AEROSPACE REPORT NO.
ATR-2019-02679

AE9/AP9-IRENE Module Architecture (Static Environments)

December 1, 2019

T. P. O'Brien
Space Sciences Department
Space Science Applications Laboratory

Prepared for

Senior Vice President
Engineering and Technology Group

Authorized by: Engineering and Technology Group

Acknowledgments

The author acknowledges useful discussions with S. Huston, W. R. Johnston, C. Roth, T. Guild, and P. Jiggens regarding the challenges and solutions presented in this report. This work was funded by The Aerospace Corporation's Technical Investment Program, as part of a Long-Term Capability Development project. MATLAB is a registered trademark of The MathWorks, Inc. Python is a registered trademark of the Python Software Foundation. Octave is a registered trademark of Carnegie Mellon University.

Abstract

Initial versions of the AE9/AP9-IRENE climatology model separated energetic (>~100 keV) particles into high- and low-altitude grids that were developed and run together, and low energy (<~100 keV) particles into a completely separate plasma model. The high energy and plasma results were combined in post-processing. This arrangement led to some anomalies, such as discontinuities at the energy boundary and statistical perturbations that should have been correlated across the energy boundary between models being handled independently. Further, this initial structure limits the extensibility of the model to accommodate sub-models (or "modules") of specific regions of space or particle populations. In this report, we describe development of a new architecture that abstracts the different regions or particle populations into separate modules. This new module architecture allows the plasma particles to be combined correctly and smoothly with the higher-energy particles at runtime while also generalizing and standardizing the interconnections between modules. The new architecture, therefore, addresses statistical shortcomings of the older architecture while also preparing for future extensions when new particle populations, such as solar protons, are incorporated into the model. At this time, the module architecture only covers static environments, since the plasma environment does not yet have dynamic capability to be merged with the dynamics of higher-energy particles.

Contents

Figures

Tables

1. Introduction

AE9/AP9-IRENE is a climatology model of the trapped radiation and plasma near Earth, which is periodically updated with new data, new capabilities, and improved methodology [2][3][9]. In this report, we describe the first stage in a major architectural change, variously referred to as the "version 2.0 architecture" or the "module architecture." In the current architecture (through at least version 1.5), there are separate high-energy (radiation) models (AE9 and AP9) and low-energy (plasma) models (SPME, SPMH, etc.). The high-energy models have high- and low-altitude sub-grids within them; combining high- and low-energy models is done in post-processing. In the new architecture, the model consists of single-grid modules that are stitched together at runtime. In this first stage of the module architecture transition, we implement mainly the changes needed for static environments, leaving the upgrade of Monte Carlo dynamics to a future development.

This report is broken into three technical areas: (1) changes to the precomputed tables and the turnkey system that generates them, (2) changes to the runtime calculations that evaluate the model and compute quantities requested by the user, and (3) future development needs to fully flesh out the module architecture as envisioned by the AE9/AP9-IRENE team and its users. The prototype code implements the turnkey and runtime changes, and this document explains the purpose and algorithms for them. Included in the exposition is a brief demonstration of how the module architecture stitches the low- and high-energy components of the proton environment together.

2. Changes to the Turnkey System and Precomputed Tables

One important change in the module architecture is that the turnkey code that generates the runtime tables is now written in Python. This rewrite was largely a translation of the old MATLAB turnkey code, with changes for the module architecture made as needed along the way. The change to Python dramatically simplifies the use of parallelization, since MATLAB is licensed software and requires purchase and management of every license used in a multithread/multicore environment. In prior versions, the management of MATLAB licenses was avoided through the use of GNU Octave [1], a free, open-source MATLAB work-alike for which parallelization was possible but not a native behavior. Since Python is free open-source software, it has no such license restrictions and has robust support for parallelization built in. The other major changes to the turnkey system are:

- breaking up the high- and low-altitude AE9/AP9 models into distinct modules
- associated restructuring of the data files
- coordinating sensor groupings across modules for bootstrapping
- designating which models have correlated "Stheta" (S_θ) for perturbing flux maps

The original turnkey system is described in *Generation of AE9/AP9/SPM V1.0 Runtime Tables* (Aerospace Report Number TOR-2014-00295) [6].

The module architecture breaks up the high- and low-altitude portions of the AE9 and AP9 sub-models into individual modules. Every module now covers only a single grid. So, whereas there was previously a single AP9 model with two sub-grids, there is now a pair of modules: AP9V20Kphi, which covers the high-altitude grid (in K, Φ coordinates), and AP9V20Khmin, which covers the low-altitude grid (in K,h_{min} coordinates). AE9 is similarly split into AE9V20KPhi and AE9V20Khmin. The SPMH and SPME models are recast as modules SPMHV20AlphaLm and SPMEV20AlphaLm.

Although the module architecture does not yet address Monte Carlo mode, the module data files no longer store any Monte Carlo information (previously the AE9 and AP9 data files stored the Monte Carlo evolution tables). In the new architecture, the data files are plain HDF5 files (not MATLAB .mat files stored in MATLAB's structured HDF5 format). Table 1 provides a short summary of the contents of a module file. A typical runtime tables file has a name such as AP9V20Khmin_runtime_tables.h5. An accompanying idiosyncratic Stheta file AP9V20Khmin_runtime_tables_idio_Stheta.h5 contains only *whichm*, *info_string*, and *Stheta* for use in idiosyncratic runs that represent model variants that might occur if only a single in situ dataset were available (used for model validation).

Another change to the turnkey system is that sensors are grouped together at the top (project) level, rather than at the model or module level. Thus all sensors from a given mission are grouped together. Part of estimating the model errors is resampling different sets of sensors and rebuilding the flux maps to determine how much the addition of new sensor data will change the flux maps. Resampling during bootstrapping causes the same missions to be included in the same bootstrap group across modules. For example, bootstrap draw 1 might include the Polar mission but not Van Allen Probes, and this will be true regardless of which module is being built (AE9V20Khmin, AE9V20KPhi, AP9V20Khmin, SPMHV20AlphaLm, etc.). By controlling the seeding of the random number generator, the same sensor groups are drawn in the same order for all modules when bootstrapping over groupings. If a group is drawn with no sensors for the module being constructed, another group is drawn at random (from a different random number generator so as not to affect the progression through the seeded random sequence). This is one part of enabling the possibility for correlated model errors across modules. For the idiosyncratic S_θ, the same approach is taken, but only one sensor group is drawn with the seeded random

number generator, and only one sensor is drawn from that group, using a different random number generator.

The other part of ensuring model errors are correlated across modules is S_θ grouping. (S_θ) can be thought of as a large matrix where the columns represent different bootstrap cases, and the rows span the grid of the module first for θ_1 then for θ_2 (the parameters of the flux probability distribution at each grid point). All modules that should have correlated errors are listed in a settings file (settings.py) as belonging to an S_θ group. At present, the S_θ groups are done only by species so that, for example, there are three modules in the proton group (AP9V20Khmin, AP9V20KPhi, and SPHMV20AlphaLm). Because the different bootstraps used to generate module-specific S_θ matrices share the same sensor groups across all modules, all their columns correspond to the same combinations of missions. Therefore, as illustrated in Figure 1, we can assemble a super- S_θ by stacking the individual S_θ matrices for the entire group (each S_θ matrix is appended as additional rows in the super- S_θ matrix). We use linear algebra (singular value decomposition) to simplify this super-S_θ matrix into fewer columns, which represent the essential uncertainty in the flux maps. Then we break this reduced super- S_θ matrix back up into new S_θ matrices, one for each module. These new S_θ matrices all have the same number of columns as each other (fewer columns than the original), and they all have the same number of rows as the original S_θ matrix for the module they came from. This whole procedure enables generation of correlated errors at runtime because the process of building these S_θ matrices has preserved the uncertainty associated with adding or subtracting a subset of missions to or from our observation set.

The rest of the turnkey system was translated from MATLAB to Python with essentially no change. The entire process is illustrated in Figure 2. A minor note is that the MATLAB version used many temporary files stored in .mat format (MATLAB's HDF5-based save format), while the Python version uses so-called "pickle" files. Whereas the MATLAB save files could, in principle, be read by many other languages, Python pickle files are not easily read by other languages. A future improvement to the turnkey code may be to replace the pickle files with a structured use of HDF5 or other hierarchical data file format.

Table 1. Contents of a Module File

Variable	Type	Contents
whichm	string	module name, e.g., AE9V20KPhi
fluxunit	string	#/cm^2/sr/s/MeV
species	string	Particle species, e.g., electron
marginal_type	string	Marginal distribution type, e.g., lognormal
stheta_perturb_style	string	How Stheta is perturbed, e.g., uniform
theta	Nx2 real	Statistical parameters of flux map
Stheta	(2N)xM real	Flux map parameters error covariance
info_string	string	Module build information
capabilities	list of strings	List of module capabilities, e.g., MEAN,…
grid	structure	Structured module grid description

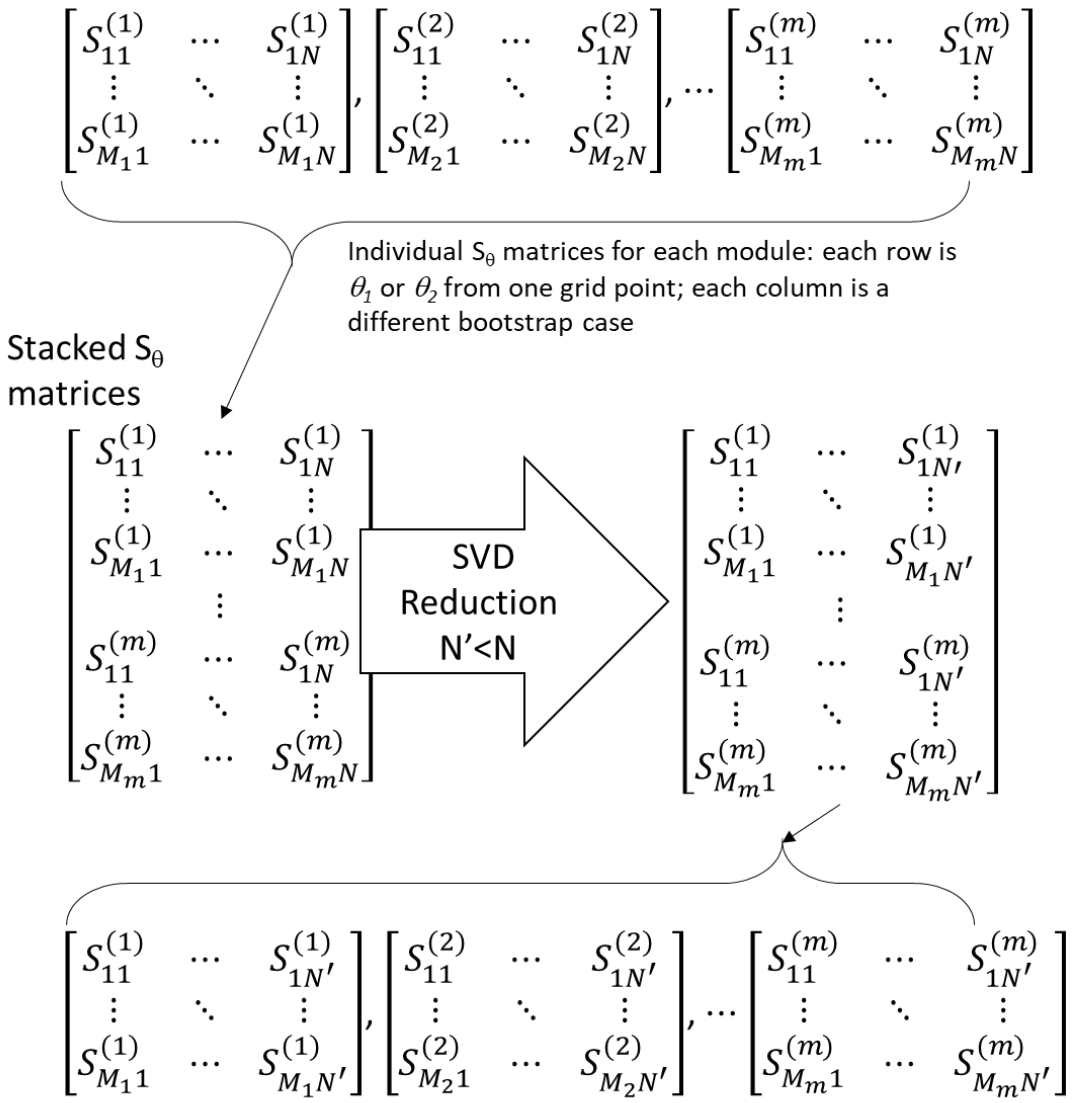

Figure 1. The process of combining, reducing, and then parsing the S_θ matrices to ensure flux map uncertainties are correlated across models.

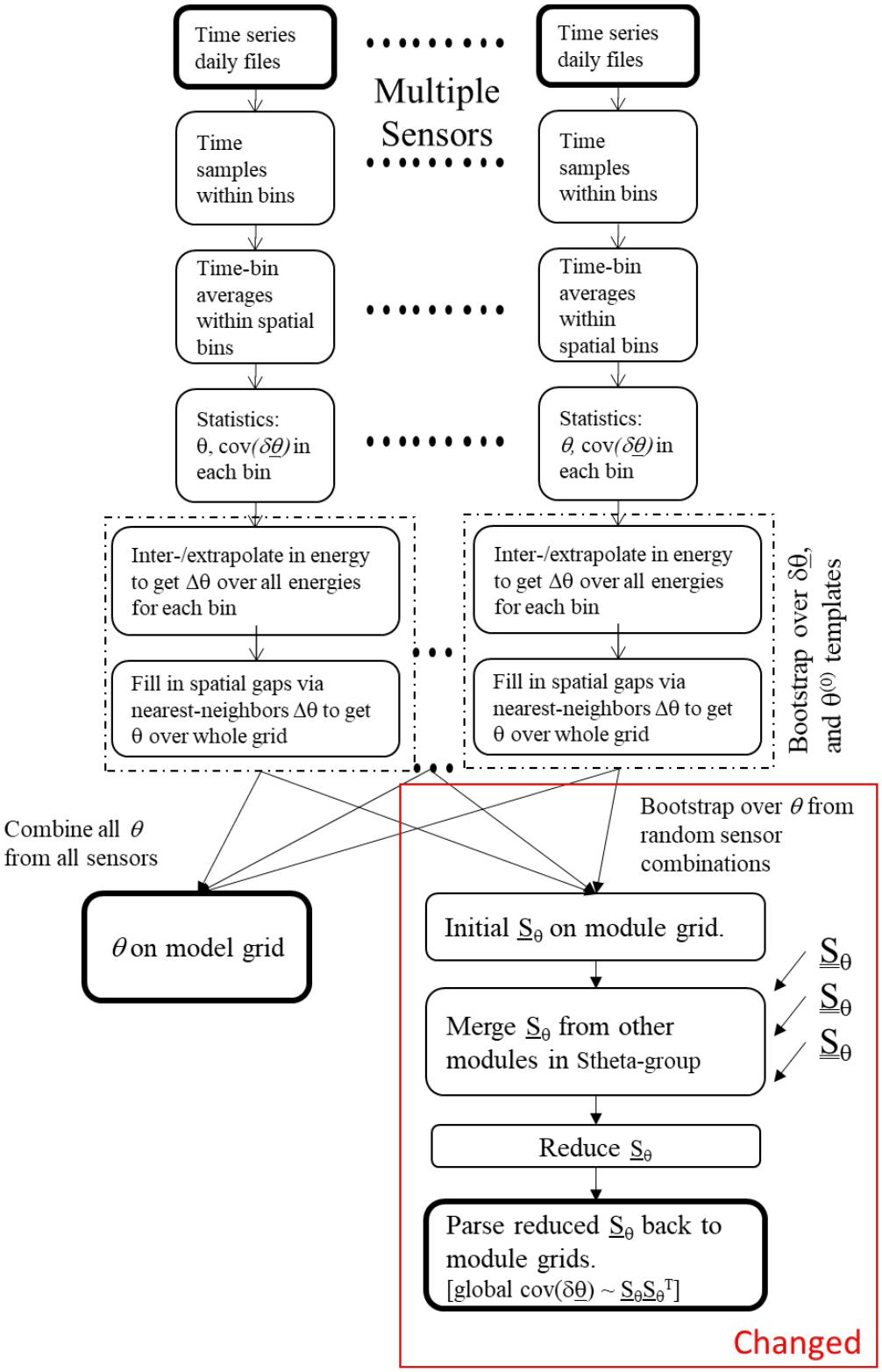

Figure 2. The turnkey system for generating the runtime data tables used by the module architecture.

3. Changes to the Runtime Algorithms

The bulk of the changes to realize the module architecture occur in the runtime algorithms. Some time ago, the runtime algorithms had already been rewritten in Python from a MATLAB prototype, so incorporating the new module architecture algorithms was a more typical modification, not a rewrite-in-translation.

At a high level, the new architecture starts with a module catalog that specifies which modules are available, what energies they cover, and how they are stitched together. To ensure that expensive magnetic coordinate calculations are not repeated, a coordinate manager is introduced to cache and share coordinate calculations between modules. Because stitching may occur across energies, we also introduce a general-purpose concept of a "flux transform," which converts differential flux into other quantities. This transform frees the module from having to do the work of converting to user's requested energy channels (and types of energy channels). It can also incorporate the kernel function that converts from flux to dose or other effect quantities.

3.1 Module Catalog

The module catalogue is an eXtensible Markup Language (XML) text file. Figure 3 depicts the module catalog only for the proton modules, although the catalogue would normally include electrons and ions as well. The figure shows three modules and two stitches. Each module has a species and a lower and upper energy limit, in MeV. Each stitch has a species, coordinate, and transform type. A "Linear" stitch progressively weights the lower module from 1 to 0 and the upper module from 0 to 1, in a linear fashion, as the designated coordinate varies from the lower to upper limit. A "Log" transform type indicates that the transition takes place as a linear function of the logarithm of the stitch coordinate. The sign of the transition is implied by whether the upper and lower limit are in increasing or decreasing order. A stitch may be applied simultaneously to multiple upper or lower modules, as is seen for the energy stitch—it stitches both AP9 modules as upper modules to the SPMH lower module. Whereas in the prior versions of AE9/AP9-IRENE, the user specified which model to run, the new architecture allows the user to specify what outputs are desired, and the module catalog is used to determine which modules are needed.

3.2 Output Requests

The user specifies one or more output requests. An output request requires a species, a transform, and an accumulator. The species designators are e-, H+, He+, and O+. A transform can be a differential flux interpolation (i.e., from differential flux to differential flux but on different energy channels), a wide flux transform (energy channels with finite width), an integral flux transform, or a kernel flux transform [5][8]. In the past, conversion from model energy channels to user-requested energy channels was considered part of the calculation of the model weights (spatial and energy interpolation/integration were done together) [4]. In the module architecture, the modules work exclusively at their native energy channels, and each module can use its own idiosyncratic energy grid. By allowing multiple output requests in a single run, the new architecture allows calculation of differential and integral channels in the same run. The accumulators are an extended set of the ones already available in prior versions of AE9/AP9-IRENE: mean, fluence, running boxcar average, running exponential average [7]. An option (usually employed) is provided to the boxcar average and exponential average to report the running maximum. The accumulators actually manage the flux transforms, since this can gain some efficiencies both in terms of compute time and numerics. For example, the averaging is linear, so computation can be saved by applying the kernels only when reporting results. However, if running maxima are needed, maximum is a nonlinear operation, so the kernels must be applied every time step. Conversely, energy interpolation transforms should be thought of as nonlinear kernels and should be performed after all the linear

operations to achieve the best numerics. We note that the module architecture will assume power-law and semi-log flux interpolation between energy grid points when performing energy interpolation and integration, which is why these flux-to-flux transforms are effectively nonlinear, even if they are nominally linear operations (see Appendix A).

An output request can also supply optional report cadence, specific report times, and a specific run type. The report cadence is helpful when the user needs to know the accumulator results at a uniform cadence, such as daily, monthly, or yearly. The specified report times are useful for non-periodic orbits, such as solar-electric orbit raising. Together, these options reduce the amount of output produced by the model, and they can also reduce the amount of computation, as, for example, kernel transforms may only need to be performed at reporting times, not at all time steps. The run types are an extended set of options already available in AE9/AP9-IRENE: mean, perturbed mean, idiosyncratic perturbed mean, Monte Carlo, unperturbed Monte Carlo, idiosyncratic perturbed Monte Carlo, static percentile, perturbed percentile, idiosyncratic perturbed percentile. The idiosyncratic variants use the idiosyncratic S_θ to perform flux map perturbations. Some of these run modes have further options, such as percentile or scenario. Since multiple output requests are allowed in a single simulation, multiple scenarios or even multiple run modes can be performed, allowing maximum re-use of calculations. Note: the *flyin* function (actually called *flyin2* to distinguish it from the function used in prior versions) allows top-level options to control the default report cadence/times and run mode, or those options can be specified uniquely with some or all output requests.

3.3 Run Preparation

One of the first steps in the new architecture is identifying which modules are active. The first determinant is whether the module species matches any of the output request species. The second determinant is energy range: a module is active unless its lower bound is higher than the upper energy requested for all output requests, or its upper bound is lower than the lower energy requested for all output requests for its species. (At this time, the module architecture prototype does not support multi-species output requests, such as “total dose,” which combines dose from protons and electrons.)

Once the active modules are identified, the active stitches must be identified. A stitch is active if at least one of its lower modules is active and at least one of its upper modules is active.

If the user has requested omnidirectional quantities (integrated over all angles of incidence), a virtual angle grid is created that starts at 5 degrees local pitch angle, and then has points at 10, 20, … 90 degrees. This is the same pitch angle grid used for omnidirectional calculations in the prior versions of the model.

A coordinate manager (CM) is then created, which caches magnetic coordinates and partial results needed for their computation. It is used to prevent repetition of costly calculations, such as magnetic field line tracing or evaluation of neural networks. The CM stores all the computed coordinates as a function of time and direction (angle).

The next step is to determine which energies are used from each active module by figuring out which module energy channels are needed by each transform. For example, if the user only asks for a few differential energy channels, then only the surrounding energies are needed for interpolation. A set of flags is used for each module to keep track of which energies are used, saving calculation time spent generating fluxes at unused energies.

The final step is to initialize the scenes used by each of the run modes. A scene instantiates (and, in Monte Carlo mode, evolves) the flux map for a given module for a given run mode. Since a scene is tied to a run mode, scenes can be shared by output requests that use the same run mode. For example, all output

requests using perturbed mean scenario 32 can share the same scenes (one scene for each module). This scene sharing saves memory and compute time.

Figure 4 provides an overview of the runtime algorithm for the module architecture, including the preparation steps as well as the time-stepping loop.

3.4 Time-Stepping Loop

After the run preparation is finished, the runtime algorithm actually begins to loop through the time steps requested by the user. At each time step, the spacecraft's magnetic coordinates (with the CM) are computed, and the interpolation weights from the model coordinates onto the spacecraft location. Sometimes these quantities are precomputed and stored. Because they do not involve energy interpolation/integration, these weights are the same across all scenes for a given module at a given time step for all output requests.

Next, the algorithm loops over each output request interpolating the flux from the scene flux map onto the spacecraft location using the module-specific weights. Then, separately for each direction, the fluxes are stitched together. Each stitcher provides a weight from 0 to 1 for its associated modules. Each flux (which depends on energy and direction) is multiplied by each weight for its associated active stitchers. There may be multiple active stitchers for a given module (e.g., an h_{min} stitcher and an energy stitcher), and both weights are multiplied by the associated flux. At this point, each module contributes flux versus energy and direction, but the energies are module specific. A merged energy list is created that includes all the energies from all the active modules, and the fluxes from each module are interpolated onto this energy list. At out-of-range energies, interpolated fluxes are set to zero. Otherwise, log-log interpolation is used when fluxes are positive, and log-linear interpolation is used when one involved flux is zero. We usually refer to this as "smart log interpolation," and it is given by Eq. (a) in Appendix A. The interpolated fluxes from each module are then added together. If the user requested omnidirectional flux, at this time, the omnidirectional integral is performed using the same angular weighting scheme from prior versions:

$$h_i = \frac{\alpha_{i+1}\left(I_0(\alpha_{i+1}) - I_0(\alpha_i)\right) - \left(I_1(\alpha_{i+1}) - I_1(\alpha_i)\right)}{\alpha_{i+1} - \alpha_i} + \frac{\left(I_1(\alpha_i) - I_0(\alpha_{i-1})\right) - \alpha_{i-1}\left(I_0(\alpha_i) - I_0(\alpha_{i-1})\right)}{\alpha_i - \alpha_{i-1}} \tag{1}$$

$$I_0(\alpha) = 1 - \cos\alpha \tag{2}$$

$$I_1(\alpha) = \sin\alpha - \alpha\cos\alpha \tag{3}$$

Here, local pitch angle α is taken in radians. When one α_{i+1} or α_{i-1} is off either end of the list of pitch angles, the term it's used in is left out of h_i. The angular integral must be done after the stitching because, at a given location, different angles of incidence may come from different modules; e.g., particles coming from near the magnetic field direction (low pitch angle) will come from the *K*-h_{min} module, while particles coming perpendicular to the magnetic field direction may come from the *K*-Φ module, so the stitching is angle dependent.

At this point in the time-stepping loop, the flux for each output request is fed into its accumulator, which handles the flux transform, if needed, as well as time integration and averaging. Then the time tag is compared to output reporting times/cadence to determine whether it is time to report the results. If so, the output request, accumulator, and flux transform are used together as needed to produce the required output, which is stored for the user.

```
<ModuleCatalog>

  <Module>
    <Name>AP9V20KPhi</Name>
    <Species>H+</Species>
    <LowEnergy>0.1</LowEnergy>
    <HighEnergy>2000</HighEnergy>
  </Module>

  <Module>
    <Name>AP9V20Khmin</Name>
    <Species>H+</Species>
    <LowEnergy>0.1</LowEnergy>
    <HighEnergy>2000</HighEnergy>
  </Module>

  <Module>
    <Name>SPMHV20AlphaLm</Name>
    <Species>H+</Species>
    <LowEnergy>0.001</LowEnergy>
    <HighEnergy>0.1643</HighEnergy>
  </Module>

  <Stitch>
    <Species>H+</Species>
    <LowerModule>SPMHV20AlphaLm</LowerModule>
    <UpperModule>AP9V20KPhi</UpperModule>
    <UpperModule>AP9V20Khmin</UpperModule>
    <Coordinate>Energy</Coordinate>
    <TransformType>Log</TransformType>
    <LowerLimit>0.1</LowerLimit>
    <UpperLimit>0.1643</UpperLimit>
  </Stitch>

  <Stitch>
    <Species>H+</Species>
    <LowerModule>AP9V20Khmin</LowerModule>
    <UpperModule>AP9V20KPhi</UpperModule>
    <Coordinate>hmin</Coordinate>
    <TransformType>Linear</TransformType>
    <LowerLimit>600</LowerLimit>
    <UpperLimit>1000</UpperLimit>
  </Stitch>
…

</ModuleCatalog>
```

Figure 3. Module catalog XML file.

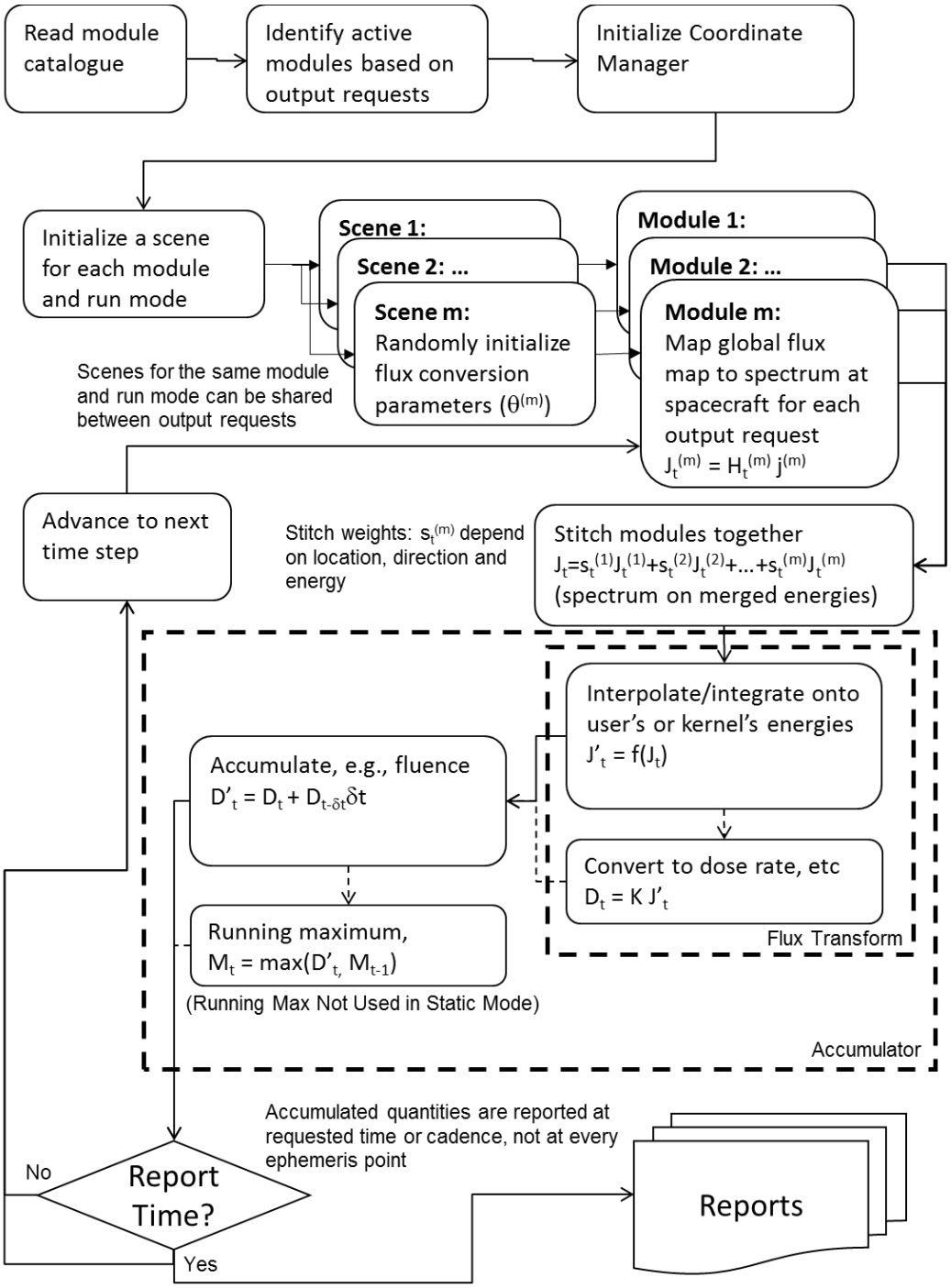

Figure 4. The runtime algorithm for the module architecture.

4. Stitching Demonstration

To demonstrate the stitching in a realistic case, we flew a geostationary orbit through one day in the new module architecture. We ran 40 perturbed mean scenarios, and computed the 50-, 75-, and 95-percent confidence levels (CLs). The Python prototype code allows an option to exclude or include only certain modules. This enables us to show the SPMH (plasma) and AP9 (high energy) parts of the model in isolation. The AP9 part of the model itself includes the AP9V20Khmin (low altitude) and AP9V20KPhi (high altitude) spatial modules, which are stitched together at runtime. The SPMH part of the model is a single module, SPMHV20AlphaLm. Normally, all three of the modules are run together, but we have broken out SPMH and AP9 to show how the earlier versions (prior to v2.0) would appear.

Figure 5 shows that SPMH still has unrealistically little uncertainty, causing all three CLs to be nearly the same (they differ by only a few percent). This is a known issue that is still being worked by the AE9/AP9 development team. SPMH also has slightly lower flux at its high end than AP9 has at its low end, at least at geostationary orbit. Manually connecting the SPMH CLs to AP9 CLs would create a discontinuity at the low-high energy boundary. The module architecture resolves this issue by gradually stitching AP9 and SPMH together with a log-linear transition from 100 to 164.3 keV at every point along the spacecraft orbit, well before the CLs are computed. The resulting CLs show a more realistic smooth transition from the very low uncertainty in SPMH at low energies to the more realistic uncertainties at high energy in AP9. Thus the module architecture has achieved one of its goals: to remove unphysical discontinuities from the statistical outputs of the model.

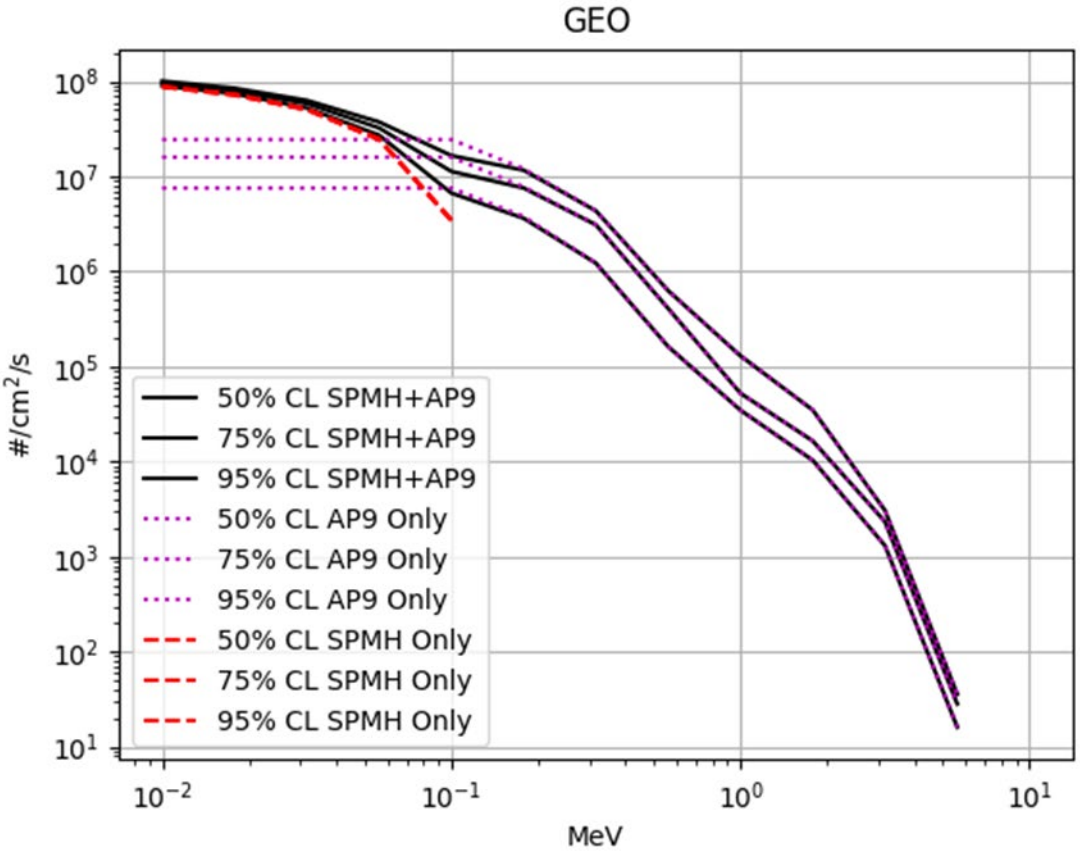

Figure 5. Average integral proton flux at geostationary orbit for the plasma (SPMH) and high-energy (AP9) parts of the model at three CLs.

5. Future Improvements

Several future improvements are apparent or necessary at this stage in the module architecture development. First and foremost is incorporation of a Monte Carlo engine. Presumably this will be another major tag in the module catalog, and its code counterpart will manage the shared state among the modules that coordinate their dynamic evolution. A version of this already exists in the AE9 and AP9 models in v1.5 and prior, but its structure is hard-coded into the model. A new framework will be needed to abstract that structure and move its settings into the module catalog. One of the biggest challenges is how/whether to allow modules with different time cadences to share state, or whether different Monte Carlo engines at different cadences will be used depending on which modules are active.

The next needed improvement is a way to manage output requests that cover multiple species. The first of these will need to address "total" dose that includes proton and electron effects. Farther in the future, single-event effects output requests will be needed that combine proton and heavy ion effects.

In this report, we have looked only at protons. That was driven in part by the fact that the electron plasma model (SPME) and electron radiation model (AE9) do not overlap in energy at this time—they have a boundary at 40 keV, whereas the proton models (SPMH and AP9) overlap from 100 keV to 164 keV. Without overlapping energies, it is not possible to gradually stitch the electron plasma and radiation models. A critical step in enabling that stitching is the creation of electron plasma templates that extend the SPME map to 100 keV. Ideally, the modules have substantial energy overlap, allowing sensor data to bridge the stitching gap and ensuring that the high energy of the lower module and the low energy of the upper module match well before stitching.

Finally, two performance improvements are available. One performance improvement can be achieved if the kernels are augmented with a caching capability so that if the same particle spectrum is fed in a second time, the result from the previous evaluation is returned. This will potentially save many matrix-vector multiplications, when multiple different output requests are using the same kernel (e.g., as inputs to different worst-case running averages of internal charging with different time constants). Another performance improvement can be achieved if the stitching weights are computed and stored before the full spatial interpolation weights are computed. This may save calculation of some magnetic coordinates for modules that have zero stitching weight.

6. Conclusions

We have described upgrades to the turnkey and runtime algorithms for the AE9/AP9-IRENE radiation and plasma climatology model to support the new module architecture. The new architecture represents the environment as a set of single-grid modules that are stitched together at runtime. To enable this change in architecture, new algorithms were needed in the turnkey system to properly capture the correlation in model uncertainty across modules—specifically, uncertainty associated with the addition of future datasets. The runtime algorithms evolved substantially, including not only the runtime stitching of modules at energy and spatial boundaries, but also a new approach to how users (or application programmers) request output and how fluxes are transformed from the module spectrum to the user-specified spectra or effects. These latter changes offer improved efficiency through the re-use of expensive calculations.

The primary work that remains on the module architecture upgrade is implementation of the Monte Carlo dynamic capability. It is already clear that some kind of shared state will be needed to control correlated temporal dynamics between modules. What is not clear is whether the plasma modules, which do not currently have Monte Carlo capability, should somehow be included. Another important question is whether all modules in a Monte Carlo group must evolve at the same time scale, which is important since plasmas evolve on much shorter time scales than radiation belts. In addition to these questions about the Monte Carlo implementation, further opportunities for optimization are available in the module architecture.

While the module architecture was initially envisioned to address anomalies in the current AE9/AP9-IRENE architecture, it also eases extensibility. Once fully implemented, the module architecture should make it relatively straightforward to incorporate dynamic solar protons, heavy ions, and new plasma populations, such as high-latitude aurora and the plasma sheet. These extensions of AE9/AP9-IRENE will expand the set of user problems that can be addressed by the model, and enable a consistent approach to radiation and plasma environment specification across more hazards.

7. References

[1] Eaton, J. W. et al., (2019). GNU Octave version 5.1.0 manual: a high-level interactive language for numerical computations. Retrieved from www.gnu.org/software/octave/doc/v5.1.0

[2] Ginet, G. P. et al., "AE9, AP9 and SPM: New models for specifying the trapped energetic particle and space plasma environment," *Space Sci. Rev., 179*, 579-615 (2013).

[3] Johnston, W. R. et al., "Recent updates to the AE9/AP9/SPM radiation belt and space plasma specification model," *IEEE Trans. Nucl. Sci., 62*(6):2760-2766 (2015).

[4] O'Brien, T. P. *AE9/AP9/SPM V1.0 Runtime Algorithms*, Aerospace Report No. TOR-2014-00294, The Aerospace Corporation, El Segundo, CA, December 30, 2013.

[5] O'Brien, T. P., and B. P. Kwan, *Using pre-computed kernels to accelerate effects calculations for AE9/AP9: A displacement damage example*, Aerospace Report No. TOR-2013-00529, The Aerospace Corporation, El Segundo, CA, 2013.

[6] O'Brien, T. P. *Generation of AE9/AP9/SPM V1.0 Runtime Tables*, Aerospace Report No. TOR-2014-00295, The Aerospace Corporation, El Segundo, CA, December 30, 2013.

[7] O'Brien, T. P. *Algorithms for Parallelizing AE9/AP9*, Aerospace Report No. TOR-2014-00361, The Aerospace Corporation, El Segundo, CA, December 30, 2013.

[8] O'Brien, T. P., and P. Whelan, *Specification for radiation effects kernels for use with AE9/AP9*, Aerospace Report No. ATR-2015-02436, The Aerospace Corporation, El Segundo, CA, 2015.

[9] O'Brien, T. P. et al., "Changes in AE9/AP9-IRENE version 1.5," *IEEE Trans. Nucl. Sci., 65*(1):462-466, doi:10.1109/TNS.2017.2771324 (2018).

Appendix A. Analytical Expressions for Energy Integrals

The expression below can be used to convert from a table of differential fluxes on an energy grid to user-requested differential, wide, or integral channels. First, we define the implied flux between the energy grid points using either semi-log interpolation or power-law interpolation:

$$j(E) = \begin{cases} 0 & j_i = j_{i+1} = 0 \quad \text{no flux} \\ j_i + (\ln E - \ln E_i)\frac{j_{i+1}-j_i}{\ln E_{i+1}-\ln E_i} & \text{only one of } j_i, j_{i+1} > 0 \quad \text{semi-log} \\ j_i (E/E_i)^{\frac{\ln(j_{i+1}/j_i)}{\ln(E_{i+1}/E_i)}} & \text{both } j_i, j_{i+1} > 0 \quad \text{power-law} \end{cases} \tag{a}$$

The definite integral flux from *A* to *B,* when $j_i \le A < B \le j_{i+1}$ is given by:

$$\int_A^B j(E)dE = \begin{cases} 0 & \text{no flux} \\ (B-A)j_i + \left(B(\ln B - 1) - A(\ln A - 1) - \ln E_i\,(B-A)\right)\frac{j_{i+1}-j_i}{\ln E_{i+1}-\ln E_i} & \text{semi-log} \\ j_i E_i^{-\frac{\ln(j_{i+1}/j_i)}{\ln(E_{i+1}/E_i)}} \frac{B^{\frac{\ln(j_{i+1}/j_i)}{\ln(E_{i+1}/E_i)}+1} - A^{\frac{\ln(j_{i+1}/j_i)}{\ln(E_{i+1}/E_i)}+1}}{\frac{\ln(j_{i+1}/j_i)}{\ln(E_{i+1}/E_i)}+1} & \text{power-law} \end{cases} \tag{b}$$

To compute a differential flux at an arbitrary energy *E'*, simply evaluate Eq. (a) in the appropriate interval: $E_i \le E' < E_{i+1}$. To compute an integral flux above *E',* use Eq. (b) with the following expression starting at the appropriate interval $E_k \le E' < E_{k+1}$:

$$J_>(E') = \int_{E'}^{\infty} j(E)dE = \int_{E'}^{E_{k+1}} j(E)dE + \sum_{i>k} \int_{E_i}^{E_{i+1}} j(E)dE \tag{c}$$

To compute a wide differential channel spanning from *E'* to *E''*, use Eq. (b) when *A* and *B* fall in the same interval between energy grid points. Otherwise, use the following expression, where $E_k \le E' < E_{k+1}$ and $E_l \le E'' < E_{l+1}$:

$$\int_{E'}^{E''} j(E)dE = \int_{E'}^{E_{k+1}} j(E)dE + \sum_{i>k}^{i\le l} \int_{E_i}^{E_{i+1}} j(E)dE + \int_{E_l}^{E''} j(E)dE \tag{d}$$

After computing the integral, the wide differential channel flux is normalized, dividing the integral by the channel energy bandwidth:

$$j_{\text{wide}}(E) = \frac{1}{E''-E'} \int_{E'}^{E''} j(E)dE \tag{e}$$

AEROSPACE REPORT NO.
ATR-2019-02679

AE9/AP9-IRENE Module Architecture (Static Environments)

Approved Electronically by:

Margaret W. Chen, ASSOC DIRECTOR
SPACE SCIENCES DEPARTMENT
SPACE SCIENCE APPLICATIONS LABORATORY

Cognizant Program Manager Approval:

Joseph E. Mazur, PRINC DIRECTOR
SPACE SCIENCE APPLICATIONS LABORATORY
PHYSICAL SCIENCES LABORATORIES

Aerospace Corporate Officer Approval:

Charles L. Gustafson, SR VP ENG & TECH
ENGINEERING & TECHNOLOGY GROUP

Content Concurrence Provided Electronically by:

T Paul P. O'Brien, SCIENTIST SR
MAGNETOSPHERIC & HELIOSPHERIC SCIENCES
SPACE SCIENCES DEPARTMENT

SY0473

AEROSPACE REPORT NO.
ATR-2019-02679

AE9/AP9-IRENE Module Architecture (Static Environments)

Office of General Counsel Approval Granted Electronically by:

Kien T. Le, ASST GEN COUNSEL
INTELLECTUAL PROPERTY
OFFICE OF GENERAL COUNSEL & SECRETARY

Export Control Office Approval Granted Electronically by:

Edward Pevzner, EXPORT CONTROL STAFF IV
GOVERNMENT SECURITY
SECURITY OPERATIONS
OFFICE OF THE CHIEF VELOCITY OFFICER

SY0473