1.3 AE9/AP9/SPM Runtime Algorithms

This section describes the algorithms employed by the AE9/AP9/SPM V1.0/V1.1 runtime library to compute radiation and plasma environments. The algorithms can be used to generate mean, percentile, perturbed mean, and dynamic scenario environments. The algorithms populate the desired global environment on the model grid, evolve it forward in time as needed, and project it onto the times, locations, energy, and angular channels requested by the user.

1.3.1 Introduction

This introduction provides a high-level description of the AE9/AP9/SPM runtime algorithms. These algorithms are an evolution of those used in prior alpha and beta versions of AE9/AP9 [see, e.g., *O'Brien*, 2005; *O'Brien*, 2007; *O'Brien and Guild*, 2010] and have been summarized in *Ginet et al.* [2013]. Later sections will repeat this information in more detail with equations and mathematical notations. The runtime algorithms generate mean, percentile, and static perturbed mean radiation and plasma environments, as well as dynamic Monte Carlo scenarios of the radiation environment. In the case of the mean and percentile environments, the parameters of the statistical distribution at each grid point are given by unperturbed parameter maps. At each grid point, the parameter map provides a two-element vector that can be converted to the median and 95th percentile of the local statistical distribution. The median and 95th percentile can then be converted to the parameters of either a Weibull or log-normal statistical distribution. With those parameters the mean or any percentile can be computed. The AE9 electron model uses a Weibull distribution, while the AP9 proton model and the plasma models use log-normal distributions. In the case of the perturbed mean static environments and the dynamic Monte Carlo environments, the unperturbed parameter map is perturbed using an anomaly matrix to obtain a different, perturbed parameter map for each scenario. The perturbations are uniquely identified by a random number seed, which is the scenario identifier (ID) number 1 through 999. The anomaly matrix represents the uncertainty in the parameters of the statistical distributions, including sensor response uncertainties, limited duration of sampling missions, counting statistics, and spatial extrapolation/interpolation.

For the dynamic Monte Carlo environments an initial random state is generated and evolved forward in time for many iterations to ensure proper temporal correlations are established before the mission simulation begins. Time evolution is achieved by a multi-lag, multivariate auto-regressive process, whose time evolution matrices (persistence and innovation conditioning matrices) are derived from observed spatiotemporal correlations, and the innovations themselves are again keyed to a random number seed corresponding to the scenario ID. The state vector itself is a reduced ~10-element vector of principal component amplitudes. The relationship between the principal component amplitudes and the flux level at any given grid point is provided by principal component matrices which are square root matrices of a specialized spatial covariance matrix. In fact, the spatial covariance matrix is computed not from the fluxes, but from the fluxes converted to standard Gaussian variables. All of the multivariate linear time iteration operations retain the Gaussian nature of the data, and also preserve the zero mean and unit variance of a standard Gaussian. Thus, one must convert from the standard Gaussian to the local Weibull or log-normal statistical distribution to obtain flux at each grid point.

With flux given at each grid point, one must then compute the requested particle fluxes at the spacecraft location. For unidirectional differential fluxes, the computation is simply linear interpolation from the grid onto the local magnetic coordinates. However, for more commonly-requested omnidirectional and/or integral fluxes, the computation may also involve integrating over the local magnetic coordinates. Angular integrals to obtain omnidirectional flux are conducted in terms of local pitch angle, assuming symmetry around the local magnetic field direction.

The particle flux in each desired energy channel is computed at each point along the spacecraft trajectory, thus providing a simulation of what one would measure with an idealized particle sensor. For the static environments, and often for the dynamic environments, time variation is due mainly to motion of the spacecraft and to a lesser extent to motion of the model currents inside and outside the Earth. However, over time, the dynamic environments will explore a larger range of variation than what appears in the static environment due simply to spacecraft motion.

The simulated sensor data can be fed into an effects code to compute the outcome of radiation or plasma interactions with matter, parts, or systems. For effects due only to whole-mission linear accumulation (average, fluence, dose), the mean and perturbed mean static environments can be used, and can be run for only a representative time sample (either a few orbits, or a set of times randomly but uniformly distributed over the course of the mission). For effects that depend on the instantaneous particle flux (single event effects, dose rate) or time history of particle flux (internal charging), the dynamic Monte Carlo scenarios must be used. Running full mission simulations can be so time consuming and can produce so much raw flux data that they can be a burden even to mid-range computing clusters and mass storage.

To obtain statistical confidence intervals, one must run multiple scenarios of either the static perturbed means or the dynamic Monte Carlo scenarios, compute the required effects from the fluxes, and the compute statistical distributions of those effects. Computing effects from percentiles of fluxes will likely lead to incorrect results because the calculation of percentiles is a nonlinear operation (sorting), even if the effects themselves are linearly dependent on the flux.

In the remainder of this section, we will describe the algorithms and equations in detail.

1.3.2 Runtime Tables

The AE9/AP9/SPM model depends on several databases of runtime tables. Each model AE9, AP9, SPMH, SPME, SPMO, and SPMHE has a model-specific database with a name like AE9V10_runtime_tables.mat. These are binary Matlab save sets and conform to the HDF5 format standard (Matlab is a trademark of The Mathworks, Inc.). There are also support databases, e.g., the coefficients of the International Geophysical Reference Field (IGRF) model [*Finlay et al.*, 2010]. The contents of a model-specific database are identified in Table 2. Each quantity in Table 2 will be used in later in this section. Section 1.4 describes the statistical manipulations required to generate each quantity.

Table 2. Main quantities in each model-specific runtime tables data file.

Quantity	Variable Name	Symbol	Size	Purpose
Parameter map	theta	θ	~20,000* x 2	Represent the transformed 50th and 95th percentile flux on the coordinate grid
Anomaly matrix	Stheta	S_θ	~40,000† x ~10	Represents error covariance matrix for θ due to measurement errors
Principal Component Matrix‡	Q	Q	~20,000* x ~10	Represents the principal components of spatial variation
Time Step‡	dt	δt	Scalar	
Persistence Matrix‡	G	G	~10 x 10 x ~5	Represents persistence of principal component amplitudes
Innovation Conditioning Matrix‡	C	C	~10 x 10	Allocates white noise driver to principal components
Conditioning Time‡	conditioning_time	N/A	Scalar	Specifies length of conditioning time needed to initialize state history
Grid	grid	N/A	(structured)	Stores information about the grid

*20,000 includes two grids and is appropriate for AE9 and AP9. SPM models are smaller
†40,000 includes two entries for each grid point
‡Monte Carlo quantities are not defined for the plasma (SPM) models.

1.3.3 Magnetic Coordinates Grid

Section 1.2 provides a detailed discussion of the drift and bounce invariant coordinate systems used in the component models of AE9/AP9/SPM. Here we will just provide a short review of the coordinate systems and then describe how they are used at runtime. The specific coordinate identities, ranges, and resolutions are given in Table 1.

The radiation belt models AE9 and AP9 actually use two grids. The high altitude grid uses energy (E), Kauffman's K, and the third adiabatic invariant or flux invariant Φ. The low altitude grid uses E, K, and the minimum altitude encountered on a drift orbit (h_{min}). It should be noted that K and h_{min} are drift invariants but not adiabatic invariants (i.e., they are constant only in a static magnetic field).

The plasma models (SPM) use E, equatorial pitch angle (α_{eq}), and McIlwain L (L_m) coordinates. Equatorial pitch angle is a bounce invariant, not a drift invariant, but was selected for historical reasons. It should be noted that the plasma particles respond to the global electric field strongly enough that their magnetic coordinates are not drift invariant anyway (a set of electromagnetic coordinates would be required).

The *grid* variable in the model-specific runtime tables provides all the information from, including identity of the coordinates, their units, ranges and resolutions, and any transforms needed (such as exponentiation or logarithms).

For AE9 and AP9, the runtime tables combine the parameter maps (θ) and the principal components matrix *Q* for the high and low altitude grids by "stacking" them, or concatenating them along the first dimension. For S_θ a second stacking is done with respect to θ_1 and θ_2. The runtime code keeps track of this stacking with help from parameters stored in the *grid* variable.

Because (as we will see later) the model formulation can represent only non-zero fluxes, zero fluxes are achieved by removing points from the model grid. This is done in two stages: first, an atmospheric loss cone is defined and use to remove points based on a relationship between the 2nd and 3rd coordinates. This loss cone filter is independent of energy and flux intensity. The second filter is determined during the statistical processing of the data (see Section 1.4), and it removes points where the median flux is smaller than the maximum flux by a specified ratio. For AP9, this ratio is 10^{-10}; for the other models, the ratio is 10^{-9}. Thus, the second filter is energy and intensity dependent. The lists of which grid points are involved in each filter are provided as part of the grid variable in the model-specific runtime tables.

Finally, the grid variable includes a set of "linear basis functions" that represent the lower, middle, and upper end points. These basis functions are tracked separately for Energy as opposed to the 2-D spatial region covered by the 2nd and 3rd invariants. The lower and upper bounds are defined to account for different kinds of grid boundaries when performing grid interpolations: no flux is allowed below K=0, below the lowest energy grid point, nor in the loss cone. At runtime there is then no need to consult the definitions of the boundaries, instead relying on the lower and upper bounds for each grid point.

Because the grid variable is simply a saved version of the Matlab object used to define the grid for all the backend processing, it contains many vestigial entries that are not used at run time. In a later version of the model, we may remove the items from the file to reduce potential for confusion.

1.3.4 Populating a Requested Global State

For each model the parameter maps (θ) provide the median (m_{50}) and 95th percentile (m_{95}) flux at each grid point. At each grid point the map provides two values θ_1 and θ_2 such that:

$$m_{50} = e^{\theta_1} \tag{7}$$

$$m_{95} = e^{\theta_1} + e^{\theta_2} \tag{8}$$

This transform ensures that any pair of real values for θ_1 and θ_2 will give $m_{95} > m_{50} > 0$. With the two percentiles in hand, one can derive the standard parameters of either the Weibull (used for AE9 electrons) or Log Normal distribution (used for everything else), and thereby compute any desired statistical property (usually the mean or a chosen percentile) of the flux at that grid point.

The cumulative distribution function for the Weibull is given by

$$F(x) = 1 - \exp[-(x/x_0)^\gamma], \tag{9}$$

and the parameters are related to the 50th and 95th percentiles by

$$\gamma = \left[\ln\frac{\ln 20}{\ln 2}\right] \Big/ \left[\ln\frac{\ln m_{95}}{\ln m_{50}}\right] \tag{10}$$

$$x_0 = \frac{m_{50}}{(\ln 2)^{1/\gamma}}. \tag{11}$$

The cumulative distribution function for the Log-Normal is given by

$$F(x) = \Phi\left(\frac{\ln x - \mu}{\sigma}\right), \tag{12}$$

where Φ is the cumulative distribution function for the unit normal or standard Gaussian. The parameters of the Log Normal are related to the 50th and 95th percentiles by:

$$\mu = \ln m_{50} \tag{13}$$

$$\sigma = \frac{\ln m_{95} - \mu}{\Phi^{-1}(0.95)}. \tag{14}$$

1.3.4.1 Mean

To populate the global state for a run through the mean environment, the θ parameters are used to compute m_{50} and m_{95} at each grid point according to the formulae above, and then the following formulae are applied to compute the mean at the grid point:

$$\langle x\rangle = x_0\Gamma(1 + 1/\gamma)\ \text{(Weibull)}, \tag{15}$$

$$\langle x\rangle = \exp[\mu + \sigma^2/2]\ \text{(Log Normal)}, \tag{16}$$

where <> represents the population mean or average, and Γ is the complete Gamma function.

1.3.4.2 Percentiles

When a given percentile flux map is requested, it is rescaled onto the [0,1] domain by the variable *u*. At each grid point, θ is converted to the parameters of the relevant cumulative distribution, and the flux is then given by the inverse of the cumulative distribution function *F* given above

$$x = F^{-1}(u). \tag{17}$$

1.3.4.3 Perturbed mean

The perturbed mean uses the same formulae to compute the mean as does the unperturbed mean. However it uses a perturbed parameter map. To distinguish the unperturbed and perturbed maps, we add the superscript (0) to the parameter map: $\theta^{(0)}$. The perturbation is computed from S_θ, which we will treat as a matrix, and we will treat $\theta^{(0)}$ and θ as vectors. The parameter map perturbation equation is then given in linear algebra notation as:

$$\vec{\theta} = \vec{\theta}^{(0)} + \underline{\underline{S}}_\theta \vec{\varepsilon} \tag{18}$$

Where $\vec{\varepsilon}$ is a vector of uncorrelated uniform random variables distributed evenly between $-\sqrt{3}$ and $+\sqrt{3}$. This distribution has zero mean and unit variance, but, unlike the unit normal, it has finite bounds. It is therefore the case that the global error covariance of the parameter map is given by:

$$\mathrm{cov}\left(\vec{\theta} - \vec{\theta}^{(0)}\right) = \langle\left(\vec{\theta} - \vec{\theta}^{(0)}\right)\left(\vec{\theta} - \vec{\theta}^{(0)}\right)^T\rangle = \underline{\underline{S}}_\theta \langle \vec{\varepsilon}\vec{\varepsilon}^T \rangle \underline{\underline{S}}_\theta{}^T = \underline{\underline{S}}_\theta \underline{\underline{S}}_\theta{}^T \tag{19}$$

This equation directly ties the runtime perturbations to the pre-computed errors in the parameter map θ via S_θ.

The perturbed parameter map can then be used along with equations from Section 1.3.4.3 to compute the perturbed mean flux at every grid point.

We note that the random number generator used to produce $\vec{\varepsilon}$ is seeded with the scenario ID number, 1-999.

1.3.4.4 Monte Carlo scenarios

To create a Monte Carlo scenario, one begins by perturbing the parameter map according to equation (18). Next, one must initialize the dynamic state of the radiation belts, using a state history matrix. The state history is initialized to a series of vectors $\vec{q}_t$ of Gaussian white noise (uncorrelated unit normal random variables). Then, for a model-dependent conditioning time [see *O'Brien*, 2012], the state history is updated using the autoregressive equations:

$$\vec{q}_t = \sum_{i=1}^{N_G} \underline{\underline{G}}_i \vec{q}_{t-\tau_i} + \underline{\underline{C}}\vec{\eta}_t, \tag{20}$$

where the $\underline{\underline{G}}$ and $\underline{\underline{C}}$ matrices and τ_i are part of the runtime tables, and $\vec{\eta}_t$ is a series of Gaussian white noise vectors. Through a set of relationships given in *O'Brien* [2012] the $\underline{\underline{G}}$ and $\underline{\underline{C}}$ matrices tie the sequence of $\vec{q}_t$'s to the observed spatiotemporal correlations of the fluxes. After advancing equation (20) for the prescribed conditioning time, the state history is expected to represent the spatiotemporal lag correlations in the associated radiation belts. However, $\vec{q}_t$ is only a small state vector, and it must be converted flux on the model grid via a three-step process.

The first step in converting the state vector into flux is expanding it onto the model grid. The state vector $\vec{q}_t$ is actually a vector of amplitudes of a set of principal components of spatial variation. The principal component matrix $\underline{\underline{Q}}$ (provided in the runtime tables) converts from the state vector to the normalized fluxes $\vec{z}_t$:

$$\vec{z}_t = \underline{\underline{Q}}\vec{q}_t. \tag{21}$$

By construction, each q and each z has a long-term statistical distribution that is also a unit normal, and the q's are uncorrelated with each other. Thus the spatial covariance of the normalized fluxes is given by:

$$\underline{\underline{\Sigma}} = \text{cov}(\vec{z}_t) = \langle \vec{z}_t \vec{z}_t^{\,T} \rangle = \underline{\underline{Q}} \langle \vec{q}_t \vec{q}_t^{\,T} \rangle \underline{\underline{Q}}^T = \underline{\underline{Q}}\,\underline{\underline{Q}}^T. \qquad (22)$$

This equation directly ties the observed spatial covariance ($\underline{\underline{\Sigma}}$) of the normalized fluxes to the dynamic states generated by the Monte Carlo scenarios.

The second and third steps un-normalize each of the *z*'s. The second step converts each *z* into its corresponding probability level *u* on the [0,1] domain, according to:

$$u = \Phi(z). \qquad (23)$$

The final step is to convert each *u* into a flux using the inverse of the cumulative distribution function at the grid point, i.e., using the same procedure in Section 1.3.4.2.

The global Monte Carlo state advances in time according to the time step in Table 3 and equation (20). The conversion from $\vec{q}_t$ to flux is performed at each grid point at each time step.

We note that the random number generator used to initialize the state vector history $\vec{q}_t$ and to produce $\vec{\eta}_t$ is seeded with the scenario ID number, 1-999. Also, the perturbed parameter map is the same one produced for the same scenario ID number for a perturbed mean state, i.e., in Section 1.3.4.3.

Table 3. Monte Carlo parameters for AE9 and AP9.

Monte Carlo Parameter	AE9	AP9
Number of Principal Components (N_q)	8	9
Number of persistence matrices (N_G)	6	4
time step (δt, days)	1	7
Time lags ($\tau_i \delta t$, days)	1, 7, 14, 27, 183, 365	7, 28, 182, 364
Conditioning time (days)	1253.1	1503.5

1.3.5 Projecting the Environment onto the Spacecraft

1.3.5.1 Fast coordinates

For the drift invariants, Φ and h_{min}, the full coordinate calculation in the Olson-Pfitzer quiet field [*Olson and Pfitzer*, 1977] is too slow even on modern computers to be practical to do on demand in an interactive application for any significant number of telemetry points. We solve this problem by using a fast field line tracer [*Pfitzer*, 1991; 1995] and separate neural networks trained on previously computed databases of either Φ and h_{min}. The neural networks are trained to be high precision replacements for the full drift trace. In addition, we introduced algorithms to identify and exclude Shabansky (bifurcated) drift orbits, as well as identifying points that are outside the domain of the model (which a drift shell integral would reveal, but which a neural network would mistake for valid inputs). The details of this technique are given in Section 1.5.

We note that the use of neural networks to avoid the full drift invariant calculation was developed simultaneously by our partners at Los Alamos National Laboratory (LANL) for a dynamic external field model [*Koller et al.*, 2009; *Koller and Zaharia*, 2011]. Our networks are similar in concept. However, they address a simpler, quiet magnetic field model but have had to meet tighter requirements in terms of error performance and boundary definitions.

1.3.5.2 Interpolation onto target magnetic coordinates

An essential step in determining the requested flux at the spacecraft location is determining the differential, unidirection flux at a specific energy and direction of incidence at the spacecraft location. For V1.0/V1.1, we do not account for finite gyroradius effects (such as the East-West effect). The model provides the differential, unidirectional flux in number/(cm^2 sr s MeV) on a grid in drift invariants. Interpolation is performed in a linear sense onto a grid of local coordinates, i.e., local pitch angles. Using our fast coordinate calculators, we can quickly obtain K, h_{min}, Φ, or L_m, or α_{eq} for any given location and direction. In the case of the AE9 and AP9 models, we begin with K, h_{min}, and determine whether the requested point falls within the low altitude grid. If so, we proceed with the interpolation using the low altitude grid. If not, we compute Φ and determine if the point falls in the high altitude grid.

Regardless of which coordinate gird we use, we compute the weights used for a linear interpolation as if we are multiplying the entire global state by a vector of weights (v). Of course, because only a small number of grid points is actually used to interpolate to a desired target location/direction, the weights are very sparse. The linear interpolation is performed in terms of the product of separate 1-D linear basis functions defined in the grid. Each linear basis function provides has the following form:

$$v(x) = \begin{cases} \frac{x - x_{low}}{x_{mid} - x_{low}} & x_{low} < x < x_{mid} \\ \frac{x_{high} - x}{x_{high} - x_{mid}} & x_{mid} < x < x_{high} \\ 0 & \text{otherwise} \end{cases}. \qquad (24)$$

The flux j at a given set of 3 coordinates ($x^{(1)}$, $x^{(2)}$, $x^{(3)}$) is then:

$$j(x^{(1)}, x^{(2)}, x^{(3)}) = \sum_{i,k,l} v_i^{(1)}(x^{(1)}) v_k^{(2)}(x^{(2)}) v_l^{(3)}(x^{(3)}) \hat{j}_{ikl}. \qquad (25)$$

where $\hat{j}_{ikl}$ is the model flux at the grid point i,k,l, computed in Section 1.3.4.

1.3.5.3 Energy integrals

The user can request differential, integral, or "wide differential" energy channels. In the latter two cases, weights must be computed to provide energy integrals either from a lower bound to infinity (effectively the upper end of the model range) or between two specified energies, respectively. The integral energy weights are computed by replacing $v_i^{(1)}$ in Equation (25) with an appropriate integral of Equation (24). The "wide differential" weights are computed by subtracting the integral weights computed for the upper energy limit from the integral weights computed for the lower energy limit, and then dividing by the difference between the channel energy limits. That is, the wide differential channel is computed as the difference of two integral channels divided by the energy bandwidth.

1.3.5.4 Angle integrals

Typically, the user requests an omnidirectional flux (*J*), which is taken to be an integral of directional fluxes over the span of local pitch angles. Local pitch angle integrals turn out to be integrals along a trajectory in the 2nd and 3rd coordinates because both depend on local pitch angle α. This is approximated by an integral in α between fluxes interpolated onto a local grid in α:

$$J = 4\pi \int_0^{\frac{\pi}{2}} j\left(x^{(1)}, x^{(2)}(\alpha), x^{(3)}(\alpha)\right) \sin\alpha \, d\alpha \tag{26}$$

The integral is approximated linearly at 5, 10, 20, ... 90 degrees:

$$J \approx \sum_m w_m j\left(x^{(1)}, x^{(2)}(\alpha_m), x^{(3)}(\alpha_m)\right). \tag{27}$$

1.3.5.5 Combined interpolation and integration

The weights w_m are combined with the linear basis functions v_i, v_j, and v_k, into a composite weight h_n that provides the combined weight for each flux on the grid, accounting for energy and directional integrals. This vector of weights h_n spans the coordinate grid and captures all the interpolation and integration required to produce the nth energy channel. A separate h_n is computed for each energy channel using the same weights for the 2nd and 3rd coordinates.

The complete weights for the various energy channels are then combined together into a matrix *H* that spans the grid in rows and the requested energy channels in columns. Thus, if the global flux state is represented as a vector of fluxes $\vec{j}$ spanning the grid, the fluxes $\vec{J}$ in the requested energy channels are given by

$$\vec{J}_t = \underline{\underline{H_t}}\vec{j}. \tag{28}$$

By judicious use of sparse matrices, *H* is computed first and then only the needed components of $\vec{j}$ are computed, thus saving evaluations of the statistical functions in Chapter 4. We have included subscripts *t* in (28) to indicate that the weights must be recomputed every time step along the spacecraft trajectory (the model fluxes $\vec{j}$ may also change, but we will address that in the next section).

1.3.5.6 Temporal interpolation (dynamic scenarios only)

The final consideration for interpolating the global state onto the spacecraft trajectory and the desired energy and directional channels is addressing time interpolation in the dynamic Monte Carlo scenarios. Time interpolation is handled linearly in the computed flux channel. That is, the global state is computed only at fiducial times (integer time steps), and the requested flux channels are computed by linear interpolating the derived local flux $\vec{J}_t$ in time. This is equivalent to allowing the global state to progress linear between fiducial times.

1.3.5.7 Plasma-radiation stitching in energy

One complication arises from the separation of the electron and proton models into radiation models (AE9/AP9) and plasma models (SPME/SPMH): integral and wide differential channels that start in the plasma energy range but extend into the radiation energy range. In V1.0/V1.1 we resolve this issue in post processing (see the application user's guide [*Roth.*, 2014]). The approach is to request a wide differential channel from the plasma model and a complementary integral or wide differential channel from the radiation model.

If the lowest energy in the radiation model is denoted E_0, then an integral channel at E_1 is given by:

$$J(E \geq E_1) = J_{\text{plasma}}(E_1 < E < E_0)[E_0 - E_1] + J_{\text{radiation}}(E \geq E_0) \quad (29)$$

As in Section 1.3.5.3, a wide differential channel is computed from the difference of two integral channels divided by the energy bandwidth.

1.3.6 Summary

This section presents a detailed description of the AE9/AP9/SPM runtime algorithms for the V1.0/V1.1 releases. These algorithms address the mathematical and geophysical representation of the radiation and plasma climatology models. The set of models can specify the design environments for total radiation dose, internal charging, proton single event effects, and plasma dose effects for any Earth orbit. The runtime algorithms provide mechanisms for estimate the probability of occurrence for various hazardous conditions due both to dynamic variability and to model uncertainty.