1.4 Generation of AE9/AP9/SPM Runtime Tables

This section describes the algorithms employed to generate the AE9/AP9/SPM V1.0/V1.1 runtime data tables. These algorithms begin with time series in situ particle data that have already been cleaned and calibrated. The algorithms produce maps of the radiation and plasma environment as a function of magnetic field coordinates, matrices needed to perturb those maps to represent uncertainty, and matrices needed to evolve dynamic (Monte Carlo) scenario environments in time.

This section has also been released as Aerospace Report No. TOR-2014-00295. Matlab is a trademark of The Mathworks Inc.

1.4.1 Introduction

The AE9/AP9/SPM runtime tables are generated by what is known within the team as the "turnkey system." The turnkey system consists of a set of Matlab codes and scripts to run them. These codes progress through a series of stages in the processing from cleaned and calibrated time series data to the data tables used at run time by the AE9/AP9/SPM library and application. The turnkey system is run separately for each of the various models: AE9V10 (electron radiation), AP9V10 (proton radiation), SPMEV10 (electron plasma), SPMHV10 (proton plasma), SPMHEV10 (Helium plasma), SPMOV10 (Oxygen plasma).

The stages of the turnkey system are:

- **prepdata** – read in the data from each sensor and, if necessary, apply flags and break up into daily, sorted time series files
- **definebins** – define the coordinate bins to be used (e.g., *E*, *K*, Φ)
- **assign2bins** – bin the time series data from each sensor into coordinate bins
- **binavg** – compute time averages (e.g., 24 hours or 7 days) within bins
- **calctheta** – compute θ_1 and θ_2 (statistical moments/percentiles in each bin) from binavg results
- **fill_maps** – use templates to fill in gaps in theta maps for each sensor
- **combinetheta** – combine filled, sensor-specific theta maps into one consensus map with errors (Stheta)
- ***calccov covs*** – compute a set of spatial covariances at zero lag
- ***calccov lagcovs*** – compute a set of spatial covariances at various time lags
- ***makecov cov*** – use calccov results to fill in spatial covariance matrix
- ***makeQ*** – compute principal components
- ***makecov lagcov*** – use calccov and makeQ results to fill in spatial covariance matrices at various time lags
- **buildmc** – compute the final Monte Carlo scenario quantities, if needed, and save the runtime tables to a single file (e.g., AE9V10_runtime_tables.mat)

- **figs** – generate an extensive set of diagnostic figures

The turnkey process is depicted graphically in Figure 3. The items above in *italics* are not applicable to the plasma models which do not have Monte Carlo scenario capabilities. Each stage in the processing is controlled by a set of "global" variables that describe the model and set parameter values. Thus, it is possible to restart the entire processing chain with a single command, and it will run unattended through to completion. However, as the algorithms themselves were often being updated during the V1.0 development process, it is also possible to run only part of the turnkey process, starting and stopping at an arbitrary stage using partial results from a prior run.

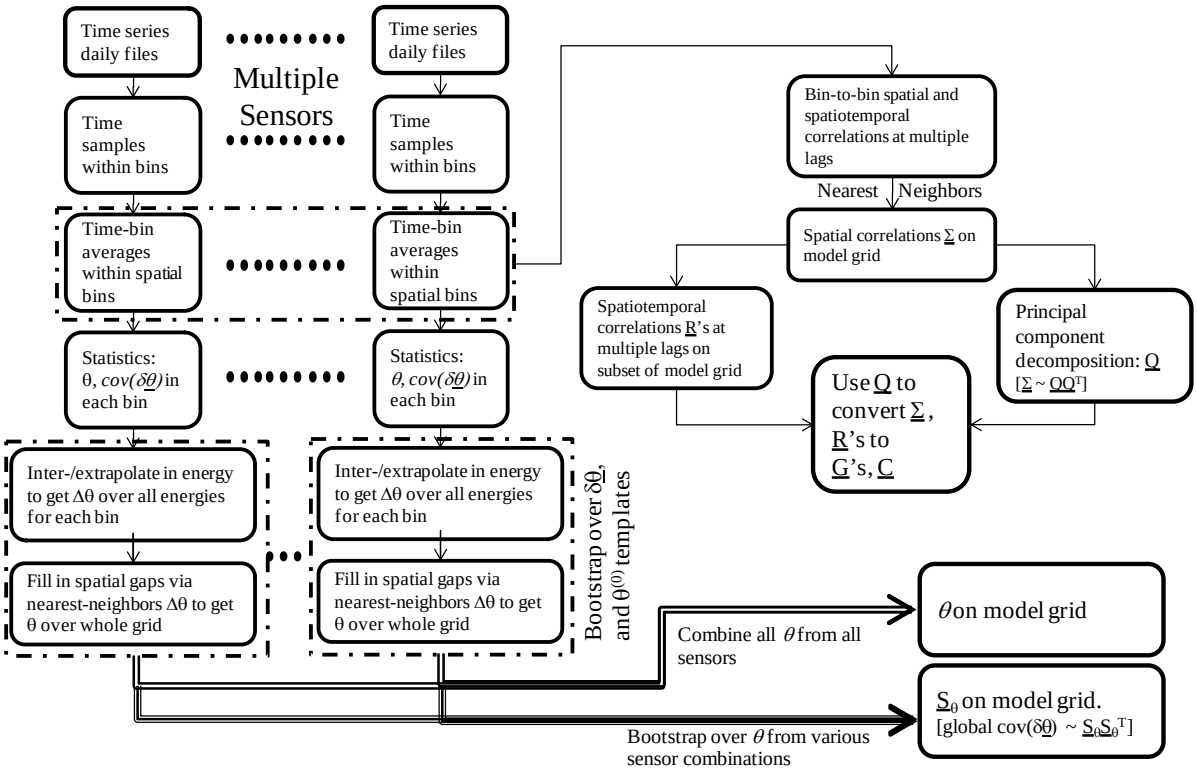

Figure 3. Graphical depiction of the "Turnkey System" that generates the runtime tables from cleaned and calibrated time series data.

It is important to note, as it will come into play later in this section, that most of the models assume (based on visual inspection) that the long-term statistical distribution of data within a given spatial bin has a log-normal distribution. However, for the energetic electrons in AE9, the shape is assumed to be Weibull. The properties of these distributions will be discussed in more detail in later sections.

1.4.2 Generating Flux Maps

A flux map is a tabular representation of the statistical properties of the flux in each coordinate bin. In AE9/AP9/SPM V1.0/V1.1, the flux map has two parameters at each location, which are expressed in terms of the median (m_{50}) and 95th percentile (m_{95}) flux within the bin:

$$\theta_1 = \ln m_{50} \quad (30)$$

$$\theta_2 = \ln(m_{95} - m_{50}) \quad (31)$$

This transformation from flux percentiles to θ guarantees that, for any real values of θ_1, θ_2, the fluxes will satisfy $m_{95} > m_{50} > 0$. Further, for $m_{95} >> m_{50}$, as is often the case, θ_2 becomes approximately the natural log of m_{95}. Thus, a Gaussian distribution of errors in θ translates to approximately log-normal errors in m_{50} and m_{95}. Log-normal errors are common in space particle fluxes, and so we assume the errors in θ are Gaussian. The errors in θ are given by $cov(\delta\theta)$, which is the local 2x2 matrix that represents the uncertainty in θ as measured by a single sensor on a single spacecraft. The entries in this matrix shrink with the square root of the sample size within the bin, and the off-diagonal term reflects the fact that errors in θ_1 and θ_2 are necessarily correlated. When we combine data across sensors, this $cov(\delta\theta)$ will feed into how we combine them and into the global error representation S_θ.

For each sensor, we generate a separate tabulation of θ and $cov(\delta\theta)$ on the spatial grid (a flux map). That process involves several steps.

1.4.2.1 *Loading and filtering (*prepdata*)*

The first step in generating a flux map is reading the time series data, filtering it and reorganizing as needed, and saving it into a file with a standard structure. The filtering typically involves removing times flagged for possibly having contamination (e.g., from solar energetic particles), and possibly applying *ad hoc* filters. *Ad hoc* filters originated because it was technically less cumbersome to include a few last-minute filters to the data at the start of the turnkey process rather than to continually generate new versions of the turnkey input data files. Reorganizing the data involves unwrapping an angular axis, if one is present, so that the flux table has only two dimensions: time and energy. Further, variable name translation may be necessary, such as converting h_{min} to h_min, etc.

At the completion of the prepdata step, each sensor is organized into daily files of filtered, calibrated, and cleaned particle flux, as well as associated coordinate and error information [*Ginet et al.*, 2013; *Guild et al.*, 2009; *O'Brien*, 2012b]. The flux errors are provided in terms of the standard deviation of the natural log of flux, either using an ad hoc value or as a result of an inversion and/or cross-calibration.

1.4.2.2 *Defining the bins (*definebins*)*

Before we can bin the data, we must define the bins, specifically, the bins in the 2nd and 3rd coordinates (such as *K*, Φ, and other described in *Ginet et al.* [2013]). The data are not binned in energy—rather they are handled at their original energy channels provided to the turnkey system and will be interpolated onto the model energy grid at a later stage. The coordinate grid itself defines the nominal spatial bins. However, because the coordinate grid is rectangular, it is often the case that a large portion (half) of the nominal grid falls inside the loss cone. Therefore, the definebins step not only describes the bin limits for each spatial bin, it also applies a loss cone filter to remove bins that fall in the loss cone.

For example, the AP9V10 high altitude grid (K-Φ) uses a loss cone defined by the maximum value of K as a function of Φ. This maximum value is derived from the largest K at a given Φ that corresponds to an h_{min} of 200 km (that is, the K for which the particle's drift-bounce orbit eventually intersects the atmosphere at 200 km geodetic altitude). The maximum K thus defined is given by a polynomial in Φ (rounded off):

$$\mathrm{Log}_{10} K = -21.8\ 135.0\Phi^7 + 135.0\Phi^6 - 342.0\Phi^5 +$$

$$458.0\Phi^4 - 348.0\Phi^3 + 150.8\Phi^2 - 36.1\Phi + 4.7 \qquad (32)$$

for K in $G^{1/2}R_E$ and Φ in $G\ R_E^2$.

Bins whose center falls in the loss cone are removed from further consideration. Bins that partially overlap the loss cone are limited such that the bin edges in the second coordinate (e.g., K) are entirely outside the loss cone. Equivalently, bins whose upper or lower edges fall outside the valid range for one of the coordinates will have the edge adjusted to be at the limit of the valid range (e.g., K<0 is replaced by K=0).

A bin consists of a lower, middle, and upper value for each spatial coordinate, and the lower and upper values typically split the difference between adjacent bin centers. The "full" grid of bins spans the full range of the second and third coordinates, and is thus referred to as the full Q2Q3 grid. It can be referenced either by the row/column type subscripts or by a 1-D index that unwraps the 2-D grid onto a 1-D list. The "reduced" grid is only referenced by a 1-D index, which spans only the N_{red} bins that are not in the loss cone. Tables ired2full and ifull2red are used to convert 1-D indices between the two grids. Matlab's sub2ind and ind2sub routines are used to convert between 2-D subscripts and 1-D indices. In a table with N_1 rows and N_2 columns the coordinate (i_1,i_2) maps to a 1-D coordinate j according to:

$$j = N_1(i_2 - 1) + i_1. \qquad (33)$$

That is, Matlab uses a 1-based, column-major indexing system, and therefore, so does AE9/AP9/SPM.

An additional consideration is that AE9 and AP9 support two different grids, identified in the code as subgrids. The calculations described in this section are applied separately to the subgrids and combined only at the very end of generating the runtime tables.

*1.4.2.3 Binning (*assign2bins*)*

The assign2bins step is fairly straightforward: each spectrum from a given sensor at the original time resolution is assigned to a bin in the two spatial coordinates. The result is a set of files, each one containing all the time samples from a specific sensor in a specific bin. The files are identified by the model, sensor, subgrid, and the reduced grid index for the bin.

For the AE9 and AP9 models, there are two subgrids, so the assign2bins step is done twice, once for each subgrid. Because the subgrids overlap, some time samples may appear once in each

subgrid. This is allowed because it is believed to help ensure a smooth interface of the two grids. It does not lead to over-counting, since at runtime the grids are spliced, not summed.

1.4.2.4 Averaging (binavg)

Time averaging is used to suppress (and, sometimes assess) random measurement errors, and to put the data from different sources on a consistent time index.

First, the data are separated into "passes" within a bin. A pass is defined as any set of time samples in the bin that have no temporal gaps longer than 15 minutes. This definition allows for vehicles that physically move in and out of a bin, but also for spinning sensors where the field of view moves in and out of the bin. Successive spins would usually count as a single pass through the bin. Data within each pass are combined using a weighted flux average.

The weighted flux average is a bit complicated because it combines two different approaches to computing the error in the average. If the raw flux data is marked as *x* with error *dx* (which is the standard error of the natural log flux, sometimes denoted *dlnj* or *dlogflux*) and the average is given as *y* with error *dy*, then the weighted flux average

$$y = \frac{\sum_{i=1}^{N} \frac{x_i}{dx^2}}{\sum_{i=1}^{N} dx^{-2}} \tag{34}$$

$$dy = \max\left(\sqrt{\frac{\sum_{i=1}^{N} \frac{x_i^2}{dx^2}}{y^2 \sum_{i=1}^{N} dx^{-2}} - 1, \left(\frac{1}{N} \sum_{i=1}^{N} dx^{-2} \right)^{-1/2} } \right) \tag{35}$$

The resulting *dy* is further limited to be between 0.1 and ln(10), which is to say that the resulting uncertainty is constrained to be between 10% and a factor of 10. Note that we do not apply a square-root-N factor to shrink *dy*. We experimented with this, but determined that there was sufficient correlation within a single pass that the square-root-N was not justified.

After the pass averaging, we then proceed to average all the passes within time bins, using the same formulas used for pass averaging. For AP9 the time bin size is 1 week; for all other models, it is one day.

1.4.2.5 Computing theta (calctheta)

We require at least 50 time-averages for a given energy channel for a given sensor in a bin to proceed with the calculation of θ. Otherwise, the bin is designated as empty for the particular sensor and channel (we will fill in these empty bins by interpolation and extrapolation in a later step).

In the alpha and beta versions of AE9/AP9, the calculation of theta (θ) was fairly straightforward – the median and 95th percentiles were calculated directly from the averaged samples within bins. That approach guarantees good agreement between the model and the data at the 50th and 95th

percentiles. However, a desire to guarantee good agreement with the *mean* and the 95th percentile motivated a change to a more indirect approach.

The approach used in V1.0/V1.1 fits θ_1 and θ_2 using one of three approaches depending on the model. In all models, the θ fit must match the observed 95th percentile in the bin (m_{95}). For the plasma model, the θ fit must also match the observed mean $<x>$ in the bin. For AE9 and AP9, the θ fit must match ($<x>_{95}$) the mean of the flux up to the 95th percentile. We chose $<x>_{95}$ because, in the radiation belt data sets, we are still concerned that the tail of the distribution might be affected by instrumental effects. To do the fitting, we need several new relationships. For the log Normal, we have:

$$P[X < x] = \Phi\left(\frac{x-\mu}{\sigma}\right) \tag{36}$$

$$\Phi(z) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{z} e^{-t^2/2}\, dt \tag{37}$$

$$< x > = \exp(\mu + \frac{\sigma^2}{2}) \tag{38}$$

$$\mu = \ln m_{50} \tag{39}$$

$$\sigma = \frac{\ln {m_{95}}/{m_{50}}}{\Phi^{-1}(0.95)} = \frac{\ln m_{95} - \mu}{\Phi^{-1}(0.95)} \tag{40}$$

$$< x >_{95} = \frac{\sum_{x_i < m_{95}} x_i}{\sum_{x_i < m_{95}} 1} = \frac{\exp(C)\Phi(v_{95})}{0.95} \tag{41}$$

$$C = \frac{1}{2}\left(a^2 - \left({\mu}/{\sigma}\right)^2\right) \tag{42}$$

$$a = \sigma + {\mu}/{\sigma} \tag{43}$$

$$v_{95} = \ln m_{95} - a \tag{44}$$

Here $\Phi()$ is the cumulative unit normal (standard Gaussian) distribution, with mean 0 and standard deviation 1, and $\Phi^{-1}()$ is its inverse. We begin with the observed m_{95} and an guess for $\mu = \ln m_{50}$, and perform a 1-D optimization with respect to μ, fitting either $<x>$ or $<x>_{95}$, depending on the model (AP9 or a plasma model). With μ and m_{95}, we can obtain θ_1 and θ_2 directly using (39) then (30) and (31). We will later need to transform from flux x to a unit normal z using this relation:

$$z = \frac{\ln x - \mu}{\sigma} \tag{45}$$

For AE9, we are working with a Weibull distribution, and we use a different set of formulas:

$$P[X < x] = 1 - \exp\left(-\left(\frac{x}{\sigma}\right)^{\gamma}\right) \tag{46}$$

$$< x > = \sigma\Gamma(1 + 1/\gamma) \quad (47)$$

$$\Gamma(z) = \int_0^\infty t^{z-1}\, e^{-t} dt \quad (48)$$

$$\gamma = \frac{\ln\left(\ln 0.05 / \ln 0.5\right)}{\ln m_{95}/m_{50}} \quad (49)$$

$$\sigma = \frac{m_{50}}{(\ln 2)^{1/\gamma}} = \frac{m_{95}}{(\ln 20)^{1/\gamma}} \quad (50)$$

$$< x >_{95} = \frac{\sum_{x_i < m_{95}} x_i}{\sum_{x_i < m_{95}} 1} = \sigma\gamma^*\left(\ln 20\,, 1 + \frac{1}{\gamma}\right)/0.95 \quad (51)$$

$$\gamma^*(z, v) = \int_0^v t^{z-1}\, e^{-t} dt \quad (52)$$

Here Γ() is the gamma function, and $\gamma^*()$ is the incomplete gamma function (this notation varies slightly from some conventions because we needed to distinguish between the gamma function and the statistical parameter γ). Given m_{95}, we can compute σ using (50). We then perform a 1-D search for γ in the range 0.01 to 10 so as to reproduce $<x>_{95}$ according to (51). With γ and m_{95}, we can obtain θ_1 and θ_2 directly using Eq. (49) then Eqs. (30) and (31). For the Weibull, the conversion between x and z is:

$$\Phi(z) = 1 - \exp\left(-\left(\frac{x}{\sigma}\right)^{\gamma}\right) \quad (53)$$

Whichever statistical distribution we use, we compute θ for every sensor and every energy channel in every spatial bin using the entire data set. We then compute an error covariance for θ by bootstrapping over resamples (with replacement) from the data within the bin. For each time sample in a bootstrap, we perturb the flux using a log Normal error distribution consistent with the error computed in the binavg step (*dy*, the standard error for the natural log of the averaged flux). We bootstrap 200 times, with different error perturbations for the selected time averages in each bootstrap. From these 200 bootstraps, we compute one estimate of θ. We can then compute the 2x2 error covariance for θ using these 200 bootstrap estimates. We denote this error covariance $cov(\delta\theta)$ as described in the introduction to this section. We will use θ and $cov(\delta\theta)$ in the next step, when we fill in and combine the flux maps.

*1.4.2.6 Filling in each sensor with each template (*fill_maps*)*

We fill in the gaps in the flux map from each sensor using a set of masks and templates. A mask is simply an arbitrary set of points that are manually selected for removal. Masks affect a very small number of points that did not pass visual inspection of the resulting flux maps from a prior build of the model. Masks are only used for SPMHE (Helium plasma) and SPMO (Oxygen plasma).

A template is a global specification of the shape of the radiation belt as a function of the model coordinates (e.g., *E*, *K*, Φ). Templates are generated by manually stitching together select data sets, models, and ad hoc extrapolations. The templates represent human input into the system,

such as specifying how the spectrum should fall off past the last energy channel. Because humans may be uncertain, or different humans may disagree, there are many templates for each subgrid within each model. The templates do not specify an absolute flux level, but merely how flux varies across the model domain. The actual templates used in AE9/AP9/SPM are described in Chapter 2 and *O'Brien* [2013]. The templates themselves are used only in developing the runtime tables; they are not used at runtime, and so are not distributed with the model. A list of templates used for each model is provided in Appendix D.

Templates are stored on disk in terms of the common log of flux (arbitrary units), but are used in the model as natural log flux (for compatibility with θ). The conversion is performed on load. In this report, we will denote an arbitrary template, in natural log flux, as $\theta^{(0)}$. Since there are many templates for any given subgrid, we denote a particular template as $\theta^{(0,k)}$.

Every template will be used to fill in the entire flux map for every sensor for several instantiations of errors (10 bootstraps) in the original sensor θ, represented by local $cov(\delta\theta)$.

First, at each grid point, we perturb the local 2-element θ using a random 2-element unit-normal perturbation η, processed through a conditioning matrix $\underline{s}$. The conditioning matrix is nominally the square root of the local $cov(\delta\theta)$, except that the $cov(\delta\theta)$ is adjusted so that the variance (diagonal elements) are never less than 0.1^2, that is the local error in θ is never allowed to be less than 10%. The perturbed θ is denoted θ'.

$$\vec{\theta}' = \vec{\theta} + \underline{s}\vec{\eta} \tag{54}$$

$$\underline{\underline{ss}}^T = \mathrm{cov}(\delta\theta) \tag{55}$$

This process is repeated at every grid point with a unique perturbation.

From this point forward, the filling in is done separately for θ_1 and θ_2, and it is assumed that θ_2 has the same shape on the grid as a log flux would, even though it is not exactly a log flux.

The next step is to perform a log-log interpolation from the sensor channel energies in θ' to the grid energies at each spatial grid point (e.g., for each K,Φ pair). This applies only to energy grid points that are at or between the lowest and highest sensor channel energies. The low and high energy extrapolations are handled separately, by adjusting either tail of the template spectrum to match the lowest/highest interpolated energy grid point. When only one energy channel is present, the template spectrum at the local grid point is log-log interpolated onto that one channel, and then entire template spectrum is adjusted up or down to match that interpolated value. In either case, the template shape is mainly used only for extrapolating the spectrum. At this point, at each spatial grid point we either have no data or we have a complete spectrum (for the sensor we are working with). We will call this partially filled map θ''.

Spatial interpolation and extrapolation (e.g., in K,Φ) is considerably more complicated. First, we subtract the template we are using from the θ'' to create a map of deviations:

$$\Delta\theta = \theta'' - \theta^{(0)} \tag{56}$$

Then, we step through all the spatial grid points to find cases where there is no data from the sensor we are working with. At those "gaps" we compute a composite $\Delta\theta$', which is the average of nearby $\Delta\theta$. "Nearby" is defined in a Pythagorean sense, but on a transformed coordinate grid. Namely, grid coordinates (which are often transformed from the physical units, e.g., Φ is actually log Φ on the grid) are rescaled to cover the range from 0 to 1. The number of nearest neighbors used is scaled to $1/7^{th}$ the number of points on the reduced grid; this scale factor was arrived at through trial and error against human judgment of what was adequately smooth. The final, filled in map, is obtained by adding the template back to the composite $\Delta\theta$'.

$$\theta''' = \Delta\theta' + \theta^{(0)} \qquad (57)$$

This nearest-neighbors process ensures that there are no artificial spatial features added in at the interfaces between original and gap-filled points. The spatial filling in is depicted graphically in Figure 4.

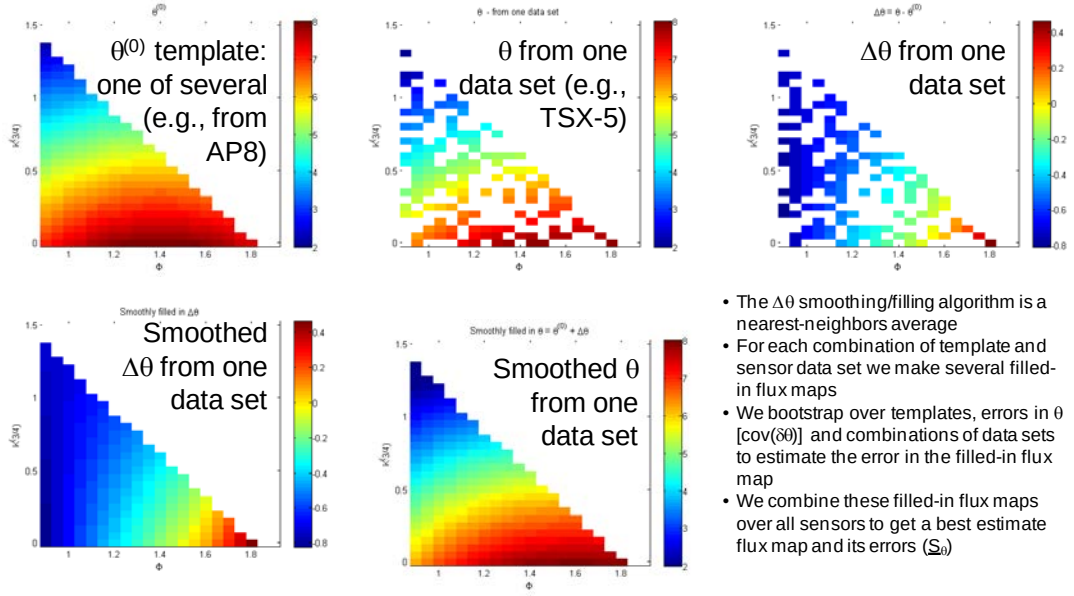

Figure 4. A graphical depiction of the spatial filling technique.

At this stage, for each sensor and for each template, we have many (10 bootstraps) completely filled in flux maps θ'''. We will next combine these together to obtain a global best estimate flux map and to obtain the new local error in that filled flux map for each sensor. The best estimate $\underline{\theta}$ for the sensor is obtained by computing an average over all templates and all bootstraps of θ'''. The local error, again denoted $cov(\delta\theta)$, is computed as the covariance over all templates and all bootstraps of θ'''. Now we have a filled in flux map and its local error for each sensor.

1.4.2.7 Combining filled-in maps and estimating global errors (combinetheta)

The many filled sensor-specific flux maps will be averaged together to obtain a best estimate of θ and its global error.

The best estimate of θ is obtained by a weighted average of the individual sensor-specific $\underline{\theta}$. The weighting is given by the reciprocal of the diagonal elements of each local $cov(\delta\theta)$ (i.e., the weighting is inverse square of the local standard error). Thus, locally, sensors with large error (due either to measurement error, poor sampling, or lots of interpolation/extrapolation uncertainty) are weighted less than those with small error.

This combination process allows some roughness, due to possible roughness in $cov(\delta\theta)$, and so an ad hoc smoothing is applied. The smoothing is defined by a 3-D smoothing mask. Smoothing parameters are given in Appendix E.

Next, for AE9 and AP9 only, we "stitch" the K-h_{min} grid and the K-Φ grid together. This stitching process finds the grid interface at h_{min} = 1000 km, and adjusts the θ at all h_{min} for a given E,K to match the ratio of the θ value at 1000 km between the two grids. Because the relationship between Φ and h_{min} is epoch dependent, the stitching is done based on the epoch 1-Jan-2010 00:00:00 UTC. The adjustments are linear in θ, which means they are effectively multiplicative in flux. For K-h_{min} points where the K-Φ value does not exist (i.e., it is outside the grid), the corresponding K-h_{min} points are flagged for removal. (This stitching relegates the data used to populate the K-h_{min} grid to representing only the h_{min} gradients, and it will have to be replaced with something better in future versions.)

Next we determine the "activepoints" filter. For the plasma model, this filter simply removes any points for which there remains a bad data flag after all the filling and smoothing. For the radiation models, this filter also removes points whose median flux is less than an ad hoc factor (10^9 for AE9, 10^{10} for AP9) less than the maximum median flux anywhere on the grid. These factors were estimated from corresponding values in AE8 and AP8. This filter is applied because some of the energy extrapolations lead to fluxes that are so small as to be neglected (or, simply, not credible given the limitations of the measurements). The activepoints filter maps between the N_E*N_{red} set of all q and the Nact $\leq N_E$*N_{red} domain. Arrays active2all and all2active translate between these domains in the same ways as their counterparts for the reduced grid.

Now we have our final theta (θ) for use in the runtime tables. The version of theta stored in the runtime tables is concatenated such that all elements of a subgrid are "unwrapped" to produce an N_{act}x2 matrix, and then each subgrid is stacked to produce a new, larger matrix, still with 2 columns.

To compute S_θ (Stheta), which represents the global errors in θ, we repeat the entire process of building the flux map 50 times, but each time we resample the set of sensors used to compute the flux map and we perturb each $\underline{\theta}$ using its local $cov(\delta\theta)$. This bootstrapping represents different combinations of sensor data sets, which captures the uncertainty that the model has with regard to adding new sensor data, and it represents the error in each data set when combined with the

others. We recognize that some sensors should be grouped together because they have the same host vehicle or are very similar in orbit, design, or calibration. The bootstrapping keeps those groups together. For example, CRRES MEA and HEEF should be kept together, as should all the LANL-GEO data. Initially we concatenate the multiple estimates of θ into a large (N_{act} x2)x50 matrix Θ. We have to unwrap the individual N_{act}x2 θ matrices by stacking the two original columns into a single column of Θ. The initial estimate of S_θ is then given by:

$$\underline{\underline{S}}_\theta = \frac{1}{\sqrt{49}}\left(\underline{\underline{\Theta}} - \underline{\underline{\bar{\Theta}}}\right) \tag{58}$$

Here $\underline{\underline{\bar{\Theta}}}$ is a matrix whose columns are the average of the rows of Θ. This relation gives

$$\underline{\underline{S}}_\theta \underline{\underline{S}}_\theta^T = \frac{1}{49}\left(\underline{\underline{\Theta}} - \underline{\underline{\bar{\Theta}}}\right)\left(\underline{\underline{\Theta}} - \underline{\underline{\bar{\Theta}}}\right)^{\mathrm{T}} \tag{59}$$

which means that S_θ is the square root of the error covariance of θ, as estimated from 50 bootstrap samples. (The runtime algorithms will use this matrix to generate perturbations on θ in order to represent uncertainty in the flux maps. Those perturbations are generated from the square root of the error covariance matrix, just as we did with $\underline{s}$ in (54)-(57).)

Next we "clip" this initial S_θ so that no entry is larger than $\ln(X)/2/\sqrt{49}$, where X = 100 for AE9 and X=10 for all the other models. This imposes a maximum uncertainty on any flux as having a 95% confidence interval of a factor of X.

Next we compress S_θ using singular value decomposition. Any real matrix can be decomposed into a column space (and its null space), a set of singular values, and a row space (and its null space):

$$\underline{\underline{S}}_\theta = \underline{\underline{USV}}^T \tag{60}$$

$$\underline{\underline{UU}}^T = \underline{\underline{VV}}^T = \underline{\underline{I}} \tag{61}$$

The singular values are the diagonal elements of the diagonal matrix S. We are actually interested in the properties of $\underline{\underline{S}}_\theta \underline{\underline{S}}_\theta^T$, which is:

$$\underline{\underline{S}}_\theta \underline{\underline{S}}_\theta^T = \underline{\underline{USS}}^T \underline{\underline{U}}^T \tag{62}$$

We can dispense with V entirely, and we can approximate $\underline{\underline{S}}_\theta \underline{\underline{S}}_\theta^T$ by retaining only a subset of the singular values in S. Namely, we retain enough diagonal elements of S to retain 90% of the variance (the sum of the squares of the diagonals of S). If the original S_θ is N_Sx50, and we retain only $N_{S'}$ singular values, then S' is N_S x $N_{S'}$, and so is S_θ':

$$\underline{\underline{S'}}_\theta = \underline{\underline{US'}} \tag{63}$$

Typically, this compression requires only about 10 columns in S_θ' rather than 50 in S_θ. The compression saves substantially (~80%) on disk space and memory because S_θ is by far the largest item used by the runtime tables.

Finally, we clip' S_θ' again, using the same value of X as above to ensure that the compression did not introduce any new, very large errors.

This completes the calculation of the flux map θ and its global error $S\theta$ for the runtime tables. Figure 5 shows several diagnostics of the process described above.

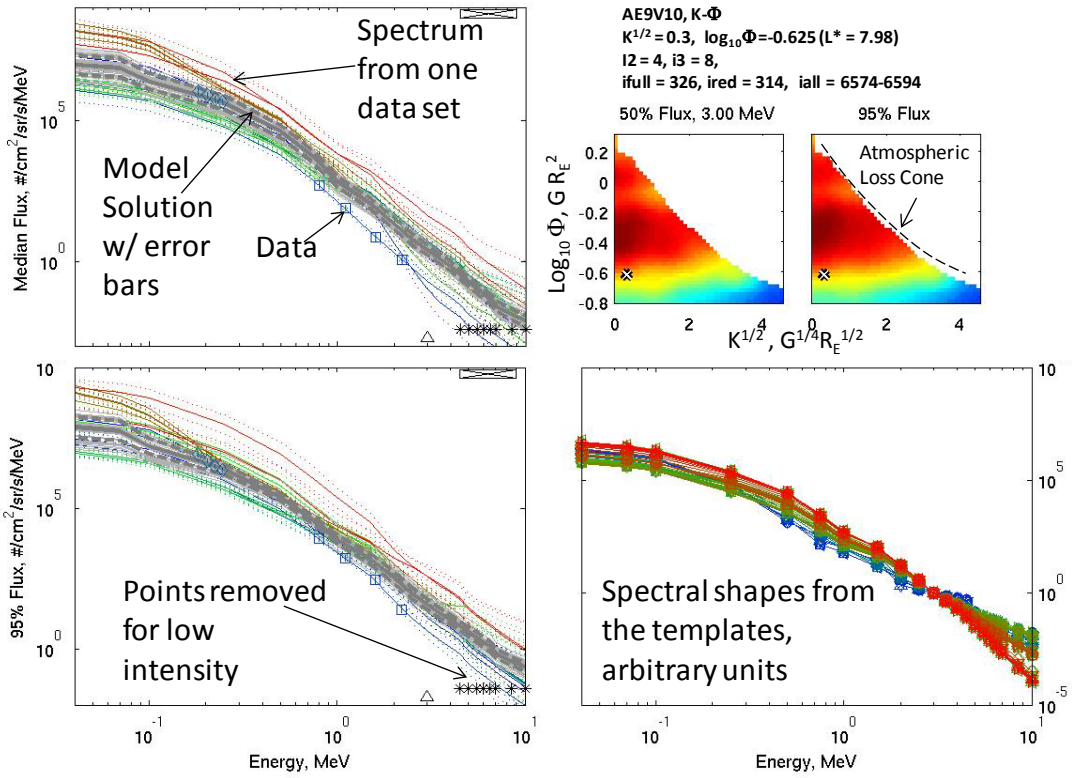

Figure 5. A "binspectra" figure for AE9V1.0 on the *K*-Φ grid. It shows the median and 95th percentile flux (left) for every sensor data set (colored thin curves) and for the model fit (thick gray curves) in the selected *K*-Φ bin. Error bars are shown as dotted lines. Original sensor data in the bin is shown with symbols. Energies that are removed in the bin for low flux are marked with black asterisks. On the top right is a set of bin identification information as well as color contours of the $\log_{10}$ median and 95th percentile flux at the selected energy. The atmospheric loss cone is identified on the 95th percentile flux map. At bottom right are all the templates used, normalized to have a value of 1 at 3 MeV in this bin.

1.4.3 Time Evolution Matrices

The report *O'Brien* [2012] presents the equations that govern time evolution in AE9 and AP9 monte carlo scenarios. That evolution is controlled by a set of principal component amplitudes:

$$\vec{q}_t = \sum_{i=1}^{N_G} \underline{\underline{G}}_i \vec{q}_{t-\tau_i}^T + \underline{\underline{C}}\vec{\eta}_{t+\delta t} \quad (64)$$

There are N_G lag persistence factors, and at each time step there is a unit Gaussian white noise innovation η (a series of uncorrelated Gaussian random variables with mean zero and unit variance). The *G* and *C* matrices condition the persistence and innovation, respectively. The τ_i are prescribed integer multiples of the fundamental time step δt.

To derive *G* and *C* according to *O'Brien* [2012], we need to obtain from the data spatial covariance matrix across the entire grid, and spatiotemporal covariances (lag covariances) on a decimated grid. The time lags that we will need correspond to the unique set of time offsets between every time lag used in the time evolution equation. For example, an equation that involved states at times 1, 3, and 6 would require covariances at lags of 2 (= 3-1), 3 (=6-3), and 5 (= 6-1).

A (lag) covariance is defined as

$$\underline{\underline{\hat{R}}}_M = \langle \vec{z}_t \vec{z}_{t-M\delta t}^T \rangle = \underline{\underline{\hat{R}}}_{-M}^T \quad (65)$$

where *z* is a vector of normalized fluxes over an entire subgrid. The normalization is such that all the time averaged points within a bin are replaced according to:

$$z_i = \Phi^{-1}\left(\frac{\mathrm{i}}{\mathrm{N}+1}\right) \quad (66)$$

where *i* represents the order the point would have in a sorted list, such that z_1 replaces the smallest flux and z_N replaces the largest. Thus with a bin, the set of *z*'s has a Gaussian distribution with mean 0 and variance 1. The time offset is *M* time steps of size δt, and the angle brackets represent an average, effectively an average over time. For *M*=0, we have the spatial covariance matrix Σ:

$$\underline{\underline{\hat{R}}}_0 = \langle \vec{z}_t \vec{z}_t^T \rangle = \underline{\underline{\Sigma}} = \underline{\underline{\mathrm{QQ}^{\mathrm{T}}}}. \quad (67)$$

The matrix *Q* defines the relationship between the principal component amplitudes *q*'s and the *z*'s:

$$\vec{z}_t = \underline{\underline{\mathrm{Q}}}\vec{q}_t. \quad (68)$$

Like the *z*'s, the *q*'s are Gaussians with zero mean and unit variance.

*1.4.3.1 Computing spatial (*calccov covs*) and lag correlations (*calccov lagcovs*)*

We first build a database of covariances at various lags. We do this by repeatedly selecting two spatial bins at random from two randomly selected sensors, and a random energy channel for each sensor. We then find the intersection of the binavg data in the first bin with binavg data in the second bin at the appropriate lag. If there are enough intersecting points (100), we compute a

covariance between the z's for two data sets and store it in a database for that lag. We repeat this until we have $3x10^5$ points for the spatial covariance and 10^4 points for each lag covariance (we will see below why we need far less lag covariance information because of the principal component transform).

For spatial covariance (lag zero), we also do this across subgrids, where we select one point from the primary subgrid (K-Φ) and one from the subordinate subgrid (K-h_{min}).

*1.4.3.2 Filling in the spatial covariance matrix (*makecov cov*)*

With on the order of $10^4 - 10^5$ grid points per subgrid, the full covariance matrix Σ is huge. Further, as we will see below, it contains a lot of "noise" correlations that we will not retain anyway, and we do not have enough simultaneous data to actually constrain such a huge covariance matrix. Therefore, to keep compute times to a reasonable level, we decimate the spatial subgrids for computing the spatial covariance matrix. This has the effect of reducing the fine detail of the spatial correlation structure, but that is the part we believe is least credible given the limitations of our observations. We decimate the subgrid iteratively over the 3 dimensions, increasing the decimation factor from 1 (meaning keep all points in that dimension) to 2 (meaning skipping every other point), etc. The dimension that is decimated is whichever is longest given the decimation up to that point. We iterate this process until the implied covariance matrix has less than about 200 million entries. Typically the energy dimension is not decimated, while each spatial dimension is decimated by a factor of 2. Special care is taken to retain the end points of decimated dimensions.

Next we obtain all the covariances from the lag-zero database. We are going to use a nearest-neighbors averaging to populate the covariance matrix on the decimated grid. First, we rescale their coordinates of the points in the database. The energy dimension is scaled first with a logarithm, and then linearly scaled to span 0 to 1. The spatial dimensions are simply linearly scaled to span 0 to 1 (although this already includes the coordinate transforms inherent in the grid, such as taking the logarithm of Φ).

If we are working with the primary grid, then we are creating a symmetric, square matrix, and we can save calculations by only doing the diagonal and one triangle, and copying the results to the other triangle.

To compute a point in one of the covariance matrices, we average the 100 nearest neighbors of that point's grid coordinates, with the neighbors selected from the database for that lag. The set of nearest neighbors is determined by Pythagorean distance in the rescaled coordinates domain (a hypercube from 0 to 1 in 6 dimensions).

It is worthwhile to note how the subgrids relate to each other. If we have 2 subgrids, the covariance matrix Σ can be represented in block matrix notation as:

$$\underline{\underline{\Sigma}} = \begin{bmatrix} \underline{\underline{\Sigma}}_{11} & \underline{\underline{\Sigma}}_{12} \\ \underline{\underline{\Sigma}}_{12}^{T} & \underline{\underline{\Sigma}}_{22} \end{bmatrix} \tag{69}$$

where Σ_{11} is the spatial covariance matrix on subgrid 1, Σ_{22} is the spatial covariance matrix on subgrid 2, and Σ_{12} is the spatial covariance matrix between subgrids 1 and 2. We will see below that we only need the column for the primary grid, which in the case of AE9 and AP9 is subgrid 2 (K-Φ). Thus, we only need Σ_{12} and Σ_{22}.

1.4.3.3 Computing principal components (makeQ)

The principal components that drive the Monte Carlo scenarios are related to the spatial covariance according to Q (suppressing the subgrid subscripts for the moment):

We determine Q via eigenvalue decomposition of Σ:

$$\underline{\underline{\Sigma}} = \underline{\underline{V\Lambda V}}^{\mathrm{T}} \tag{70}$$

$$\underline{\underline{VV}}^{\mathrm{T}} = \underline{\underline{I}} \tag{71}$$

where the eigenvalues are the diagonal elements of the diagonal matrix Λ. We only retain enough entries in Λ such that QQ^T represents most of the variance in Σ. The decision on how many entries in Λ to retain is based on two criteria: we remove all entries in Λ that represent less than 1% of the variance in Σ. Because of numerical limitations, we also ensure that we stop retaining entries in Λ after we have stored more variance than exists in Σ (this happens because sometimes Λ has negative entries, which are numerical noise). We compute a preliminary Q from the truncated Λ' as:

$$\underline{\underline{\mathrm{Q}}} = \underline{\underline{\mathrm{V}}}\sqrt{\underline{\underline{\Lambda'}}}. \tag{72}$$

While Q has as many rows as there are grid points, it typically only has about 10 columns – thus all the temporal variation is controlled by changing amplitudes about 10 principal components q of spatial variation. The columns of Q are the principal components of spatial variation on the decimated (primary sub)grid.

The expressions for Q above give the principal components on the decimated primary subgrid. To obtain Q for a decimated subordinate grid, we have to relate Q to Σ_{12} via Σ_{22} (when the primary subgrid is number 2). Specifically:

$$\underline{\underline{\Sigma}} = \begin{bmatrix} \underline{\underline{\mathrm{Q}}}_{11} \\ \underline{\underline{\mathrm{Q}}}_{22} \end{bmatrix} \begin{bmatrix} \underline{\underline{\mathrm{Q}}}_{11} \\ \underline{\underline{\mathrm{Q}}}_{22} \end{bmatrix}^{T} = \begin{bmatrix} \underline{\underline{\mathrm{Q}}}_{11}\underline{\underline{\mathrm{Q}}}_{11}^{\mathrm{T}} & \underline{\underline{\mathrm{Q}}}_{11}\underline{\underline{\mathrm{Q}}}_{22}^{\mathrm{T}} \\ \underline{\underline{\mathrm{Q}}}_{22}\underline{\underline{\mathrm{Q}}}_{11}^{\mathrm{T}} & \underline{\underline{\mathrm{Q}}}_{22}\underline{\underline{\mathrm{Q}}}_{22}^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} \underline{\underline{\Sigma}}_{11} & \underline{\underline{\Sigma}}_{12} \\ \underline{\underline{\Sigma}}_{12}^{T} & \underline{\underline{\Sigma}}_{22} \end{bmatrix} \tag{73}$$

Thus:

$$\underline{\underline{\mathrm{Q}}}_{11}\underline{\underline{\mathrm{Q}}}_{22}^{\mathrm{T}} = \underline{\underline{\Sigma}}_{12} \tag{74}$$

which means

$$\underline{\underline{Q}}_{11} = \underline{\underline{\Sigma}}_{12}\underline{\underline{Q}}_{22}^{-T} \quad (75)$$

The matrix Q_{22}^{-T} is computed as the transpose of the pseudoinvese of Q_{22} (again, via singular value decomposition). This approach allows a single set of principal components on the primary subgrid to drive temporal variation on both subgrids via Q_{11} and Q_{22}.

Next we interpolate the Q's from the decimated subgrids to the full subgrids. We then rescale the rows of Q for each subgrid such that each has a sum-of-squares that is 1 – this is important, as it guarantees the validity of the statistical transforms from q to z retain the property that z has zero mean and unit variance, which, in turn, guarantees that the flux x, when produced from z according to (45) or (53) has the right distribution.

The remainder of the operations can be computed only on the primary subgrid, since the fluxes on the subordinate subgrid can be computed from the same q used on the primary grid.

*1.4.3.4 Filling in lag correlation matrices (***makecov lagcov***)*

As described in *O'Brien* [2012] we can compute the lag covariance matrices on a further decimated grid. We select the points in the new decimated grid at random from the original undecimated grid. The number of points is the larger of $5N_q^2$ and $50N_q$, where N_q is the number of principal components. We then proceed through this new decimated grid to compute all the needed lag covariance matrices using the same nearest neighbors strategy used for the spatial covariance matrix in Section 1.4.3.2.

*1.4.3.5 Computing time evolution matrices (***buildmc***)*

The equations and algorithms for computing G and C in frm the R's are given in *O'Brien* [2012] and will not be repeated here (they are complicated).

Once the Q's, G's, and C are computed, we can save the entire set of runtime tables to a single file, such as AE9V10_runtime_tables.mat, which is both a Matlab save set and an HDF5 file.

1.4.4 Diagnostic Figures

A large number of diagnostic figures are generated. Manually browsing this set is a required visual validation of the model. Further empirical validation is performed by comparison to other data sets and models (see *Johnston et al.* [2014b]).

Briefly, the diagnostic figures are:

- **thetafigs** - theta maps, i.e., flux maps, for each sensor
- **filledthetafigs** - theta maps, filled, for each sensor
- **fluxmaps** - flux maps for combined model
- **binspectra** - energy spectra & fit in spatial bins (similar to Figure 5)
- **pcfigs** - Principal Components figures
- **radial_profile** - equatorial radial profiles vs AE8/AP8

- **SAA_profile** - SAA latitude profiles vs AE8/AP8
- **FieldLine_profile** - field line profiles vs AE8/AP8
- **LEO_map** - flux contour at multiple altitudes