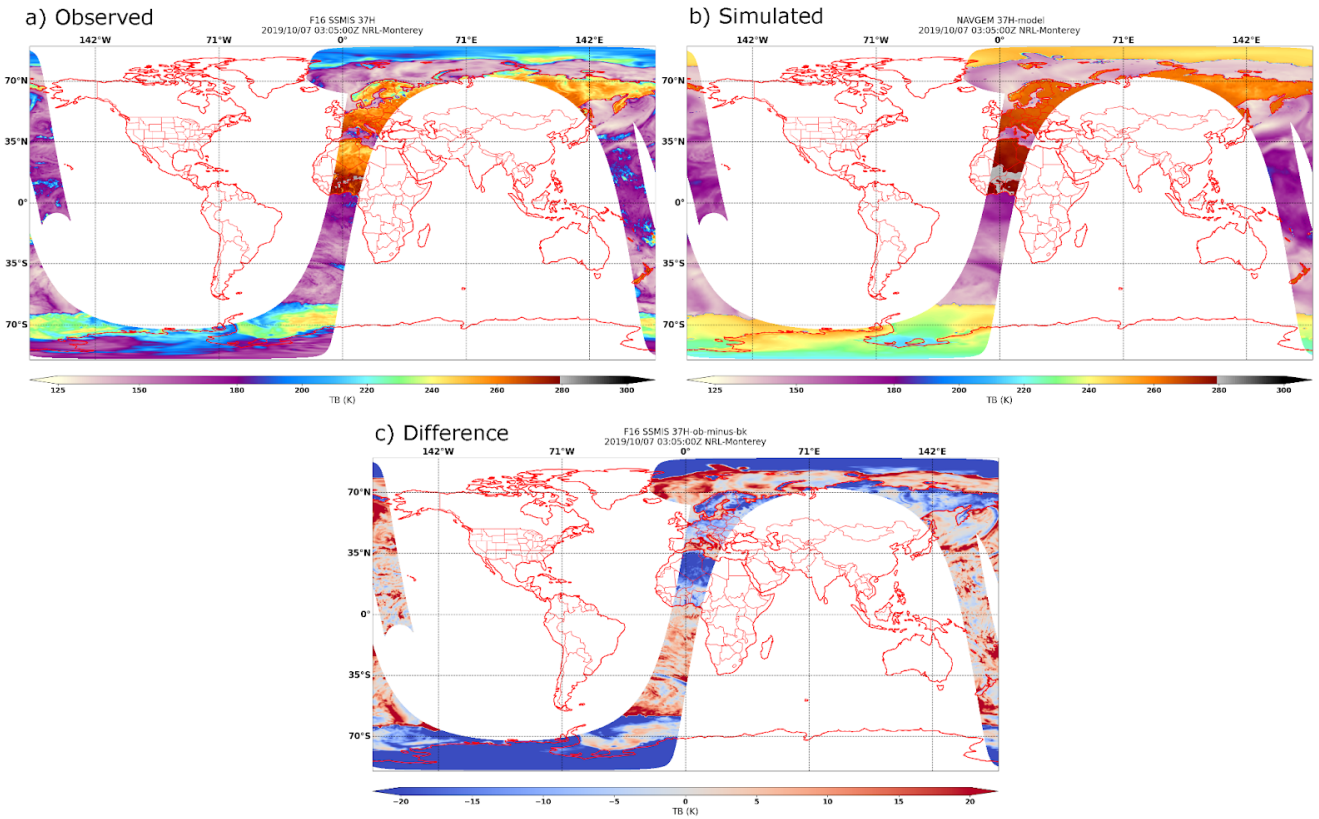

Figure 9: An example of SSMIS channel 37H brightness temperature (a), simulated brightness temperature using NAVGEM model outputs (b), and their differences (c).

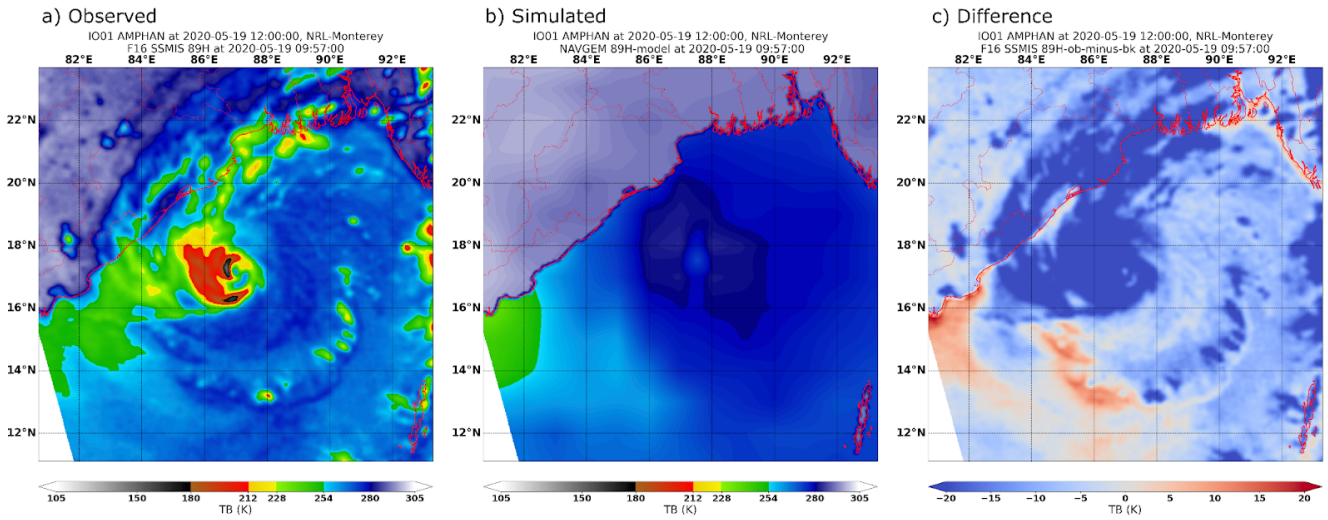

Figure 10: GeoIPS/pyCRTM observed (a) and simulated (b) SSMIS channel 89GHz brightness temperatures using NAVGEM model outputs, and their differences (c).

5.3. Hyperspectral Infrared Simulations Including Sensitivities

Another NRL application of pyCRTM relates to aerosol impacts on hyperspectral IR data. The impact of scattering and absorbing particles on satellite infrared radiances has been long recognized (e.g. Griggs [18]), and the quantitative studies have shown that 1) aerosol impacts can frequently be large relative to other uncertainties in forward calculations (Marquis et al. [19]) and 2) fairly precise estimates of aerosols are needed to constrain satellite radiances within tolerances needed by atmospheric and oceanic prediction systems (Bogdanoff et al. [20]). However, computation of vertically-resolved particle scattering and absorption adds computational cost and potentially time to the forward calculation of radiances. In preparation for implementation of aerosol-aware radiance DA, it was necessary to establish criteria for when the aerosol computations were likely to significantly affect the forward solutions. NRL approached this problem by applying aerosol fields from the Navy Aerosol Analysis and Prediction System (NAAPS, Lynch et al. [21]) using CRTM to estimate the impacts of aerosols on simulated infrared radiances, and the fraction of forward radiance computations that would need to include aerosol to achieve a specified uncertainty. The pyCRTM interface, working inside a Jupyter notebook, greatly simplified this experimental setup, and the output of results directly in the interactive Python notebook meant that visualization of results could be done immediately after computations were completed. The time saved by the use of Python tools for this work translates into additional testing, resulting in a robust method for establishing thresholds for operational implementation of aerosol-aware radiance assimilation.

At NASA GMAO, pyCRTM has proven to be a valuable tool in a number of studies using hyperspectral sounders. While data assimilation systems such as NASA's Global Earth Observing System- Atmospheric Data Assimilation System (GEOS-ADAS) , and NOAA's Global Data Assimilation System (GDAS) utilize Jacobians in the data assimilation process, they do not provide a convenient method to output or visualize temperature or constituent Jacobians. The ability to easily access Jacobians from the CRTM has enabled visualizations of the sensitivity of constituents, and temperature in data assimilation applications. An example is shown in Fig. 8 where temperature, water vapor and ozone Jacobians are calculated offline using pyCRTM. The analysis fields from the GEOS-ADAS are read in via netCDF and are provided to the

pyCRTM interface, and Jacobians are computed. The resulting Jacobians are then convolved with an estimate of the model background error and displayed in Fig. 11. This provides an estimate of the vertical sensitivity of temperature, water vapor, and ozone in the GEOS-ADAS. The longwave infrared (LWIR) and shortwave infrared (SWIR) are compared in Fig. 8 to highlight a potential benefit of adding CrIS shortwave channels to the GEOS-ADAS. Longwave infrared temperature sounding channels display a much higher sensitivity to water vapor and ozone, whereas temperature sounding channels in the shortwave infrared are primarily sensitive to temperature. A similar approach of convolving background error with Jacobians was utilized in Karpowicz et al. [22] along with Han and McNally [23] as a criteria for selecting ozone sensitive channels. Additionally, Karpowicz et al. [22] used pyCRTM to obtain Jacobians and weighting functions of ozone sensitive channels from hyperspectral instruments.

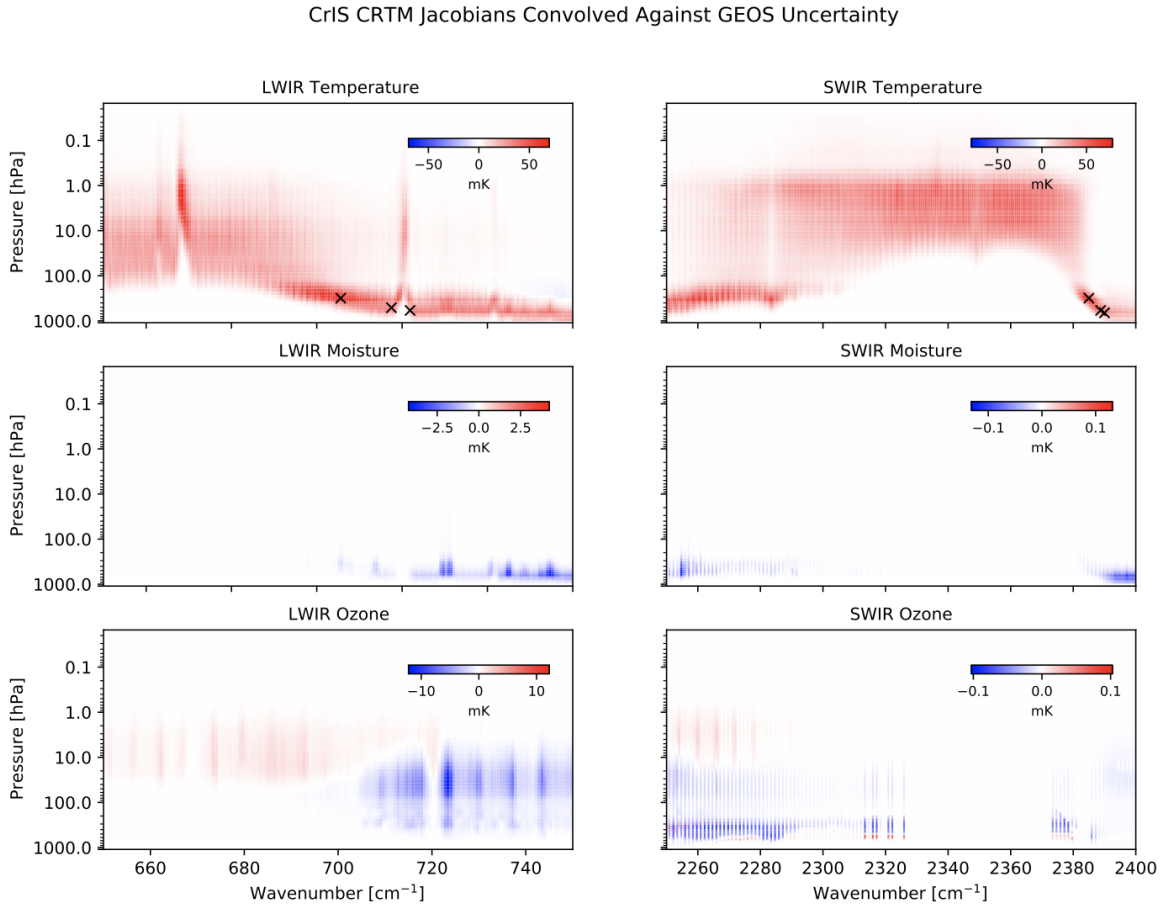

Figure 11: Jacobians computed from pyCRTM for the Cross-track Infrared Spectrometer (CrIS) convolved against an estimate of background error of the GEOS-ADAS .

6. Conclusion

The applications shown demonstrate the power of having a fast radiative transfer model paired with an interpretative language that allows for rapid prototyping for development. The ease of use provides an opportunity for students to learn the

basics of radiative transfer tools like CRTM using a widely taught interpreted language in modern undergraduate curricula, without having to learn Fortran which, as a more cumbersome compiled language, is more often taught in graduate courses. pyCRTM adds to the rich ecosystem of tools for research and development and is ideally suited for quick simulations and visualizations. The framework is flexible enough to provide simulations of varying complexity whether it be an infrared hyperspectral sounder simulation with aerosols and clouds present, or a simple simulation of cloud free microwave brightness temperature over ocean. The pyCRTM interface is freely available on the JCSDA github repository https://github.com/JCSDA/pycrtm.

Acknowledgements

NRL work is supported by the Office of Naval Research, Code 322, project number N0001421WX00385. NASA GMAO work supported by NASA's Modeling Analysis and Prediction (MAP) funding.
Joint Center for Satellite Data Assimilation research in cooperation with the NOAA OAR Office of Weather and Air Quality is supported by NOAA's Science Collaboration Program and administered by UCAR's Cooperative Programs for the Advancement of Earth System Science (CPAESS) under awards NA16NWS4620043, NA18NWS4620043B, and NA20NWS4620043C.

References

[1] Han Y, van Delst P, Liu Q, Weng F, Yan B, Treadon R, et al. JCSDA Community Radiative Transfer Model (CRTM)-Version 1. 2006.

[2] Chen Y, Weng F, Han Y, Liu Q. Validation of the community radiative transfer model by using cloudsat data. Journal of Geophysical Research Atmospheres 2009;114. https://doi.org/10.1029/2007JD009561.

[3] Tang G, Yang P, Stegmann PG, Panetta RL, Tsang L, Johnson BT. Effect of Particle Shape, Density, and Inhomogeneity on the Microwave Optical Properties of Graupel and Hailstones. IEEE Transactions on Geosciences and Remote Sensing 2017;55(11). https://doi.org/10.1109/TGRS.2017.2726994.

[4] Stegmann PG, Tang G, Yang P, Johnson BT. A stochastic model for density-dependent microwave Snow- and Graupel scattering coefficients of the NOAA JCSDA community radiative transfer model. Journal of Quantitative Spectroscopy and Radiative Transfer 202018;211, 9-24. https://doi.org/10.1016/j.jqsrt.2018.02.026.

[5] Lu CH, Liu Q, Wei SW, Johnson BT, Dang C, Stegmann PG, Grogan D, Ge G, Hu M. The Aerosol Module in the Community Radiative Transfer Model (v2.2 and v2.3): accounting for aerosol transmittance effects on the radiance observation operator. Geoscientific Model Development Discussion [preprint] 2021. https://doi.org/10.5194/gmd-2021-145.

[6] Wei SW, Lu CH, Johnson BT, Dang C, Stegmann P, Grogan D, Ge G, Hu M. The Influence of Aerosols on Satellite Infrared Radiance Simulations and Jacobians: Numerical Experiments of CRTM and GSI. MDPI Remote Sensing 2022;14(3). https://doi.org/10.3390/rs14030683.

[7] Saunders R, Hocking J, Turner E, Rayer P, Rundle D, Brunel P, et al. An update on the RTTOV fast radiative transfer model (currently at version 12). Geoscientific Model Development 2018;11:2717–37. https://doi.org/10.5194/gmd-11-2717-2018.

[8] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020).

[9] J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007.

[10] May, R. M., Arms, S. C., Marsh, P., Bruning, E., Leeman, J. R., Goebbert, K., Thielen, J. E., Bruick, Z., and Camron, M. D., 2022: MetPy: Python Package for Meteorological Data. Unidata, https://github.com/Unidata/MetPy, doi:10.5065/D6WW7G29.

[11] Cartopy. V0.6. 19-Feb-2013. Met Office. UK. http://github.com/SciTools/cartopy/archive/v0.6.0.tar.gz

[12] Wolfram Research, Inc. (n.d.). Mathematica, Version 12.0.

[13] Knuth, Donald E. (1992). Literate Programming. California: Stanford University Center for the Study of Language and Information. ISBN 978-0-937073-80-3.

[14] Kluyver, T., Ragan-Kelley, B., Fernando Perez, Granger, B., Bussonnier, M., Frederic, J., … Willing, C. (2016). Jupyter Notebooks – a publishing format for reproducible computational workflows. In F. Loizides & B. Schmidt (Eds.), Positioning and Power in Academic Publishing: Players, Agents and Agendas (pp. 87–90).

[15] "nbgrader 0.6.1 documentation". nbgrader.readthedocs.io. Retrieved 2020-11-13.

[16] "GeoIPS2". https://github.com/USNavalResearchLaboratory/geoips2. Retrieved 2022-04-07.

[17] Bodas-Salcedo A, Webb MJ, Bony S, Chepfer H, Dufresne JL, Klein SA, et al. COSP: Satellite simulation software for model assessment. Bulletin of the American Meteorological Society 2011;92:1023–43. https://doi.org/10.1175/2011BAMS2856.1.

[18] Griggs M. A METHOD TO CORRECT SATELLITE MEASUREMENTS OF SEA SURFACE TEMPERATURE FOR THE EFFECTS OF ATMOSPHERIC AEROSOLS.Journal of Geophysical Research: Atmospheres, Volume 90, Issue D7, pp. 12,951-12,959,. 1985.

[19] Marquis JW, Oyola MI, Campbell JR, Ruston BC, Córdoba-Jabonero C, Cuevas E, et al. Conceptualizing the impact of dust-contaminated infrared radiances on data assimilation for numerical weather prediction. Journal of Atmospheric and Oceanic Technology 2021;38:209–21. https://doi.org/10.1175/JTECH-D-19-0125.1.

[20] Bogdanoff AS, Westphal DL, Campbell JR, Cummings JA, Hyer EJ, Reid JS, et al. Sensitivity of infrared sea surface temperature retrievals to the vertical distribution of airborne dust aerosol. Remote Sensing of Environment 2015;159:1–13. https://doi.org/10.1016/j.rse.2014.12.002.

[21] Lynch P, Reid JS, Westphal DL, Zhang J, Hogan TF, Hyer EJ, et al. An 11-year global gridded aerosol optical thickness reanalysis (v1.0) for atmospheric and climate sciences. Geoscientific Model Development 2016;9:1489–522. https://doi.org/10.5194/gmd-9-1489-2016.

[22] Karpowicz BM, McCarty W, Wargan K. Investigating the utility of hyperspectral sounders in the 9.6 μm band to improve ozone analyses. Quarterly Journal of the Royal Meteorological Society 2021. https://doi.org/10.1002/qj.4198.

[23] Han W, McNally AP. The 4D-Var assimilation of ozone-sensitive infrared radiances measured by IASI. Quarterly Journal of the Royal Meteorological Society 2010;136:2025–37. https://doi.org/10.1002/qj.708.