SAD RMS Envelope Simulation Manual

Christopher K. Allen and Masanori Ikegami
KEK, Tsukuba, Ibaraki 305-0801 Japan
December, 2005

The capability for RMS envelope simulation, including space charge, has been implemented in the SAD environment. This note is a summary of this feature. Included is a discussion on running the simulation, along with a description of the simulation parameters. Also included are a listing of the most used functions and their arguments, a listing of all the new functions implemented, along with a brief description and their calling hierarchy, and a code excerpt demonstrating how to use the simulation in SADScript.

Table of Contents

SAD RMS ENVELOPE SIMULATION PRIMER 1

1 NOTES ON RUNNING THE SIMULATION 3
 1.1 Running the Simulation 3
 1.2 Arguments of ScheffSimulate 3
 1.3 Simulation Data 5
 1.4 Modifying Beamline Parameters 6
2 BEAM PERVEANCE AND MULTI-CHARGED PARTICLE SIMULATION 6
3 THE SECOND-ORDER MOMENT MATRIX 7
4 IMPORTANT SAD FUNCTIONS AND THEIR LOCATIONS .... ERROR! BOOKMARK NOT DEFINED.
5 FUNCTION LIST AND CALLING HIERARCHY 9
 5.1 Module Scheff.n 9
 5.2 Module Trace3dToSad.n 10
 5.3 Module TwissUtility.n 10
 5.4 Module MatrixFunctions.n 10
 5.5 Internal Additions to the SAD Interpreter 11
6 EXAMPLE CODE 12
7 OUTPUT 15

1 Notes on Running the Simulation

1.1 Units

The simulation uses all MKS units, except for beam energies and particle charges. Particle charges are normalized by the unit charge q so that they appear as integral quantities, positive or negative. Beam energies are in electron-Volts (including mass energies). All statistical quantities are given in the RMS values, for example, all emittances are specified as RMS emittances rather than effective emittance or normalized emittance.

1.2 Running the Simulation

The general procedure for running the simulation is to load the input deck, specify the initial beam parameters, convert them from Trace3D units to SAD units if necessary, then run the simulation with a call to `ScheffSimulate[]`. The results of the simulation are returned by the function and may be processed as desired.

Before the call to `ScheffSimulate[]` can be made, there are several SAD environment parameters that must be set. We list them below with at short explanation.

```
TRPT;      ! Simulate a transport section
INS;       ! Treat lattice as an insertion device (not part of a ring)
NOCOD;     ! No closed orbit dispersion
RFSW;      ! Radio frequency standing wave – otherwise traveling wave structures
```

Also, to use the function `PlotBeamBeta[]` you should set the following parameter

```
$DisplayFunction = CanvasDrawer;
```

1.3 Arguments of ScheffSimulate[]

`ScheffSimulate[]` uses an adaptive stepping procedure. Consequently, there are several numerical parameters, as well as the beam parameters, that can be passed to `ScheffSimulate[]` to fine tune this integration process. All of the numerical parameters are optional parameters having default values. These default values have been set for a reasonable performance/precision tradeoff. However, for some cases you may wish to experiment to see which gives you the better performance and/or solution. Consequently, we list here all the arguments, their description, and their default values if they are optional.

$\{\{s_n\},\{\gamma_n\},\{\sigma_n\}\}$ = `ScheffSimulate`[K_0, σ_0, h_0:0.01, $\varepsilon_{\text{soln}}$:$10^{-5}$, Δh:0.05, h_{max}:0.0]

- K_0 = Initial generalized beam perveance of the beam, that is, the perveance at the entrance of the beamline. This parameter determines the space charge force and is dependent upon particle energy. For a detailed description of generalized perveance *K* see Section 2.

- σ_0 = Initial second-order moment matrix of the beam. This is the 6×6 real, symmetric matrix containing all the second-order moments of the particle beam (with respect to the beam distribution). For a description of the initial moment matrix σ_0, and the moment matrix σ in general see Section 3.
- h_0 (optional) = The initial step size to used to start the adaptive integration algorithm. The integration algorithm continually adjusts the actually step size *h* to maintain the error tolerance specified by the argument $\varepsilon_{\text{soln}}$ (see below). Choosing an initial step close too small will marginally slow the solution time, while choosing a value too large will not significantly affect the solution time. The current default value of h_0 is 1 cm.
- $\varepsilon_{\text{soln}}$ (optional) = The integration algorithm is designed to maintain a specific accuracy in the computed solution. Morever, it is designed to keep the step size *h* as large as possible such that $\|\sigma_{\text{sim}} - \sigma_{\text{soln}}\| \leq \varepsilon_{\text{soln}}$ at each step, where σ_{sim} is the simulated moment matrix and σ_{soln} is the "exact" moment matrix. This technique provides the fastest solution time while maintaining the given error tolerance. The current default value of $\varepsilon_{\text{soln}}$ is 10^{-5}.
- Δh (optional) = Slack tolerance parameter. For each integration step in the simulation, a new value of the step size *h'* is computed. If *h'* is greater than the previous step *h*, the integration step must be rolled back and the solution recomputed with new step size *h'* in order to maintain the solution tolerance $\varepsilon_{\text{soln}}$. However, if *h'* is only marginally smaller than *h* (perhaps due to noise or rounding errors), this is a significant waste of CPU time. Thus, we only change the integration step size if $|h'-h|/h > \Delta h$. (The reason for the absolute value is that we also must recompute the element sub-transfer matrix $\exp(h\mathbf{A})$ if *h* is changed in either direction.) Thus, Δh provides slack for *h* to "jitter" before the adaptive stepping is triggered. Consequently, Δh can save significant CPU time, however, choosing a value to large will compromise the accuracy of the solution and the solution tolerance $\varepsilon_{\text{soln}}$. The current default value of Δh is 5%.
- $h_{\max}$ (optional) = Maximum allowable step size. For whatever reason, it is possible for the user to specify a maximum step size for the adaptive stepping algorithm. This value will prevent the algorithm from taking any step sizes *h* greater than $h_{\max}$. To turn off this feature set $h_{\max} = 0$. The current default value of $h_{\max}$ is 0 (no maximum step size).

1.4 Results of ScheffSimulate[]

Here we list the returned values of `ScheffSimulate[]` along with a description. No output of `ScheffSimulate[]` is stored in the SAD environment. It is all returned by the function as a list of objects.

$\{\{s_n\},\{\gamma_n\},\{\sigma_n\}\}$ = `ScheffSimulate`[K_0, σ_0, h_0:0.01, $\varepsilon_{\text{soln}}$:$10^{-5}$, Δh:0.05, h_{max}:0.0]

- $\{s_n\}$ = Set of element entrance and exit positions along the beamline. The first value in this list, s_0, is the entrance position of the first beamline element, that is, element $n = 0$. Typically this value is $s_0 = 0$. The following values are the entrance positions of the beamline elements proceeding downstream. Note that the exit position of element n is the entrance position of element $n + 1$. Thus, the last value in $\{s_n\}$ is the exit position of the final beamline element.
- $\{\gamma_n\}$ = Set of design relativistic parameters at each beamline element. Note that since there is a one-to-one correspondence between this set and the number of beamline elements, there is one less value in $\{\gamma_n\}$ as compared to $\{s_n\}$.
- $\{\sigma_n\}$ = Set of beam moment matrices at each location in $\{s_n\}$. These values are the main result of the RMS envelope simulation and contain all the beam state data. For a complete description of the second-order moment matrix σ see Section 3.

1.5 Simulation Data

The input deck is typically located in a separate file and is loaded into the SADScript environment using the `GetMAIN`[*InputDeckFile*] function of SAD. Once the input deck is loaded, you must extract the beamline that you wish to simulate with the call `BL = ExtractBeamLine`[*BeamLineName*]. Finally the beamline is specified by the FFS function `USE BL`.

Within input deck file, along with the definition of the lattice under simulation, there should be the following parameter statements

```
MASS   = 0.939294 GEV;
CHARGE = -1;
MOMENTUM = 0.610624 GEV;
```

which specify the mass, charge, and initial momentum of the beam particle, respectively (we have used the example of an 181 MeV proton). These parameters are used as design parameters for the machine lattice.

The other beam parameters are specified as arguments to the main simulation function `ScheffSimulate`[K_0, σ_0]. Specifically, these arguments are the initial generalized beam perveance K_0, the initial moment matrix $\sigma_0 = \langle \mathbf{z}\mathbf{z}^T \rangle$ (the other arguments are optional numerical tuning parameters). For a complete description of these parameters see Section 1.3.

The output data of `ScheffSimulate[]` is **not** stored in the SAD environment. The returned values of element positions $\{s_n\}$, gammas at each element $\{\gamma_n\}$, and moment matrix at each element $\{\sigma_n\}$ are all independent of SAD.

1.6 Modifying Beamline Parameters

`ScheffSimulate[]` retrieves the beamline parameters directly from the SAD environment. So it is possible to change a beamline element parameter directly then immediately re-run the simulation. For example, we may use a call to the SADScript environment such as

```
ELEMENT[“K1”, “QuadHor02”]  = PI/3;
```

which will set the parameter “K1” of the element named “QuadHor02” to PI/3. The simulation can then be re-run with the new element setting. However, except for MASS, CHARGE, and MOMENTUM, `ScheffSimulate[]` ignores all other beam input parameters in the SAD environment. The other beam parameters must be set using the values of initial generalized beam perveance K_0 and initial moment matrix $\boldsymbol{\sigma}_0$ (as arguments to `ScheffSimulate`[]).

2 Perveance and Simulating Particles with Multiple Charges

The generalized beam perveance K is given by

$$K = \frac{qI}{2\pi\varepsilon_0 m\gamma^3 v^3} = \frac{I}{2\pi\varepsilon_0}\frac{q}{mc^2}\frac{1}{\beta^2\gamma^3} = \frac{I}{2\pi\varepsilon_0}\frac{1}{E_R}\frac{1}{\beta^2\gamma^3}\,, \tag{1}$$

where q is the unit charge, ε_0 is the electric permittivity of free-space, v is the particle velocity, c is the speed of light, $\beta = v/c$ is the normalize velocity, γ is the relativistic factor, $I = qN_l v$ is the average current with N_l being the number of particles per unit length, and $E_R = mc^2/q$ is the rest energy of the beam particle in electron-Volts.

For an RF system with frequency f, bunch charge Q is related to I as

$$Q = \frac{I}{f}, \tag{2}$$

where we assume the bunch spacing is $\beta\lambda$. Substituting Eq. (2) to (1), we find

$$K = \frac{qQf}{2\pi\varepsilon_0 m\gamma^3 v^3} = \frac{Q}{2\pi\varepsilon_0}\frac{q}{mc^2}\frac{f}{\beta^2\gamma^3} = \frac{Q}{2\pi\varepsilon_0}\frac{1}{E_R}\frac{f}{\beta^2\gamma^3}\,, \tag{3}$$

which means that we need information on q, m, and Q for determining K. Physically speaking, the space-charge field E is determined by Q, and the space-charge forces acted on the individual particles are determined by the product of q and E. So, it is natural to have qQ in K.

Interpreting “m” as “m/q”, you can simulate multi-charged particles (with q other than 1 or −1), because K depends on q/m, not on q and m independently. Thus, for a particle of charge nq and the same mass m, we would substitute $E_R \to E_R/n$ in all the simulation functions (e.g., `ComputePerveance[]`, `TraceToSadLongTwiss[]`, etc.)

3 The Second-Order Moment Matrix

3.1 Definition of the Moment Matrix

We denote the phase space coordinates of a charged particle as $\mathbf{z} = (x\ x'\ y\ y'\ z\ dp/p_0)^{\mathrm{T}}$ where x, y, and z are the offsets of the particle from the synchronous particle position in the three orthogonal directions, $x' = dx/ds = \dot{x}/\dot{s} = p_x / p_0$, $y' = dy/ds = \dot{y}/\dot{s} = p_y/p_0$, s is the path length parameter, the over-dot represents differentiation with respect to time, p_0 is the design momentum of the synchronous particle, and dp is the difference in momentum from the design momentum. Every beam particle can be identified by its position $\mathbf{z}$ in phase space. Consider a distribution of particles in phase space, represented by the beam density function $f : \Re^6 \to \Re$. Thus, $f(\mathbf{z})d^6\mathbf{z}$ is the number of particles within a differential volume $d^6\mathbf{z}$ at location $\mathbf{z}$ in phase space. For any function $g : \Re^6 \to \Re$ on phase space, the moment of g with respect to the distribution f, denoted $\langle g\rangle$, is defined by $\langle g\rangle \equiv \int g(\mathbf{z})f(\mathbf{z})d^6\mathbf{z}$. The matrix of second-order moments of the beam, or moment matrix $\boldsymbol{\sigma}$, is defined by $\boldsymbol{\sigma} \equiv \langle \mathbf{z}\mathbf{z}^T\rangle$, where $\mathbf{z}\mathbf{z}^T$ is the outer product of the phase space coordinates. Expanding out this definition we see that $\boldsymbol{\sigma}$ appears as

$$\boldsymbol{\sigma} = \left\langle \mathbf{z}\mathbf{z}^T \right\rangle = \begin{pmatrix} \langle x^2\rangle & \langle xx'\rangle & \langle xy\rangle & \langle xy'\rangle & \langle xz\rangle & \langle x\delta\rangle \\ \langle xx'\rangle & \langle x'^2\rangle & \langle x'y\rangle & \langle x'y'\rangle & \langle x'z\rangle & \langle x'\delta\rangle \\ \langle xy\rangle & \langle x'y\rangle & \langle y^2\rangle & \langle yy'\rangle & \langle yz\rangle & \langle y\delta\rangle \\ \langle xy'\rangle & \langle x'y'\rangle & \langle yy'\rangle & \langle y'^2\rangle & \langle y'z\rangle & \langle y'\delta\rangle \\ \langle xz\rangle & \langle x'z\rangle & \langle yz\rangle & \langle y'z\rangle & \langle z^2\rangle & \langle z\delta\rangle \\ \langle x\delta\rangle & \langle x'\delta\rangle & \langle y\delta\rangle & \langle y'\delta\rangle & \langle z\delta\rangle & \langle \delta^2\rangle \end{pmatrix}, \qquad (4)$$

where we have abbreviated dp/p_0 as δ. This matrix is the main object of concern in the simulation containing all the particle beam information.

3.2 Initial Moment Matrix

To initialize the simulation, we need an initial value for $\boldsymbol{\sigma}$, which we denote as $\boldsymbol{\sigma}_0$. This quantity is computed from the initial Twiss parameters of the beam in all three phase planes. Given the initial Twiss parameters $(\alpha_x, \beta_x, \tilde{\varepsilon}_x)$, $(\alpha_y, \beta_y, \tilde{\varepsilon}_y)$, $(\alpha_z, \beta_z, \tilde{\varepsilon}_z)$, for each phase plane x, y, and z, respectively, the value of $\boldsymbol{\sigma}_0$ should be

$$\boldsymbol{\sigma}_0 = \begin{pmatrix} \beta_x\tilde{\varepsilon}_x & -\alpha_x\tilde{\varepsilon}_x & 0 & 0 & 0 & 0 \\ -\alpha_x\tilde{\varepsilon}_x & \gamma_x\tilde{\varepsilon}_x & 0 & 0 & 0 & 0 \\ 0 & 0 & \beta_y\tilde{\varepsilon}_y & -\alpha_y\tilde{\varepsilon}_y & 0 & 0 \\ 0 & 0 & -\alpha_y\tilde{\varepsilon}_y & \gamma_y\tilde{\varepsilon}_y & 0 & 0 \\ 0 & 0 & 0 & 0 & \beta_z\tilde{\varepsilon}_z & -\alpha_z\tilde{\varepsilon}_z \\ 0 & 0 & 0 & 0 & -\alpha_z\tilde{\varepsilon}_z & \gamma_z\tilde{\varepsilon}_z \end{pmatrix}, \qquad (5)$$

where the tilde indicate RMS quantities. Note that there is not coupling between phase planes for the initial $\boldsymbol{\sigma}$ matrix. During the simulation this condition may change (e.g., in the presence of bending magnets, misalignments, etc.). The function `CorrelationMatrix6D[]` is provided for computing this matrix given the initial Twiss parameters.

3.3 Propagation of the Moment Matrix

In linear beam optics the phase space coordinates of a particle transform as $\mathbf{z}_{n+1} = \Phi_n \mathbf{z}_n$ where Φ_n is the transfer matrix of beamline element n, $\mathbf{z}_n$ is the phase space coordinate of the particle at the entrance of n, and $\mathbf{z}_{n+1}$ is the phase space coordinate at the exit of n. With no space charge, we may derive the propagation equations for the moment matrix directly from its definition and the propagation equations for the single particle. For a beamline element n the result is

$$\sigma_{n+1} = \Phi_n \sigma_n \Phi_n^T \tag{6}$$

where Φ_n is the transfer matrix of element n, σ_n is the moment matrix at the entrance of n, and σ_{n+1} is the moment matrix at the exit of n. This is the tenet upon which the simulation is built. However, to include space charge effects we must determine the self forces (from σ) then augment the dynamics $\sigma_{n+1} = \Phi_n \sigma_n \Phi_n^T$ accordingly.

4 SADScript Packages and Important Functions

The following is a listing of the new SADScript packages used in the RMS envelope simulation. A description of each package is given, along with a listing of the functions that you will typically use most in the SAD RMS envelope simulation. The functions are included with their arguments, shown with the standard notion found in the literature. A complete listing of all the implemented functions, along with a brief description, is provided in Section 5.

- oldsad/Packages/Scheff.n
  - $\{\{s_n\},\{\gamma_n\},\{\sigma_n\}\}$ = ScheffSimulate[K_0, σ_0, h_0:0.01, ε:10^{-5}, Δh:0.05, h_{max}:0.0]
  - $\{\{s_n\},\{\gamma_n\},\{\Phi_n\}\}$ = GetBeamLineElementData[]
  - SaveBeamMatrixData[*file*, $\{s_n\}$, $\{\gamma_n\}$, $\{\sigma_n\}$]
  - PlotBeamBeta[$\{s_n\}$,$\{\sigma_n\}$]

This package is the main simulation package and includes the simulation driver, `ScheffSimulate[`K_0, σ_0, Δs:0.01`]` whose arguments are the initial generalized beam perveance K_0, the initial moment matrix $\sigma_0 = \langle \mathbf{zz}^\mathrm{T} \rangle$ and the (optional) initial step size h_0, the (optional) solution error tolerance ε, the (optional) slack tolerance in computing new stizes Δh, and the (optional) maximum allowable step size h_{max}. (For a complete description of `ScheffSimulate[]` see Section 1.3). Its function directly depends upon one other module, `MatrixFunctions.n`. You will typically use the function `ComputePerveance[]` from `Trace3dToSad.n`, to compute the beam perveance from the beam parameters, and `CorrelationMatrix6D[]` from `TwissUtility.n` to compute the initial moment matrix σ_0 from the initial Twiss parameters of the beam.

- oldsad/Packages/Trace3dToSad.n
  - K = ComputePerveance[f, E_R, W, Q]
  - $\{\alpha,\beta,\varepsilon\}_{\mathrm{SAD}}$ = TraceToSadTransTwiss[$\{\alpha,\beta,\varepsilon\}_{\mathrm{T3D}}$]
  - $\{\alpha,\beta,\varepsilon\}_{\mathrm{SAD}}$ = TraceToSadLongTwiss[f, E_r, W, $\{\alpha,\beta,\varepsilon\}_{\mathrm{T3D}}$]

A convenience module for converting the parameters used in Trace3D (i.e., those found in the Trace3D input file) to those used by SAD, and vice-versa. The two codes use different units so it is helpful to have functions to convert between them. The most used are listed above, where the argument f is the RF frequency (Hz), E_R is the beam particle rest energy ($=mc^2$) in (eV), W is the beam kinetic energy (eV), Q is the bunch charge (C), and $\{\alpha,\beta,\varepsilon\}$ are the Twiss parameters for a single phase plane. This module has no dependencies.

- oldsad/Packages/TwissUtility.n
  - σ = CorrelationMatrix6D[$\{\alpha,\beta,\varepsilon\}_x$, $\{\alpha,\beta,\varepsilon\}_{\mathrm{y}}$, $\{\alpha,\beta,\varepsilon\}_{\mathrm{z}}$]

A convenience module for working with the Twiss parameters of a beam. The most used function will typically be `CorrelationMatrix6D[]`, which builds a 6×6 moment matrix σ from Twiss parameters in the three phase planes. The moment matrix is block diagonal and composed of 2×2 sub-matrices from each of the uncoupled phase planes. This module has no dependencies.

- oldsad/Packages/MatrixFunctions.n
  - $\mathbf{F}$ = MatrixLog[$\mathbf{\Phi}$]
  - $\mathbf{\Phi}$ = MatrixExp[$\mathbf{F}$]

Many matrix functions are included in this module. The two most important are the matrix logarithm `MatrixLog[]` and matrix exponential `MatrixExp[]`. However, there are also functions for computing matrix inner products and norms. The matrix logarithm function is designed for symplectic arguments and is not robust in general. The function may no converge for arguments sufficiently far from the identity matrix, even though the argument does, indeed, have a logarithm.

5 Function List and Calling Hierarchy

The follow is list of all the new functions added to the SAD Environment, both as SADScript and compiled code. A brief description of the function is provided in the margin. For functions implemented in SADScript, a full description is given in the source code, including the arguments and the returned values. All SADScript modules are located under the `oldsad/Packages/` directory in the repository.

5.1 Module Scheff.n

ScheffSimulate	- Main RMS simulation function
ScheffZeroPropagate	- envelope simulation w/out space charge (K=0)
ScheffPropagate	- envelope simulation with space charge
ScheffPropElem	

StepSigmaMatrix	– steps the moment matrix a specified length
CompStepSize	- computes the new step size for integration
ScheffGenerator1	- compute the space charge matrix in beam frame
ScheffGenerator2	- compute the space charge matrix in the lab frame
ScheffDecoup	- decouple the phase planes in configuration space
ScheffDecoupRotMatrix	– get matrix in $SO(6)$ zeroing largest $\langle x_i x_j \rangle$
SaveBeamMatrixData	- save solution data to disk
PlotBeamBeta	- plot solution beta function in three phase planes
GetBeamlineElementData	- return beamline data $\{\{s_n\},\{\gamma_n\},\{\Phi_n\}\}$

5.2 Module Trace3dToSad.n

ComputePerveance	- Compute generalized beam perveance
TraceToSadCoords	- Convert Trace3D phase space coordinates to SAD coords
SadToTraceCoords	- Convert SAD phase space coordinates to Trace3D coords
TraceToSadTransTwiss	- Convert Trace3D transverse Twiss parameters to SAD
TraceToSadLongTwiss	- Convert Trace3D longitudinal Twiss parameters to SAD
SadToTraceTransTwiss	- Convert SAD transverse Twiss parameters to Trace3D
SadToTraceLongTwiss	- Convert SAD longitudinal Twiss parameters to Trace3D

5.3 Module TwissUtility.n

CorrelationMatrix6D	- Build full 6×6 second-order moment matrix from Twiss
CorrelationMatrixTrans	- Build transverse 4×4 moment matrix from Twiss
CorrelationMatrix	- Build 2×2 moment matrix for single phase plane
GetEnvelopeFromTwiss	- Compute $(\sqrt{\langle x^2 \rangle}, \sqrt{\langle x^2 \rangle}, \varepsilon)$ from Twiss parameters
GetTwissFromEnvelope	- Compute $(\alpha, \beta, \varepsilon)$ from the beam envelope

5.4 Module MatrixFunctions.n

MatrixLog	- Computes the log of a matrix close to identity
MatrixInnerProd2	- Computes the l_2 inner product of a matrix in $\Re^{n\times n}$
MatrixZassen	- Zassenhaus formula of two matrices to 2nd order
MatrixCommutator	- Computes the commutator of two matrices
MatrixLogTaylor	- Taylor expansion of matrix logarithm function
MatrixExp	- Compute the exponential of a matrix
MatrixNorm2	- Compute the l_2 norm (squared) of a matrix in $\Re^{n\times n}$

MatrixInnerProd2	- Induces the l_2 norm
MatrixExpTaylor	- Taylor expansion of the matrix exponent. function

5.5 Internal Additions to the SAD Interpreter

EllipticRd(*x*,*y*,*z*) - Computes the Carlson elliptic integral of the second kind given by the formula

$$R_D(x,y,z) \equiv \frac{3}{2}\int_0^\infty \frac{dt}{(t+x)^{1/2}(t+y)^{1/2}(t+z)^{3/2}}$$

6 Example Code

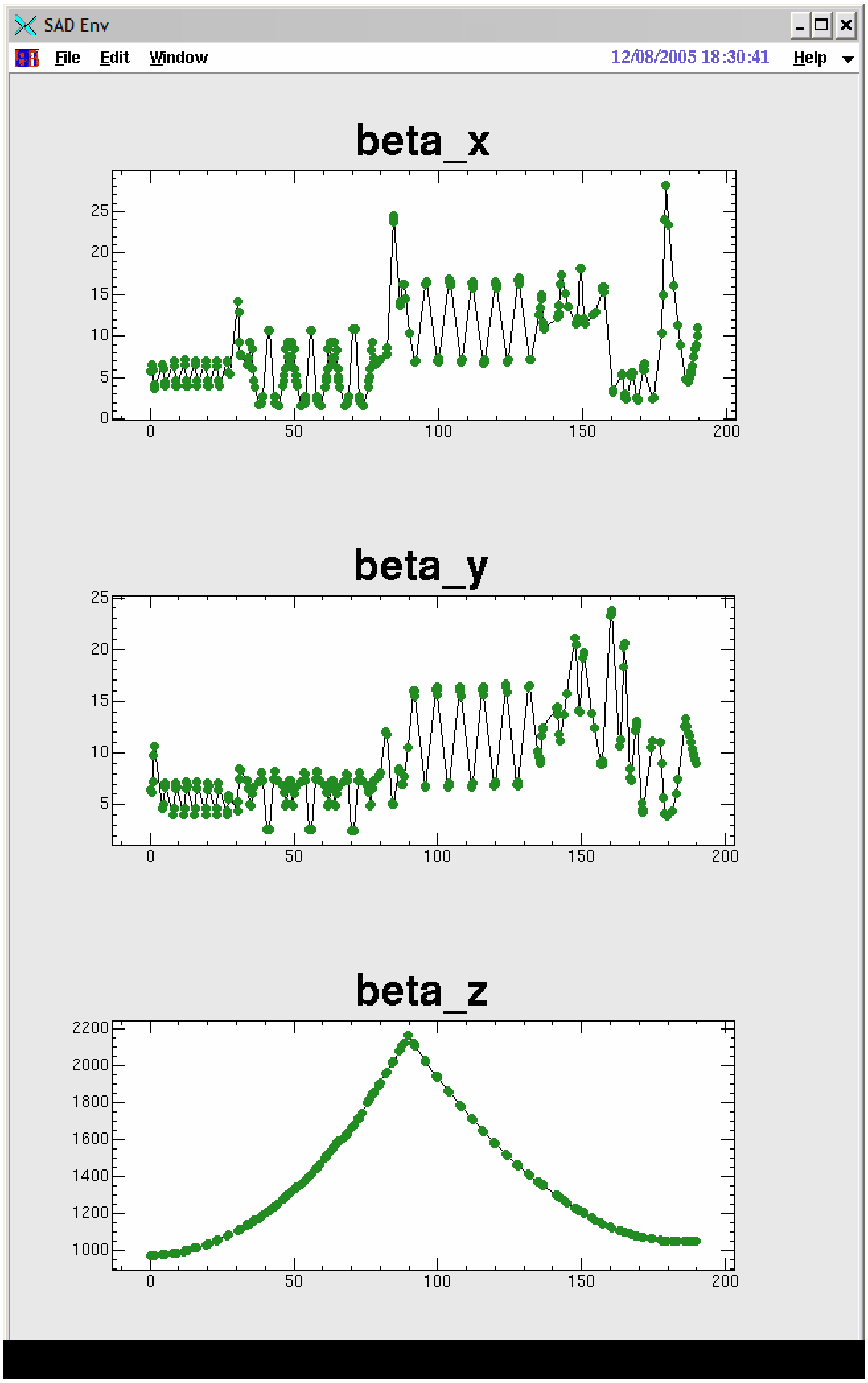
Below is a code excerpt from a SADScript file simulating the beam transport section of the J-PARC machine. This excerpt lists the essential code for a complete RMS envelope simulation. The complete file can be found on the SAD cluster at the location `/usr/users/ckallen/code/testing/ScheffTest.sad`.

```
!!========================================
!!
!!  Initialize SAD
!!
!!========================================


FFS;    ! Begin SADScript


!
!  Load Beamline
!

GetMAIN["~ckallen/J-Parc/linac/simdb-LI_L3BT01-nopmq0000.sad"];
L3BT01 = ExtractBeamLine["L3BT01all"];

USE L3BT01;


!
!         Initialize SAD Environment – These are important!
!

TRPT;           ! a transport line, not a ring
INS;            ! mimic an insertion device
CAL;            ! do SAD calculation of beamline

NOCOD;          ! no closed orbit correction
RFSW;           ! this is for RF gaps, use standing wave structure

$DisplayFunction = CanvasDrawer;          ! if you want to use PlotBeamBeta[]
```

```
!
!  INITIALIZE SIMULATION
!


!
!    TRACE3D Parameters
!

f  = 324.0e6;              ! RF frequency (Hz)

Er = 939.29432e6;      ! particle rest energy (eV)
W  = 181.0338e6;       ! beam kinetic energy (eV)
XI = 30.0e-3               ! beam current (A)

vecTwissXt3d = {-0.44117, 5.774, 1.889};
vecTwissYt3d = {0.21808, 6.4229, 1.706};
vecTwissZt3d = {0.3095, 2.0888, 466.99};

!
!          Numerical Parameters
!

h0 = 0.01;                 ! initial step length
errSoln = 1.0e-5;        ! solution error tolerance
hslack = 0.05;            ! adaptive step backlash tolerance
hmax = 0.0;               ! maximum step length (=0 turned off)


!
!    Compute SAD Parameters (Convert from Trace3D)
!

Q  = XI/f;                          ! beam bunch charge (C)

K0   = ComputePerveance[f, Er, W, Q];


vecTwissX = TraceToSadTransTwiss[vecTwissXt3d];
vecTwissY = TraceToSadTransTwiss[vecTwissYt3d];
vecTwissZ = TraceToSadLongTwiss[f, Er, W, vecTwissZt3d];

sig0 = CorrelationMatrix6D[vecTwissX, vecTwissY, vecTwissZ];
```

```
!
!  RUN SIMULATION
!


! Run simulation
{lstPos, lstGamma, lstSig} = ScheffSimulate[K0, sig0, h0, errSoln, hslack, hmax];

! Store results
SaveBeamMatrixData[“RmsEnvOutput.txt”, lstPos, lstGamma, lstSig];

!  Look at the Results
PlotBeamBeta[lstPos, lstSig];

Exit[];
```

7 Output


SAD Env
File
Edit
Window
12/08/2005 18:30:41
Help
beta_x
beta_y
beta_z
0
50
100
150
200