HPSim User Guide

X. Pang and L. Rybarcyk, AOT-AE

January 31, 2017

LA-UR-17-20805

Los Alamos
NATIONAL LABORATORY
EST. 1943

Table of Contents

Introduction

The High Performance online multi-particle beam dynamics Simulator, *HPSim [1]*, was developed to aid in the tune-up and operation of ion linacs, specifically the 800-MeV proton linac at Los Alamos. It was developed by Xiaoying Pang and Larry Rybarcyk of the Accelerators and Electrodynamics group in the Accelerator Operations and Technology Division, AOT-AE, at the Los Alamos National Lab and is based upon the physics models and coordinate system used in the PARMILA [2] code, also developed at Los Alamos. HPSim was created to produce a more accurate and realistic simulation of beam in an actual, operating linac when compared to envelope or single particle codes. HPSim achieves outstanding performance over single-threaded CPU execution through the use of NVIDA GPU (graphic processing unit) technology. The linac layout, machine parameters accessible through EPICS [3] and conversion between engineering and physics model quantities are stored in an SQLite [4] database that the code interacts with. The number-crunching code is written is CUDA [5] C and C++, which along with the GPU hardware enables the performance gain. The online version is a stand-alone C++ program that runs in a continuous loop, updating plots of various beam quantities, while monitoring changes to the linac operating parameters available via EPICS. In contrast, the offline version of the code is controlled through Python [6] scripts, which affords tremendous flexibility in the use of the code.

HPSim was created to produce a more accurate and realistic simulation of beam in an actual, operating linac when compared to an envelope or single particle codes. Since the intent was to simulate and not design a real linac, the description of the linac geometry, design specifications, etc. must come from other sources, e.g. PARMILA & SUPERFISH [7]. The simulation is meant to be more accurate than an envelope code through the use of better physics algorithms, more realistic beam distributions, i.e. multiparticle, and space-charge effects.

The state of the linac captured in the simulation is controlled through EPICS process variables, just like on the actual machine, e.g. DTL Tank 1 RF phase set point, which is used to control the phase of the rf field in tank 1. In the online mode, a server monitors EPICS and updates the SQLite db as PV's are changed. In the offline mode, the user modifies the EPICS PV values stored in the db through Python scripts. This approach enables the results from the simulation of the linac in the model to be easily compared to those from the actual machine. In order for this comparison to be correct, the model needs to be calibrated. Calibration constants and functions include, rf phase offsets and amplitude scale factors, magnet excitations curves and space-charge compensation factors. While some calibration data are obtained from off-line measurements, e.g. magnet-mapping data to obtain an excitation curve, others are obtained from beam-based measurements, e.g. linac rf phase offsets. By comparing measured data, e.g. phasescan with the simulated phasescan results from HPSim, the proper calibrations constants can be determined and installed into the dB.

Besides using HPSim for tune-up and production, it can also be used to study optimization of linac operational set points. This was performed and reported in two recent studies. The first focused on the use of HPSim with multi-objective particle swarm and genetic algorithms to optimize the DTL RF set points for low-loss, high-quality beam operation and was reported in [8]. The second study HPSim was used as a virtual accelerator and test bed for a new control scheme as reported in [9]. In this study the new real-time control algorithm used HPSim to test the concept by controlling several coupled components of the linac to achieve and maintain good beam quality.

This manual is not exhaustive, but is meant to describe those aspects of the code necessary for running it. The first section is devoted to the database structure and the beamline components used by HPSim. Four database files, tbtd.db, dtl.db, trst.db and ccl.db, contain the beamline/linac layout for the LANSCE H- beam from 750 keV low-energy beam transport (LEBT) to the end of the 800-MeV linac. The next section lists the documentation on the Python API's available for scripting the offline version, which is necessary when writing new scripts. The third section contains example scripts that use HPSim to perform a simple simulation, scan and EPICS PV, and fit PS201 data for model calibration constants. The last section has some brief comments about using the 2D and 3D versions of HPSim in the online mode.

To use HPSim, one needs a workstation/server with an NVIDIA GPU card installed, e.g. K20. Our installation uses Scientific Linux 6.3. The following software packages need to be installed:

1)	Make	yum
2)	gcc-c++ 4.4.6 or later	source
3)	NVIDIA CUDA Toolkit 5.0 or later	source
4)	Python 2.7 (.3 or later in usr/local)	source
5)	Numpy 1.6.2 or later	source
6)	SQlite3.1.	yum
7)	Glew 2.0.0	source
8)	freeglut 2.6.0	yum
9)	libGL, libGLU 9.0.0	yum
10)	libEGL 10.1.2	yum
11)	libgssglue 0.1	yum
12)	libXi, libXi-devel	yum
13)	libXmu, libXi-devel	yum
14)	blax, blas-devel	yum
15)	lapack, lapack-devel	yum
16)	tk-devel	yum
17)	Scipy 0.11 or later	source
18)	Matplotlib 1.1.1 or later	source

Please refer to the NVIDIA web site for details regarding the GPU driver and CUDA Toolkit installation process. In addition for the online mode, the following software must be installed:

1)	EPICS R3.14.11 or later.	source

Some of the packages may be installed on Linux using "yum" while others need to be build and installed from source files.

SQLite Database

The SQLite database is ultimately used to produce the physics model of the accelerator and beam lines for use by HPSim. It comprises several parts. First, it contains a description of the linac as defined by the elements that make it up. Second, it contains a local copy of the EPIC PV's used to control the accelerator's operational parameters. Third, it contains the conversion algorithms that are used to transform EPICS PV quantities into physics model quantities used in the simulation. Finally, it contains calibration constants used by the conversion algorithms in transforming those values.

The linac description is based upon elements contained within the database. Table 1 contains a list of the element types presently available in HPSim. It is

Table 1: List of element types used in HPSim.

Beam Transport elements & tables	**Database Table name**
Buncher: single-gap	buncher
Circular Aperture	caperture
Diagnostic	diagnostic
Dipole magnet	dipole
Drift	drift
Quadrupole magnet	quad
Steering magnet	steering
Rectangular aperture	raperture
Rotation (about z-axis)	rotation
Space-charge compensation	spch_comp
Steering magnet	steerer
Linac structures, elements & tables	
DTL (drift-tube linac)	
CCL (coupled-cavity linac)	
CCL tank	ccl_tank
RF accelerating gap	rf_gap
RF module	rf_module
Transit-time factor table	transit_time_factor
Miscellaneous Tables	
Diode (llrf)	diode
EPICS channel	epics_channel
Quadrupole Calibration Curve	quad_family

convenient create a database file for each major section of the linac to keep them more manageable in size. The database file contains "tables", "triggers" and "views". The tables contain all of the data used to describe the linac. A trigger is a small bit of SQL code that updates various calculated quantities whenever a related EPICS PV is changed. A view is just a particular way of presenting the information in the database. All elements of the same type are contained in one table in a database file. We will use the “tksqlite.tcl” application to show the content of the databases for some of the LANSCE beam lines in the following sections. Figure 1 shows the arrangement of tables and views in the “tbtd.db” file that describes the LANSCE 750-keV H- LEBT from the exit of the CW column to the front of the 100-MeV DTL.

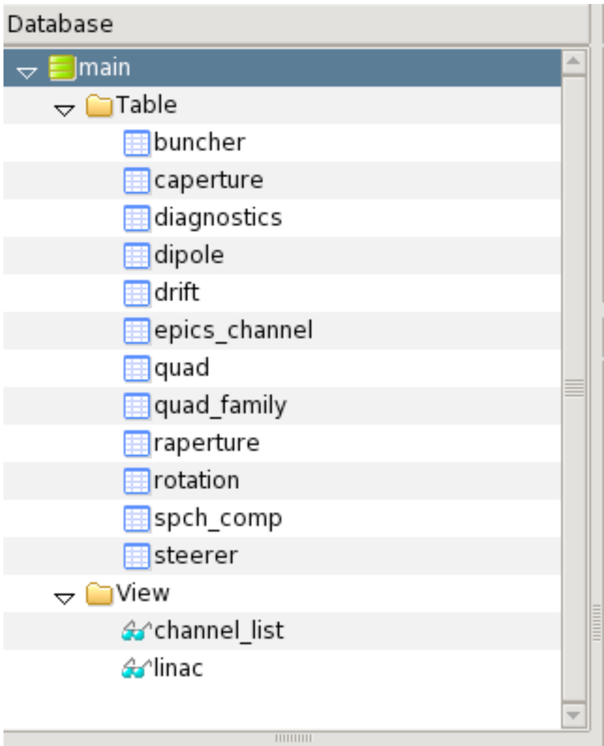

Figure 1: tksqlite.tcl application view of tbtd.db

Each beam line element in the database has a unique name. Data that are used to define the element are stored in database “fields”. The database field naming convention used is shown in Table 2.

Table 2: Database field naming convention.

Field ending in	Description
_cal	Static value used to convert EPICS PV value to actual model value
_channel	ID number of EPICS channel used as input to calculated quantity
_design	Static quantity representing nominal design of element, which are used in calculating _model values
_id	An integer used to number element of located it in a sequence of elements

_model	Calculated or static quantity written to pinned memory and used in simulation
_tmp	Internal intermediate results
_type	Name of element type

All elements have fields that are common amongst them and unique to them. The common fields are list in Table 3.

Table 3: Database field common to all elements.

Field name	Type	Description
id	int	unique id number within table
name	text	unique name to used to access element
view_index	float	unique number that places it sequence of beam line
model_type	text	type of beam line element
model_id	int	sequence # written by HPSim

All fields are readable but not writable from an HPSim Python script. Although one can used the tksqlite.tcl interface to modify *any* field in any database table, which can be dangerous, this is not true when using the HPSim Python package discussed below. This feature limits one from using scripts to easily corrupt the design and configuration of the linac in the database. Since HPSim is configured to allow the linac parameters, i.e. EPICS PV's, to control the state of the machine in the simulation, then these PV elements are adjustable in the database. RF calibration constants, space-charge compensation factors and circular and rectangular apertures size and in/out fields are also writable. All others fields are read only.

Beamline Elements

Buncher (buncher)

The buncher element is modeled as a single gap rf cavity, which is typically used in LEBT's and MEBT's to bunch the beam. It consists of the following additional fields as given in Table 4.

Table 4: Additional database fields specific to buncher element.

Field name	Type, Unit	Description
phase_model	dble, rad	cavity voltage phase
phase_offset_cal	dble, deg	phase offset to shift PV phase set point to model_phase
voltage_model	dble, MV	net voltage in RF gap
c0_cal – c4_cal	dble	polynominal coefficients for converting Amplitude set point PV value to cavity voltage
frequency_model	dble, MHz	frequency of cavity rf

aperture_model	dble, m	aperture radius of RF gap
amplitude_channel	int	ID of epics_channel for amp set point
phase_channel	int	ID of epics_channel for phase set point
on_off_channel	int	ID of epics_channel for rf on/off

An example of the buncher table for LANSCE H- LEBT is shown in Figure 2.

Table Edit [main.buncher]

id	name	view_index	on_off	phase_model	phase_offset_cal	voltage_model	c0_cal	c1_cal	c2_cal	c3_cal	c4_cal
1	TBDB01	49.0	0	-1.5707963267949	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	TBDB02	51.0	1	2.96705972839036	-100.0	0.00398831622421552	1.07190247	0.07971817	-0.00032347	3.7308e-6	-1.5969e-8
3	TDDB01	91.0	1	5.52313456050885	89.2523	0.00965338223330795	2.76867036	0.22313134	-0.0010578	1.0405e-5	-3.8059e-8

frequency_model	aperture_model	amplitude_channel	phase_channel	on_off_channel	model_index	model_type
201.25	0.01	29	30	31	57	buncher
201.25	0.01	23	24	25	59	buncher
201.25	0.01	26	27	28	105	buncher

Figure 2: Buncher table entries from tbtd.db for LANSCE H- LEBT.

Circular Aperture (caperture)

The circular aperture element represents a round aperture of zero length along the beam line. It functions to immediately remove particles, i.e. sets lost array value to this element sequence number in beam line. Space-charge is not applied at this element. The additional fields required for the circular aperture are shown in Table 5.

Table 5: Additional database fields specific to caperture element.

Field name	**Type, Unit**	**Description**
aperture_model	float, m	aperture radius
in_out_model	int, 0 or 1	1: in; 0: out

An example of the caperture table for LANSCE H- LEBT is shown in Figure 3.

Table Edit [main.caperture]

id	name	view_index	aperture	in_out_model	model_index	model_type
1	TBBA01	19.0	0.0254	0	21	caperture
2	TBBA02	27.0	0.0254	0	31	caperture
3	TBBA04	53.0	0.01	1	61	caperture
4	TDBA01	87.0	0.01	1	101	caperture

Figure 3: Caperture table entries from tbtd.db for LANSCE H- LEBT.

Diagnostic (diagnostic)

The beam diagnostic element represents a zero length placeholder along the beam line. It was created to allow the user to insert additional start/stop points in the simulation for comparison of output with actual beam diagnostics measurements in the linac. Space charge is not applied at this element. The additional fields required for the diagnostic are shown in Table 6. The monitor field is used by the online simulator to determine if beam information should be generated at each location.

Table 6: Additional database fields specific to diagnostic element.

Field name	Type, Unit	Description
diag_type	text	EM for emittance, HP for harp, CM for current monitor
monitor	int	1: in; 0: off

An example of the diagnostic table for LANSCE H- LEBT is shown in Figure 4.

Table Edit [main.diagnostics]

id	name	view_index	diag_type	monitor	model_index	model_type
1	TBEM01	9.0	EM	0	11	diagnostics
2	TBEM01COL	17.0	HP	0	19	diagnostics
3	TBEM02	29.0	EM	0	33	diagnostics
4	TBEM02COL	35.0	HP	0	39	diagnostics
7	TBEM03	46.0	EM	0	52	diagnostics
8	TBEM03COL	56.0	HP	0	66	diagnostics
9	TBEM04	70.0	EM	0	82	diagnostics
10	TBEM04COL	76.0	HP	0	88	diagnostics
11	TDEM01	96.0	EM	0	112	diagnostics
12	TDEM01COL	106.0	HP	0	122	diagnostics

Figure 4: Diagnostic table entries from tbtd.db for LANSCE H- LEBT.

Dipole (dipole)

The dipole element represents a standard large angle (>> millirads) bend where edge effects are included and is used in defining the central trajectory of the beam line. This element produces a bend in the +x direction, so bends in other directions will require the use of the *rotation* element before and after the dipole. Refer to the PARMILA manual for details about the constants used and their definitions. Space charge is applied once at the middle of this element. The additional fields required for the dipole are shown in Table 7.

Table 7: Additional database fields specific to dipole element.

Field name	Type, Unit	Description
rho_model	dble, m	radius of curvature for the reference particle
angle_model	dble, rad	bend angle
half_gap_model	dble, m	half the gap distance between pole tips
edge_angle_1	dble, rad	entrance edge angle
edge_angle_2	dble, rad	exit edge angle
k1_model	dble	first field integral
k2_model	dble	second field integral
field_index_model	dble	magnetic field index
Kenergy_model	dble, MeV	reference energy of bend
channel	int	EPICS PV associated with power supply that drives the dipole

An example of the dipole table for LANSCE H- LEBT is shown in Figure 5.

Table Edit [main.dipole]

id	name	view_index	rho_model	angle_model	half_gap_model	edge_angle1_model	edge_angle2_model
1	TBBM01	38.0	0.56634	1.41371669411	0.0381	0.445058959258554	0.445058959258554
2	TDBM01	85.0	1.6372	0.1570796326	0.03963	0.150237942011672	0.0

k1_model	k2_model	field_index_model	kenergy_model	channel	model_index	model_type
0.45	2.8	0.0	0.75		44	dipole
0.243	2.801	0.0	0.75		99	dipole

Figure 5: Dipole table entries from tbtd.db for LANSCE H- LEBT.

Drift space (drift)

The drift element represents an empty space in the beamline between other elements. For the purposes of these simulations, it is broken up into intervals defined in the space-charge API so that a space-charge kick can be applied to the beam at each interval. It also has a circular aperature radius that acts to remove particles at larger radii from futher calculation. The additional fields required for the drift are shown in Table 8.

Table 8: Additional database fields specific to a drift element.

Field name	Type, Unit	Description
length_model	dble, m	length of drift space
aperture_model	dble, m	inner radius of drift space pipe

An example of the drift table for a portion of the LANSCE H- LEBT is shown in Figure 6.

Table Edit [main.drift]

id	name	view_index	length_model	aperture_model	model_index	model_type
1	TBDR01	1.0	0.0524	0.0254	1	drift
2	TBDR02	3.0	0.05695	0.0254	3	drift
3	TBDR03	5.0	0.05695	0.0254	5	drift
4	TBDR04	7.0	0.11621	0.0254	7	drift
5	TBDR05	8.0	0.08201	0.0254	10	drift
7	TBDR06	10.0	0.05903	0.0254	12	drift
8	TBDR07	12.0	0.07273	0.0254	14	drift
9	TBDR08	14.0	0.07273	0.0254	16	drift
10	TBDR09	16.0	0.0421	0.0254	18	drift

Figure 6: Drift table entries from tbtd.db for LANSCE H- LEBT.

Quadrupole magnet (quad)

The quad element represents a quadrupole focusing magnet in the beam line. The model used is a hard-edge device with an effective length and a constant gradient over that length. A space-charge kick is applied to the beam once in the middle of the element. It also has a circular aperature radius that acts to remove particles at larger radii from futher calculation. The additional fields required for the quad are shown in Table 9.

Table 9: Additional database fields specific to a quad element.

Field name	Type, Unit	Description
length_model	dble, m	effective length of quad magnet
gradient_model	dble, T/m	field gradient over effective length
aperture_model	dble, m	inner radius of magnet
polarity_design	int	+1-> horizontally focusing pos-polarity beam, -1-> horizontally defocusing pos-polarity beam
family_cal	int	id number of calibration curve in quad family element, used to convert EPICS PV power supply value to quad gradient
channel	int	id number of EPICS PV channel associated to this magnets power supply

An example of the quad table for a portion of the LANSCE H- LEBT is shown in Figure 7.

Table Edit [main.quad]

id	name	view_index	monitor	gradient_model	length_model	aperture_model	family_cal	shunt_cal	polarity_design	channel	model_index	model_type
1	TBQL01V1	2.0	0	1.2116696080242	0.1022	0.0254	1	0.0	1	1	2	quad
2	TBQL01V2	4.0	0	-3.6325072643175	0.1022	0.0254	2	0.0	-1	2	4	quad
3	TBQL01V3	6.0	0	2.96055295675425	0.1022	0.0254	3	0.0	1	3	6	quad

Figure 7: Quad table entries from tbtd.db for LANSCE H- LEBT.

Rectangular Aperture (raperture)

The rectangular aperture element represents an adjustable-size aperture of zero length along the beam line. It functions to immediately remove particles, i.e. sets lost array value to this element sequence number in beam line. Space-charge is not applied at this element. The additional fields required for the rectangular aperture are shown in Table 10.

Table 10: Additional database fields specific to raperture element.

Field name	Type, Unit	Description
aperture_xl_model	dble, m	left aperture dist from axis
aperture_xr_model	dble, m	right aperture dist from axis
aperture_yt_model	dble, m	top aperture dist from axis
aperture_yb_model	dble, m	bottom aperture dist from axis
in_out_model	int, 0 or 1	1: in; 0: out

An example of the raperture table for LANSCE H- LEBT is shown in Figure 8.

Table Edit [main.raperture]

name	view_index	aperture_xl_model	aperture_xr_model	aperture_yt_model	aperture_yb_model	in_out_model	model_index	model_type
TBFJ01	40.0	0.0	0.0	0.0	0.0	0	48	raperture
TBFJ02	66.0	0.0	0.0	0.0	0.0	0	79	raperture
TBFJ03	78.0	0.00147477110418	0.00147477110418	0.00431303729257	0.00431303729257	0	94	raperture

Figure 8: Raperture table entries from tbtd.db for LANSCE H- LEBT.

Rotation (rotation)

The rotation element is used to perform a rotation of the particles coordinates about the z-axis at some point in the beam line. This element is typically used in conjunction with the dipole element to provide bends in directions other than the +x direction. This element has zero length along the beam line. Space-charge is not applied at this element. The additional fields required for the rotation are shown in Table 11.

Table 11: Additional database fields specific to rotation element.

Field name	**Type, Unit**	**Description**
angle_model	float, rad	angle of rotation about z-axis

An example of the roation table for LANSCE H+ LEBT is shown in Figure 9. No rotation elements are required in the H- LEBT defined in lansce-tbtd.db, which is why the H+ LEBT example is used.

Table Edit [main.rotation]

id	name	view_index	angle_model	model_index	model_type
1	TABM01-BEFORE	29.0	3.14159265358979	29	rotation
2	TABM01-AFTER	31.0	3.14159265358979	31	rotation
3	TDBM01-BEFORE	77.0	3.14159265358979	76	rotation
4	TDBM01-AFTER	79.0	3.14159265358979	78	rotation

Figure 9: Rotation table entries from tatd.db for LANSCE H+ LEBT.

Space-charge compensation (spch_comp)

The space-charge compensation element is used to modify the effective current of the beam along the beam line. This element is typically used when some degree of neutralization occurs, e.g. in a LEBT with modest residual gas pressure. The compensation represents the fraction of the actual beam current that is used in the space-charge from this point forward in the calculation. If used for example in a LEBT to reflect space-charge neutralization of the beam, it should be reset prior to entrance into an RF accelerator to reflect no further neutralization. This element has zero length along the beam line. Space-charge is not applied at this element. The additional fields required for the spch_comp are shown in Table 12.

Table 12: Additional database fields specific to spch_comp element.

Field name	**Type, Unit**	**Description**
fraction_model	float	fraction of beam current to be used in space-charge calculation from this point forward in the beam line.

An example of the spch_comp table for LANSCE H- LEBT is shown in Figure 10.

Table Edit [main.spch_comp]

id	name	view_index	fraction_model	model_index	model_type
1	spch_comp_1	0.5	1.0	0	spch_comp
2	spch_comp_2	9.2	1.0	12	spch_comp
3	spch_comp_3	29.2	1.0	35	spch_comp
4	spch_comp_4	46.2	1.0	55	spch_comp
5	spch_comp_5	70.2	1.0	86	spch_comp
6	spch_comp_6	96.2	1.0	117	spch_comp
7	spch_comp_7	109.0	1.0	129	spch_comp

Figure 10: Spch_comp table entries from tbtd.db for LANSCE H- LEBT.

Steering magnet (steerer)

The steering magnet element is used to apply a small angle correction to the transverse angle coordinates, i.e. x', y', of the particles in the beam bunch. This element has zero length along the beam line. Space-charge is not applied at this element. The additional fields required for the spch_comp are shown in Table 13.

Table 13: Additional database fields specific to steerer element.

Field name	Type, Unit	Description
bl_h_model	dble, T*m	Integral Bdl for horizontal steering magnet in the beam line.
bl_v_model	dble, T*m	Integral Bdl for vertical steering magnet in the beam line.

An example of the steerer table for LANSCE H- LEBT is shown in Figure 11.

Table Edit [main.steerer]

id	name	view_index	bl_h_model	bl_v_model	channel	model_index	model_type
1	TBSM01X	7.6	0.0	0.0		8	steerer
2	TBSM01Y	7.7	0.0	0.0		9	steerer
3	TBSM02X	20.6	0.0	0.0		24	steerer
4	TBSM02Y	20.7	0.0	0.0		25	steerer

Figure 11: Steerer table entries from tbtd.db for LANSCE H- LEBT.

RF Structures

Drift-tube Linac, DTL

To model a drift-tube linac in HPSim requires the use of several elements. The rf_module, rf_gap and transit_time_factor elements are required to build one or more dtl tanks in the model. The physics model used is the same as in the PARMILA code, so the reader should refer to that document for specific details.

RF module (rf_module)

The rf_module element is used to represent an rf accelerating structure powered by a single rf amplifier, e.g. gridded tube or klystron. This element contains operating phase and overall amplitude of the cavity field and connections to EPICS channels that control the tank rf. It is not a beam line element per se, but a container for all of the rf gaps that comprise the tank. It consists of the following additional fields as given in Table 14.

Table 14: Additional database fields specific to rf_module element.

Field name	Type, Unit	Description
on_off	int	RF state: on(1) or off(0)
phase_shift_tmp	dble, rad	intermediate module phase result
phase_offset_cal	dble, deg	phase offset to shift PV phase set point by to get model_phase
operating_amplitude_fraction_tmp	dble	fraction of design amplitude
voltage_tmp	dble, MV	intermediate module voltage result
voltage_cal	dble	voltage calibration factor
amplitude_scale_cal	dble	scale factor applied to convert EPICS PV for module amplitude to get cavity field
amplitude_design	dble, MV/m	E0 design amplitude for module
frequency_design	dble, MHz	frequency of rf module
cell_length_betalambda_design	dble	length of unit cell in beta*lambda
diode	int	ID number of diode_table diode used to detect amplitude of rf
amplitude_channel	int	ID of epics_channel for amp set point
phase_channel	int	ID of epics_channel for phase set point
phase_master_channel*	int	ID of epics_channel for rf_ph_master
delay_channel	int	ID of epics_channel for rf in-time or delayed

* Present only when used by the rf_module (see ccl.db file).

An example of the rf_module table for LANSCE DTL is shown in Figure 12.

Table Edit [main.rf_module]

id	name	view_index	on_off	phase_shift_tmp	phase_offset_cal	operating_amplitude_fraction_tmp	voltage_tmp
1	01LN	0.0	1.0	208.0	0.0	0.9807676406088888	0.5232
2	02LN	66.0	1.0	-147.0	-100.0	0.9597187418023094	0.8136
3	03LN	202.0	1.0	214.855331447	150.855331447	0.9629609290248885	0.792
4	04LN	282.0	1.0	206.0	50.0	0.9704557329737862	0.81119999999

Table Edit [main.rf_module]

voltage_cal	amplitude_scale_cal	amplitude_tilt_cal	amplitude_design	frequency_design	cell_length_betalambda_design
0.012	15.3742087861	0.0	1.606	201.25	1.0
0.012	15.7	0.0	2.4	201.25	1.0
0.012	15.4116891833	0.0	2.4	201.25	1.0
0.012	15.8	0.0	2.4	201.25	1.0

Table Edit [main.rf_module]

diode_id	amplitude_channel	phase_channel	delay_channel	model_type
1	13	14	15	rf_module
2	16	17	18	rf_module
3	19	20	21	rf_module
4	22	23	24	rf_module

Figure 12: Rf_module table entries from dtl.db for LANSCE DTL.

RF gap (rf_gap)

The rf_gap element is used to represent a radio-frequency accelerating gap within a bigger structure, e.g. DTL or CCL. This element contains the information that defines the geometry and electromagnetic properties of the gap and the final model values that are used in the simulation. These gaps must reside in an rf_module, as described above, in order to be controlled by the appropriate EPICS PV for phase and amplitude of the cavity field. Space-charge is applied once at the center of an rf_gap. It consists of the following additional fields as given in Table 15.

Table 15: Additional database fields specific to rf_gap element.

Field name	**Type, Unit**	**Description**
frequency_model	dble, MHz	rf gap frequency
length_model	dble, m	length of rf gap where E0 is calc
amplitude_model	dble, MV/m	cell E0
amplitude_tmp	dble	intermediate amplitude result
amplitude_tilt_tmp	dble	intermediate field tilt result
amplitude_design	dble, MV/m	design amplitude, E0, of cell
ref_phase_model	dble, rad	cell rf phase
ref_phase_design	dble, rad	cell rf design phase
beam_phase_shift_model	dble, rad	cell rf phase shift
beta_center_model	dble	normalized velocity of design particle in gap
dg_model	dble, m	offset between electrical and geometric center of rf gap
t_model	dble	transit-time factor, T, of gap
tp_model	dble	transit-time factor, dT/dk of gap
sp_model	dble	transit-time factor, dS/dk of gap
module_id	int	ID number of associated rf_module
tank_id	int	ID number of associated tank_id
model_type	text	“dtl_gap” for DTL or “ccl_gap” for CCL structure cells

An example of the rf_module table for LANSCE DTL is shown in Figure 13.

Table Edit [main.rf_gap]

id	name	view_index	frequency_model	length_model	amplitude_model	amplitude_tmp	amplitude_tilt_tmp	amplitude_design
1	01RG01	2.0	201.25	0.034668	1.706344287971461	1.706344287971461	0.0	1.73980483992326
2	01RG02	4.0	201.25	0.028315	1.5918448368266422	1.5918448368266422	0.0	1.6230601122182
3	01RG03	6.0	201.25	0.029596	1.4874716964346582	1.4874716964346582	0.0	1.51664026711892

Table Edit [main.rf_gap]

ref_phase_model	ref_phase_design	beam_phase_shift_mod	energy_out_model	beta_center_model	dg_model
-0.41175726533199664	-0.453785605518526	3.630284844148205	0.812832291142269	0.040679080483477	0.00257870984928687
-0.41175726533199664	-0.453785605518526	3.630284844148205	0.879743517166122	0.0423365904055705	0.000777192511924818
-0.41175726533199664	-0.453785605518526	3.630284844148205	0.950893056890789	0.0440316389241208	0.000167037995929047

Table Edit [main.rf_gap]

t_model	tp_model	sp_model	module_id	tank_id	cell_id	model_index	model_type
0.701308267780434	0.072820714657826	0.0477220034324026	1	1	1	112	dtl_gap
0.727303208803338	0.0709682978576746	0.0484978827042139	1	1	2	114	dtl_gap
0.742126074390865	0.0691716731109034	0.0492370341923739	1	1	3	116	dtl_gap

Figure 13: Rf_gap table entries from dtl.db for LANSCE DTL.

Transit-time factor (transit_time_factor)

The transit-time factors are used in the rf gap model to account for the affect of a time-varying field on the acceleration and defocusing of particles in the rf gap. This is the same physics model/approximation that PARMILA uses, so the reader is referred to the PARMILA manual for further explanation. SUPERFISH can be used to generate the TTF versus beta as required. The user must then fits these data to a polynomial to obtain the coefficients to place into the tables. Each entry is not a beam line element but is provided for each rf_gap or ccl_tank entry in the linac. It's name must be identical with the rf gap or ccl_tank that it is associated with. It consists of the following additional fields as given in Table 16.

Table 16: Additional database fields specific to transit_time_factor table.

Field name	Type, Unit	Description
module_id	int	ID number of associated rf_module
tank_id	int	ID number of associated tank
beta_g	dble	normalized design velocity of associated rf gap
beta_min	dble	minimum beta that TTF entries will are valid for
ta0	dble	polynominal exp. coeff describing TTF T(beta)
ta1	dble	“ “ “ “ “ “
ta2	dble	“ “ “ “ “ “
ta3	dble	“ “ “ “ “ “
ta4	dble	“ “ “ “ “ “
ta5	dble	“ “ “ “ “ “
sa0	dble	polynominal exp. coeff describing TTF S(beta)
sa1	dble	“ “ “ “ “ “
sa2	dble	“ “ “ “ “ “
sa3	dble	“ “ “ “ “ “
sa4	dble	“ “ “ “ “ “
sa5	dble	“ “ “ “ “ “

An excerpt from the transit_time_factor table for LANSCE DTL is shown in Figure 14.

Table Edit [main.transit_time_factor]

	id	name	module_id	tank_id	beta_g	beta_min
	1	01RG01	1	1	0.0434282647964	0.0379997316969
	2	01RG02	1	2	0.0426509020117	0.0373195392602
	3	01RG03	1	3	0.0435108344187	0.0369842092559

Table Edit [main.transit_time_factor]

	ta0	ta1	ta2	ta3	ta4	ta5
	-3.44312389619	367.714397719	-14368.6202323	302792.548644	-3311809.30427	14775243.0784
	-2.2712553929	245.483276262	-8895.96859878	177755.18157	-1874735.97444	8160319.99282
	-0.505538711209	40.5735174235	683.450864134	-46310.2762171	743060.716185	-4054100.15905

Table Edit [main.transit_time_factor]

	sa0	sa1	sa2	sa3	sa4	sa5
	0.778462792675	-17.2172818666	1233.14293132	-45144.7417055	689895.025677	-3814964.6654
	0.541028893121	16.1834049036	-620.291027311	2627.83877523	99203.6047446	-962476.918338
	-0.821685171921	177.724003542	-8274.55275347	183186.680165	-2022659.14565	8981180.96973

Figure 14: Transit_time_factor table entries from dtl.db for LANSCE DTL.

Coupled-cavity Linac, CCL

To model a coupled-cavity linac in HPSim also requires the use of several elements. The rf_module, rf_gap, ccl_tank, and transit_time_factor elements are required to build one or more ccl modules in the model. The physics model used is the same as in the PARMILA code, so the reader should refer to that document for specific details. A CCL module comprises one or more tanks. A CCL tank comprises many identical cells, i.d. same design beta. The rf_gap element and the rf_module and transit_time_factor tables have been described in the preceding sections. The new table required for the ccl is the ccl_tank.

CCL tank (ccl_tank)

The ccl_tank table entry is used to represent a ccl tank that contains the same length cells, which have a fixed phase shift between them. Multiple CCL tanks are typically further coupled together and driven by a single rf amplifier. It is not a beam line element per se, but a container for all of the rf gaps that comprise the tank. It consists of the following additional fields as given in Table 17.

Table 17: Additional database fields specific to rf_module element.

Field name	Type, Unit	Description
module_id	int	ID number of associate rf_module
tank_id	int	ID number of tank within the rf_module
avg_phase_tmp	dble, rad	temporary tank phase
avg_phase_design	dble, rad	design phase of tank
phase_offset_structure	dble, deg	phase offset between tanks

cell_length_design	dble, m	not used
t_design	dble	transit time factor, T, at design beta
tp_design	dble	transit time factor, T', at design beta
sp_design	dble	transit time factor, S', at design beta
model_type	text	ccl_tank

An example of the rf_module table for LANSCE DTL is shown in Figure 15.

Table Edit [main.ccl_tank]

id	name	view_index	module_id	tank_id	avg_phase_tmp	avg_phase_design
1	05TK01	0.0	5	1	-0.48139329732913866	-0.481393296205213
2	05TK02	44.0	5	2	-0.5895309486849826	-0.5895309478072128
3	05TK03	88.0	5	3	-0.6424632338466219	-0.6424632330621006
4	05TK04	132.0	5	4	-0.6800297749090247	-0.68002977418303
5	06TK01	177.0	6	1	-0.5876788940251546	-0.5876788931438572
6	06TK02	219.0	6	2	-0.5706339746015459	-0.5706339736868344
7	06TK03	261.0	6	3	-0.6270634751218895	-0.627063474311652
8	06TK04	303.0	6	4	-0.6701284125141674	-0.6701284117732879

phase_offset_structure	cell_length_design	t_design	tp_design	sp_design	model_type
0.0	0.0	0.0	0.0	0.0	ccl_tank
180.0	0.0	0.0	0.0	0.0	ccl_tank
0.0	0.0	0.0	0.0	0.0	ccl_tank
180.0	0.0	0.0	0.0	0.0	ccl_tank
0.0	0.0	0.0	0.0	0.0	ccl_tank
180.0	0.0	0.0	0.0	0.0	ccl_tank
0.0	0.0	0.0	0.0	0.0	ccl_tank
180.0	0.0	0.0	0.0	0.0	ccl_tank

Figure 15: Ccl_tank table entries from ccl.db for LANSCE CCL.

Miscellaneous Tables

There are a few tables that does not belong to either beam transport elements or rf structures, etc., and so are captured in the miscellaneous category. Analog LLRF control system typically uses a diode to provide an amplitude signal. The diode response curve table is discussed below. Any beam line device that has a connection to the control system requires the EPICS channels, which are placed in the epics_channel table discussed below. Also, quad magnets that require their excitation curve to be incorporated into a quad family table is also discussed below.

Diode calibration (diode)

The diode table is used to hold rf_diode detector calibration curves. Each diode calibration curve is represented with a set of polynominal coefficients. The calibrations are used to convert a detected rf signal level to an actual cavity amplitude. This approach is used in older style analog LLRF systems. More modern controllers that digitize the rf signal get the amplitude without the use of diodes, so this table would not be required. It consists of the following additional fields as given in Table 18.

Table 18: Additional database fields specific to diode table.

Field name	**Type, Unit**	**Description**
c0_cal	dble	poly coef for diode cal
c1_cal	dble	poly coef for diode cal
c2_cal	dble	poly coef for diode cal
c3_cal	dble	poly coef for diode cal
c4_cal	dble	poly coef for diode cal

An example of the diode table for LANSCE DTL is shown in Figure 16.

Table Edit [main.diode]

id	name	c0_cal	c1_cal	c2_cal	c3_cal	c4_cal
1	diode1	0.0195	0.1805	-0.0798	0.0977	-0.0485
2	diode2	0.0198	0.1854	-0.0925	0.1175	-0.0593
3	diode3	0.02218	0.19365	-0.10372	0.13623	-0.0717
4	diode4	0.0217	0.18171	-0.08447	0.10862	-0.05563

Figure 16: Diode table entries from dtl.db for LANSCE DTL.

EPICS Channels (epics_channel)

The epics channel table contains the PV channels used to control beam line devices that reside in the same database file. Therefore every database file containing active beam line elements should have an epics_channel table. It consists of the following additional fields as given in Table 18.

The "rf_phase_master" type is used to represent a phase shift associated with a reference signal that may be common to all accelerating modules in a particular section of the linac. For example, at LANSCE, the MRPH channel representa a phase shift of the 805 MHz rf reference used in the CCL with respect to the 201.25 MHz rf reference used in the LEBTs and DTL. There should at most be one rf_phase_master in a database file. The underlying db triggers statements must then be constructed to take the requisite action. N.B. For long linacs, changes to this channel can introduce a noticeable slow down in the update of the model in pinned memory.

Table 19: Additional database fields specific to epics_channel table.

Field name	**Type, Unit**	**Description**
id	int	unique ID number used in other tables to reference PV
lcs_name	text	text string with exact name of EPICS PV

value_type	text	text string indicating the type of quantity that the PV refers to e.g. "DVM", "current", "buncher_amp", "buncher_ph", "buncher_on_off", "AMP", "rf_amp", "rf_phs", "delay", "rf_ph_master".
value	dble	value that EPICS PV has in db
value_txt	text	when appropriate text representation of PV value else "NA" if not applicable

An example of the epics_channel table for LANSCE DTL is shown in Figure 17.

Table Edit [main.epics_channel]

id	lcs_name	value_type	value	value_txt	update_time
1	01QM000I01	current	360.204	NA	2014-09-11 14:21:40
2	01MP001I01	current	602.825	NA	2013-07-05 09:36:38
3	01MP002I01	current	409.019	NA	2013-07-05 09:36:45
4	01MP003I01	current	303.36	NA	2013-07-05 09:36:51
5	02MP001I01	current	373.18	NA	2014-09-10 16:58:34
6	02MP002I01	current	324.22	NA	2013-07-05 09:37:00
7	02MP003I01	current	366.837	NA	2014-05-22 08:30:57
8	02MP004I01	current	349.18	NA	2013-07-05 09:37:14
9	02MP005I01	current	332.07	NA	2013-07-05 09:37:20
10	02MP006I01	current	319.74	NA	2014-05-22 08:31:11
11	03MP001I01	current	226.312	NA	2014-05-22 08:31:21
12	04MP001I01	current	218.078	NA	2014-05-22 08:31:26
13	01JS001D01	rf_amp	43.6	NA	2015-10-22 14:43:40
14	01JS001D02	rf_ph	208.0	NA	2015-10-22 14:43:40
15	01TM001L01	delay	0.0	In Time	2016-02-28 12:12:32
16	02JS001D01	rf_amp	67.8	NA	2015-10-22 14:43:40
17	02JS001D02	rf_ph	-47.0	NA	2015-10-22 14:43:40
18	02TM001L01	delay	0.0	In Time	2015-10-22 14:43:40
19	03JS001D01	rf_amp	66.0	NA	2015-10-22 14:43:40
20	03JS001D02	rf_ph	64.0	NA	2015-10-22 14:43:40
21	03TM001L01	delay	0.0	In Time	2015-10-22 14:43:40
22	04JS001D01	rf_amp	67.6	NA	2015-10-22 14:43:40
23	04JS001D02	rf_ph	156.0	NA	2015-10-22 14:43:40
24	04TM001L01	delay	0.0	In Time	2015-10-22 14:43:40

Figure 17: Epics_channel table entries from dtl.db for LANSCE DTL.

Quadrupole Calibration Curve (quad_family)

The quad_family table is used to hold quadrupole magnet calibration curves used to convert power supply or shunt readings to quad gradient. A single entry can be used for one or more quads of the same design. This is required when the EPICS PV for a quad is actually the power supply current or a shunt reading and not the magnet gradient. The table contains the following additional fields as given in Table 20.

Table 20: Additional database fields specific to quad_family table.

Field name	Type, Unit	Description
l_eff_cal	dble, m	effective length of quadrupole
a0_cal	dble	poly coef for quad cal
a1_cal	dble	poly coef for quad cal
a2_cal	dble	poly coef for quad cal
a3_cal	dble	poly coef for quad cal
a4_cal	dble	poly coef for quad cal

An example of the quad_family table for LANSCE DTL is shown in Figure 18.

Table Edit [main.quad_family]

id	name	l_eff_cal	a0_cal	a1_cal	a2_cal	a3_cal	a4_cal
0	quad_type_0	0.02483	0.003447	0.0029138	4.5509e-6	-1.1115e-8	5.5871e-12
1	quad_type_1	0.03522	0.0090526	0.0046231	-1.5358e-6	6.2635e-9	-7.6251e-12
2	quad_type_2	0.04774	0.0028514	0.0064909	-3.4515e-6	1.0784e-8	-1.048e-11
3	quad_type_3	0.06234	0.010348	0.0081983	-4.4683e-7	1.5503e-9	-1.8316e-12
4	quad_type_4	0.08889	1.5152e-5	0.0052911	-1.158e-8	2.5253e-12	0.0
5	quad_type_5	0.176	-0.0045455	0.0047937	-2.1212e-8	2.0202e-11	0.0

Figure 18: Quad_family table entries from lansce-dtl.db for LANSCE DTL.

Python Packages

The HPSim code is accessed through Python/C API's (Application Programming Interfaces) in the Python programming language. The API's are compiled into a share library that can be imported into Python. However, additional high-level Python packages that are shown below were created to give the user more convenient access to the functionality of the shared library plus many additional features, e.g. plotting. The packages are listed in Table 20. Below are Pydoc listings of each of these packages showing the classes and functions available from each one. HPSim also requires that the Numpy, Scipy and Matplotlib packages are installed. These packages should reside in your "pylib" directory. The release may contain specific numbered versions. If that is true, then you must make a copy of or alias with the generic names as shown immediately below.

Table 21: HPSim Python packages

Name	Description
hpsim.py	Interface to HPSim code and it's classes
lcsutil.py	Functions for getting LCS related info useful with HPSim
nputil.py	Functions for manipulating Numpy data arrays
sqldb.py	Functions for accessing linac sql db.

Hpsim.py

index

hpsim_v9

```
Python wrapper for HPSim.so funtions. This version utilizes Numpy arrays
and adds some functionality. The DBConnection, Beamline and SpaceCharge objects
are hidden in the classes and used implicitly when calling functions that
depend upon them.
```

Modules

matplotlib.cm	matplotlib.mlab	os
lcsutil	matplotlib	matplotlib.pyplot
math	numpy	sys

Classes

Beam
BeamLine
BeamPlot
DBConnection
DBState

[DistPlot](#)
[Distribution](#)
[Simulator](#)
[SpaceCharge](#)

class **Beam**

```
An hpsim class for manipulating beams as Numpy arrays
```

Methods defined here:

__init__(self, **args)

```
Creates an instance of the beam class

Args:
   keywords are required and lowercase
   where

   file (string): Input file name containing beam distribution

    or

   mass (double): mc^2 of particle mass (MeV)
   charge (double): q/|e|
   current (double): bunch peak current(Amps)
   num (int): number of macro particles

Attributes:
   initial_current(double): Initial beam current (A)
   initial_frequency(double): Initial freq (MHz)
   initial_size(int): Initial number of macroparticles in beam distr

Returns:
    Returns a beam class object

Examples:
    Beam(file = filename)

     or

    Beam(mass = particle_mass, charge = particle_charge,
               current = beam_current, num = number of particles
)
```

apply_cut(self, axis, minval, maxval)

```
Remove particles from beam by apply cuts along 'x', 'y', 'p' or 'w'
```

get_avg(self, var, mask=None)

```
Return average of beam coordinates after optional mask applied.

Arguments:
   var (str): Either 'x', 'xp', 'y', 'yp', 'phi', 'w', or 'losses'
   mask (Numpy vector, optional): mask used to select particles
```

get_avg_phi(self, option='absolute')

```
Return average phi value of beam in deg
```

get_avg_w(self)

```
Return average w value of beam in MeV
```

get_avg_x(self)

```
Return average x value of beam in cm
```

get_avg_y(self)

```
Return average y value of beam in cm
```

get_betagamma(self, mask=None)

```
Return value of beta*gamma of beam.

Arguments:
   mask (Numpy vector, optional): mask used to select particles
```

get_betalambda(self, mask=None)

```
Return value of beta*lambda of beam.

Arguments:
   mask (Numpy vector, optional): mask used to select particles
```

get_charge(self)

```
Return charge of beam in q/|e|
```

get_coor(self, var, mask=None)

```
Return vector of macro particle coordinates (in USER_UNITS)
after optional mask is applied.

Arguments:
   var (str): Either 'x', 'xp', 'y', 'yp', 'phi', 'w', or 'losses'
   mask (Numpy vector, optional): mask used to select particles
```

get_current(self, mask=None)

```
Return beam current in user units of beam
```

get_distribution(self, option='all')

```
Returns a list of Numpy vectors containing the beam coordinates in
user units x, xp, y, yp, phi, w, loss = beam.get_distribution()

Argument:
   option (str): 'good', 'bad' or 'all'=default
```

get_emittance_x(self)

```
Return rms x emittance of beam cm*mr
```

get_emittance_y(self)

```
Return rms y emittance of beam in cm*mr
```

get_emittance_z(self)

```
Return rms z emittance of beam in Deg*MeV
```

get_frequency(self)

```
Return beam frequency in MHz
```

get_good_mask(self, mask=None)

```
Returns indices of particles not lost.

Arguments:
   mask (Numpy vector, optional): mask used to select particles
```

get_initial_size(self)

```
Returns initial number of beam macro particles
```

get_intersection_mask(self, mask1, mask2)

```
Returns the mask that results from the intersection of two masks.
Arguments:
   mask1 (Numpy vector): mask with condition 1 used to select partic
les
   mask2 (Numpy vector): mask with condition 2 used to select partic
les
```

get_loss_num(self)

```
Return number of lost particles
```

get_losses(self)

```
Return Numpy array of loss condition of macro particles
```

get_lost_mask(self, mask=None)

```
Returns indices of particles not lost.

Arguments:
   mask (Numpy vector, optional): mask used to select particles
```

get_mask_with_limits(self, var, lolim, uplim=None)

```
Creates a a mask, i.e. a Numpy vector of a list of indices, based up
```

```
on
variable x, xp, y, yp, phi, w or losses above lower limit and below
optional upper limit. User units

Arguments:
   var (str): Either 'x', 'xp', 'y', 'yp', 'phi', 'w', or 'losses'
   lolim (double): lower limit, above which, particles are included
in mask
   uplim (double, optional): upper limit, below which, particles are
 included
   in mask.
```

get_mass(self)

```
Return mass of beam, mc^2, in MeV
```

get_nrms_emit(self, var, mask=None)

```
Return normalized rms emittance along specified axis, x, y or z.

Arguments:
   var (str): Either 'x', 'xp', 'y', 'yp', 'phi', 'w', or 'losses'
   mask (Numpy vector, optional): mask used to select particles
```

get_phi(self, option='all')

```
Return Numpy array phi coordinates (deg) of macro particles, option
= 'good',
'bad', 'all (default)
```

get_ref_phi(self)

```
Return the reference particle's phase in degree
```

get_ref_w(self)

```
Return the reference particle's energy in MeV
```

get_sig(self, var, mask=None)

```
Return sigma of beam coordinates after optional mask applied.

Arguments:
   var (str): Either 'x', 'xp', 'y', 'yp', 'phi', 'w', or 'losses'
   mask (Numpy vector, optional): mask used to select particles
```

get_sig_phi(self)

```
Return sigma phi of beam in deg
```

get_sig_w(self)

```
Return sigma w of beam
```

get_sig_x(self)

```
Return sigma x of beam in cm
```

get_sig_y(self)

```
Return sigma y of beam in cm
```

get_size(self, mask=None)

```
Return number of beam particles with or w/o mask applied
   Without a mask: Returns number of 'good' particles, i.e. not lost
   With a mask: Returns the length of the mask, i.e. number that sat
isfy mask.

Arguments:
   mask (Numpy vector, optional): mask used to select particles
```

get_twiss(self, var, mask=None)

```
Return Twiss parameters (a,b, unnormalized, rms e) for specified coo
rds x, y or z.

Arguments:
   var (str): Either 'x', 'xp', 'y', 'yp', 'phi', 'w', or 'losses'
   mask (Numpy vector, optional): mask used to select particles
```

get_urms_emit(self, var, mask=None)

```
Return unnormalized rms emittance along specified axis, x, y or z.

Arguments:
```

```
   var (str): Either 'x', 'xp', 'y', 'yp', 'phi', 'w', or 'losses'
   mask (Numpy vector, optional): mask used to select particles
```

get_w(self, option='all')

```
Return Numpy array w coordinates (MeV) of macro particles, option =
'good',
'bad', 'all (default)
```

get_x(self, option='all')

```
Return Numpy array of x coordinates (cm) of macro particles, option
= 'good',
'bad', 'all (default)
```

get_xp(self, option='all')

```
Return Numpy array xp coordinates (mr) of macro particles, option =
'good',
'bad', 'all (default)
```

get_y(self, option='all')

```
Return Numpy array y coordinates of (cm) macro particles, option = '
good',
'bad', 'all (default)
```

get_yp(self, option='all')

```
Return Numpy array yp coordinates of (mr) macro particles, option =
'good',
'bad', 'all (default)
```

print_results(self, mask=None)

```
Prints avg, sigma, alpha, beta, Eurms, Enrms for all coord of distr.

Arguments:
   mask (Numpy vector, optional): mask used to select particles
```

print_simple(self)

```
Print particle coordinates x, x', y, y', phi, w coordinates to scree
n
```

print_to(self, output_file_name)

```
Print particle coordinates x, x', y, y', phi, w coordinates to file
```

restore_initial_beam(self)

```
Restore initial beam distribution for next simulation.
```

restore_intermediate_beam(self)

```
Restore intermediate beam distribution for next simulation.
```

save_initial_beam(self)

```
Save initial beam distribution for later restore.
```

save_intermediate_beam(self)

```
Save intermediate beam distribution for later restore.
```

set_dc(self, alpha_x, beta_x, emittance_x, alpha_y, beta_y, emittance_y, delta_phi, synch_phi, synch_w, random_seed=0)

```
Creates DC beam using PARMILA input units set_waterbag
Arguments:
   alpha_x (double): x-plane Twiss alpha parameter
   beta_x (double): x-plane Twiss beta parameter (cm/radian)
   emittance_x (double): x-plane total emittance (cm * radian)
   alpha_y (double): y-plane Twiss alpha parameter
   beta_y (double): y-plane Twiss beta parameter (cm/radian)
   emittance_y (double): y-plane total emittance (cm * radian)
   alpha_z (double): z-plane Twiss alpha parameter
   beta_z (double): z-plane Twiss beta parameter (deg/MeV)
   emittance_z (double): z-plane total emittance (deg * MeV)
   delta_phi (double): half-width of phase distribution (deg)
   synch_phi (double): synchronous phase (deg)
   synch_w (double): synchronous energy (MeV)
   random_seed (int, optional): random seed for generating distribut
ion
```

set_distribution(self, x, xp, y, yp, phi, w, loss=None)

```
Creates beam distribution using vectors of coordinates (users units)
Arguments:
   x (Numpy vector double): x coordinates cm
   xp (Numpy vector double): xp coordinates mr
   y (Numpy vector double): y coordinates cm
   yp (Numpy vector double): yp coordinates mr
   phi (Numpy vector double): phi coordinates deg
   w (Numpy vector double): w coordinates MeV
   loss (Numpy vector int, optional): loss coordinate or 0-> good
```

set_frequency(self, frequency)

```
Set beam frequency in MHz
```

set_ref_phi(self, phi)

```
Set reference particle phase, degrees
```

set_ref_w(self, w)

```
Set reference particle energy, MeV
```

set_waterbag(self, alpha_x, beta_x, emittance_x, alpha_y, beta_y, emittance_y, alpha_z, beta_z, emittance_z, synch_phi, synch_w, frequency, random_seed=0)

```
Creates a 6D waterbag using PARMILA input units
Arguments:
   alpha_x (double): x-plane Twiss alpha parameter
   beta_x (double): x-plane Twiss beta parameter (cm/radian)
   emittance_x (double): x-plane total emittance (cm * radian)
   alpha_y (double): y-plane Twiss alpha parameter
   beta_y (double): y-plane Twiss beta parameter (cm/radian)
   emittance_y (double): y-plane total emittance (cm * radian)
   alpha_z (double): z-plane Twiss alpha parameter
   beta_z (double): z-plane Twiss beta parameter (deg/MeV)
   emittance_z (double): z-plane total emittance (deg * MeV)
   synch_phi (double): synchronous phase (deg)
   synch_w (double): synchronous energy (MeV)
   frequency (double): frequency (MHz)
   random_seed (option [int]): random seed for generating distributi
on
```

translate(self, axis, value)

```
Translate particle coordinates along specified axis by given value
```

class **BeamLine**

```
An hpsim class for defining and accessing the beamline
```

Methods defined here:

__init__(self)

```
Arguments: none
Returns:
   beamline object
```

new_get_element_names(self, start_element=None, end_element=None, elem_type=None)

```
Get list of elements in beamline from start_element to end_element
with option elem_type

Arguments:
   start_element(str): first element to retrieve from beamline
   end_element(str): last element to retrieve from beamline
   elem_type(str, optional): type of element, e.g. 'WS' to retrieve

Return:
   Python list of element names
```

print_out(self)

```
Print complete beamline listing from pinned memory, element by elem
ent
```

```
        for benchmarking
```
print_range(self, start_element, end_element)

```
        Print range of elements in pinned memory from start to end,
        for benchmarking

        Arguments:
           start_element (str): first element name in range to print
           last_element (str): last element name in range to print
```

Data and other attributes defined here:

beamline = ''

class **BeamPlot**

```
    An hpsim class for creating beam plots
```

Methods defined here:

__init__(self, nrow=1, ncol=1, hsize=None, vsize=None)

```
        Creates and instance of a matplotlib figure

        Arguments:
           nrow (int): number of rows in figure plotting grid
           ncol (int): number of columns in figure plotting grid
           hsize (double): horizontal size (inches) of figure
           vsize (double): vertical size (inches) of figure
```

clear(self)

```
        Clear plot figure
```

draw(self)

```
        Draw figure. Used in interactive mode
```

hist1d(self, u_vals, nplt, nbins=50, xlabel=None, limits=None, norm=1.0, ylog=False)

```
        Create 1d histogram of arbitrary vals in numpy array

        Arguments:
           u_vals (Numpy vector):values to plot
           nplt (int):which plot in figure grid, by row,
                        1 is upper left, nrow*ncol is lower right
           nbins (int, optional): number of bins to plot
           xlabel (str, optional): x-axis label
           limits (optional, [list of doubles]) [[xmin, xmax], [ymin, ymax]]
           norm (double, optional): normalization factor for plot
           ylog (logical, optional): True-> semilog plot, False(default)-
        > linear plot
```

hist1d_coor(self, coor, beam, mask, nplt, nbins=50, xlabel=None, limits=None, norm=1.0, ylog=False)

```
        Create a histogram style profile of beam coordinate

        Arguments:
           coor (str): coordinate to plot, either 'x', 'xp', 'y', 'yp', 'phi
        ', 'w' or 'losses'
           beam (beam object): beam object containing coordinates to plot
           mask (numpy vector): mask for filter beam prior to plotting
           nplt (int): which plot in figure grid, by row,
                        1 is upper left, nrow*ncol is lower right
           nbins (int, optional): number of bins to plot
           xlabel (str, optional): x-axis label
           limits (list, doubles, optional): [[xmin, xmax], [ymin, ymax]]
           norm (double, optional): normalization factor for plot
           ylog (logical, optional): True-> semilog plot, False-
        > linear plot
```

hist2d(self, u_vals, v_vals, nplt, labels=None, nbins=100, limits=None)

```
Create an 2d histogram of user given u & v values

Arguments:
   u_vals (Numpy vector):values to plot u-axis
   v_vals (Numpy vector):values to plot v-axis
   nplt (int): which plot in figure grid, by row,
               1 is upper left, nrow*ncol is lower right
   labels ([str, str]): u- and v-axes lables
   nbins (int, optional): number of x and y bins
   limits (list, doubles, optional): [[xmin, xmax], [ymin, ymax]]
```

hist2d_phase_space(self, coor, beam, mask, nplt, nbins=100, limits=None)

```
Create an 2d histogram phase-space plot

Arguments:
   coor (str): Phase space to plot, either 'xxp', 'yyp', or 'phiw'
   beam (beam object): beam object containing coordinates to plot
   mask (Numpy vector, int): mask for filtering beam prior to plotti
ng
   nplt (int): which plot in figure grid, by row,
               1 is upper left, nrow*ncol is lower right
   nbins (int, optional): number of x and y bins
   limits (list, doubles, optional): [[xmin, xmax], [ymin, ymax]]
```

iso_phase_space(self, coor, beam, mask, nplt, nbins=50)

```
Create an isometric phase-space plot.

Arguments:
   coor (str): Phase space to plot, either 'xxp', 'yyp', or 'phiw'
   beam (beam object): beam object containing coordinates to plot
   mask (Numpy vector): mask for filtering beam prior to plotting
   nplt (int): which plot in figure grid, by row,
               1 is upper left, nrow*ncol is lower right
   nbins (int, optional): number of x and y bins, respectively
```

phase_space(self, coor, beam, mask, nplt, marker='b,', limits=None)

```
Create beam phase space dot plot as nth subplot to figure

Arguments:
   coor (str): text string either 'xxp', 'yyp' or 'phiw'
   beam (object): object containing beam to be plotted
   mask (Numpy array): mask for filter beam prior to plotting
   nplt (int): which plot in figure grid, by row,
                1 is upper left, nrow*ncol is lower right
   marker (str, optional): matplotlib color and marker, e.g. 'r.'
   limits (list of doubles, optional): plot limits [[xmin, xmax], [y
min, ymax]]
```

profile(self, coor, beam, mask, nplt, marker='g-', nbins=50, limits=None, ylog=False)

```
Create a profile of beam coordinate

Arguments:
   coor (str): coordinate to plot, either 'x', 'xp', 'y', 'yp', 'phi
', 'w' or 'losses'
   beam (beam object): beam object containing coordinates to plot
   mask (numpy vector): mask for filter beam prior to plotting
   nplt (int): which plot in figure grid, by row,
               1 is upper left, nrow*ncol is lower right
   marker (str, optional): matplotlib color and marker, e.g. 'r.'
   nbins (int, optional): number of bins to plot
   limits (list, doubles, optional): [[xmin, xmax], [ymin, ymax]]
   ylog (logical, optional): True-> semilog plot, False-
> linear plot
```

show(self)

```
Show the plots. Used in non-interactive mode
```

surf_phase_space(self, coor, beam, mask, nplt, nbins=100, limits=None)

```
Create a surface phase-space plot

Arguments:
   coor (str): Phase space to plot, either 'xxp', 'yyp', or 'phiw'
   beam (beam object): beam object containing coordinates to plot
   mask (Numpy vector): mask for filtering beam prior to plotting
   nplt (int): which plot in figure grid, by row,
               1 is upper left, nrow*ncol is lower right
   nbins (int, optional): number of x and y bins
   limits (list, doubles, optional): [[xmin, xmax], [ymin, ymax]]
```

title(self, title)

```
Place title string in window bar
Arguments:
   title (str): figure title
```

class **DBConnection**

```
An hpsim class for creating the database connection.
The user must provide the following arguments to constructor:

databases: an ordered python list containing the individual database
           filenames to be used in the simulations. The database must be
           ordered to represent the linac from upstream to downstream
```

Methods defined here:

__init__(self, db_dir, databases, libsql_dir, libsql_file)

```
Init loads and attaches databases so those original functions are no
t
separately available

Arguments:
   db_dir (str): path of dir containing db files
   databases (list of str): ordered list of database filenames in co
rrect sequence
   libsql_dir (str): path of directory that contains external sql li
b
   libsql_file (str): name of libsqliteext.so file

Returns:
   dbconnection object
```

clear_model_index(self)

```
Clears model index. Must be called once db connection established
```

get_epics_channels(self)

```
Returns a list of all the EPICS PV's in the db's connected thru
dbconnection
```

print_dbs(self)

```
Prints names of datatbases
```

print_libs(self)

```
Prints the database library
```

Data and other attributes defined here:

dbconnection = ''

class **DBState**

```
An hpsim class capturing the state of all EPICS PV's in the connected dB's
```

Methods defined here:

__init__(self)

Create dB state object

get_db_pvs(self, file=None)

```
Record all Epics PV names and values from db
If filename present then also write to that file

Arguments:
   file (str, optional): filename to write output to
```

print_pvs(self, pvname=None)

```
Print vals of EPICS PVs in DBState object that correspond to pvnam
e
Print all PVs vals in state object if pvname is not supplied

Arguments:
   pvname (str, optional): print value of named Epics PV
```

restore_db_pvs(self, file=None)

```
Restore EPICS PV in file or DBState object back into dB
If file present use file else use DBState object

Arguments:
   file (str, optional): filename from which to extract dB Epics PV
 values
```

turn_off(self, pv_name)

```
Set all PV's with name pv_name to val of zero

Arguments:
   pv_name(str): name of Epics PV
```

turn_on(self, pv_name)

```
Restore all PV's with name to associated vals from DBState

Arguments:
   pv_name(str): name of Epics PV
```

class **DistPlot**

```
An hpsim class for creating plots of beam distribution objects
```

Methods defined here:

__init__(self, nrow=1, ncol=1, hsize=None, vsize=None)

```
Creates and instance of a matplotlib figure
```

clear(self)

```
Clear plot figure
```

draw(self)

```
Draw figure. Used in interactive mode
```

hist1d(self, u_vals, nplt, nbins=50, xlabel=None, limits=None, norm=1.0, ylog=False)

```
Create 1d histogram of arbitrary vals in numpy array

Arguments:
   u_vals (Numpy vector):values to plot
   nplt (int):which plot in figure grid, by row,
               1 is upper left, nrow*ncol is lower right
   nbins (int, optional): number of bins to plot
   xlabel (str, optional): x-axis label
   limits (optional, [list of doubles]) [[xmin, xmax], [ymin, ymax]]
   norm (double, optional): normalization factor for plot
   ylog (logical, optional): True-> semilog plot, False(default)-
> linear plot
```

hist1d_coor(self, coor, nplt, nbins=50, xlabel=None, limits=None, norm=1.0, ylog=False)

```
Create a histogram style profile of beam coordinate
```

```
Arguments:
   coor (str): coordinate to plot, either 'x', 'xp', 'y', 'yp', 'phi
', 'w' or 'losses'
   dist is beam-distribution object
   nplt (int): which plot in figure grid, by row,
               1 is upper left, nrow*ncol is lower right
   nbins (int, optional): number of bins to plot
   xlabel (str, optional): x-axis label
   limits (list, doubles, optional): [[xmin, xmax], [ymin, ymax]]
   norm (double, optional): normalization factor for plot
   ylog (logical, optional): True-> semilog plot, False-
> linear plot
```

hist2d(self, u_vals, v_vals, nplt, labels=None, nbins=100, limits=None)

```
Create an 2d histogram of user given u & v values

Arguments:
   u_vals (Numpy vector):values to plot u-axis
   v_vals (Numpy vector):values to plot v-axis
   nplt (int): which plot in figure grid, by row,
               1 is upper left, nrow*ncol is lower right
   labels ([str, str]): u- and v-axes lables
   nbins (int, optional): number of x and y bins
   limits (list, doubles, optional): [[xmin, xmax], [ymin, ymax]]
```

hist2d_phase_space(self, coor, dist, nplt, nbins=100, limits=None)

```
Create an 2d histogram phase-space plot

Arguments:
   coor (str): Phase space to plot, either 'xxp', 'yyp', or 'phiw'
   dist is beam-distribution object
   nplt (int): which plot in figure grid, by row,
               1 is upper left, nrow*ncol is lower right
   nbins (int, optional): number of x and y bins
   limits (list, doubles, optional): [[xmin, xmax], [ymin, ymax]]
```

iso_phase_space(self, coor, dist, nplt, nbins=50)

```
Create an isometric phase-space plot.

Arguments:
   coor (str): Phase space to plot, either 'xxp', 'yyp', or 'phiw'
   dist is beam-distribution object
   nplt (int): which plot in figure grid, by row,
               1 is upper left, nrow*ncol is lower right
   nbins (int, optional): number of x and y bins, respectively
```

phase_space(self, coor, dist, nplt, marker='b,', limits=None)

```
Add beam phase space as nth subplot to figure

Arguments:
   coor (str): text string either 'xxp', 'yyp' or 'phiw'
   dist is beam-distribution object
   nplt (int): which plot in figure grid, by row,
                1 is upper left, nrow*ncol is lower right
   marker (str, optional): matplotlib color and marker, e.g. 'r.'
   limits (list of doubles, optional): plot limits [[xmin, xmax], [y
min, ymax]]
```

profile(self, coor, dist, nplt, marker='g-', nbins=50, limits=None, ylog=False)

```
Add profile of beam coordinate
Arguments:
   coor (str): coordinate to plot, either 'x', 'xp', 'y', 'yp', 'phi
', 'w' or 'losses'
   dist is beam-distribution object
   nplt (int): which plot in figure grid, by row,
               1 is upper left, nrow*ncol is lower right
```

```
      marker (str, optional): matplotlib color and marker, e.g. 'r.'
      nbins (int, optional): number of bins to plot
      limits (list, doubles, optional): [[xmin, xmax], [ymin, ymax]]
      ylog (logical, optional): True-> semilog plot, False-
   > linear plot
```

show(self)

```
   Show the plots
```

surf_phase_space(self, coor, dist, nplt, nbins=100, limits=None)

```
   Create a surface phase-space plot

   Arguments:
      coor (str): Phase space to plot, either 'xxp', 'yyp', or 'phiw'
      dist is beam-distribution object
      nplt (int): which plot in figure grid, by row,
                  1 is upper left, nrow*ncol is lower right
      nbins (int, optional): number of x and y bins
      limits (list, doubles, optional): [[xmin, xmax], [ymin, ymax]]
```

title(self, title)

```
   Place title string in window bar
```

class **Distribution**

```
An hpsim class for holding a masked beam of particles as a static np-
array.
Faster for analysis and plotting than using beam array
```

Methods defined here:

__init__(self, beam, mask=None)

```
   Init creates an instance of the Distribution object containing al
   l
   the vectors of coordinates from the beam object that satisfy the mas
   k

   Attributes:
      mass (double): mc^2 of particle double mass (MeV)
      charge (double): q/|e|
      current (double): bunch peak current(Amps)
      frequency (double): MHz
      size (int): number of macro particles
      betagamma (double): beta * gamma of masked beam
      betalambda (dounble): beta * lambda of masked beam
      ref_phi (double): reference particle phase (Rad)
      ref_w (double): reference particle energy (MeV)

   Returns:
       Returns a beam distribution object
```

get_avg(self, var)

get_betagamma(self)

```
   Return betagamma of beam
```

get_betalambda(self)

```
   Return betalambda
```

get_charge(self)

```
   Return charge of beam in q/|e|
```

get_coor(self, var)

get_current(self)

```
   Return beam frequency in MHz
```

get_frequency(self)

```
   Return beam frequency in MHz
```

get_loss_num(self)

```
Returns number of macro-particles lost transversely
```
get_mass(self)
```
Return mass of beam, mc^2, in MeV
```
get_nrms_emit(self, var)
```
Return normalized rms emittance along specified axis, x, y or z
```
get_ref_phi(self)
```
Return the phase of reference particle
```
get_ref_w(self)
```
Return the phase of reference particle
```
get_sig(self, var)

get_size(self)
```
Returns total number of beam macro particles
```
get_twiss(self, var)
```
Return Twiss parameters (a,b,unnormalized, rms e) for specified coor
ds x, y or z
```
get_urms_emit(self, var)
```
Return unnormalized rms emittance along specified axis, x, y or z
```
print_results(self)
```
Prints avg, sigma, alpha, beta, Eurms, Enrms for all coord of distr
```

Data and other attributes defined here:

coor_index = {'losses': 6, 'phi': 4, 'w': 5, 'x': 0, 'xp': 1, 'y': 2, 'yp': 3}

class **Simulator**

```
An hpsim class for defining the simulator
```

Methods defined here:

__init__(self, beam)
```
Creates an instance of the simulator class

Arguments:
   beam (object): beam class object

Returns:
   Simulator class object
```
set_space_charge(self, state='off')
```
Turn space charge on or off
state (str, optional): "on", "off"(default)
```
simulate(self, start_elem_name, end_elem_name)
```
Simulate from 'start' element to 'end' element, inclusive
```

class **SpaceCharge**

```
An hpsim class for defining and modifying the space charge used in
the simulation
```

Methods defined here:

__init__(self, nr, nz, interval, adj_bunch, type="'scheff'")
```
Creates an instance of the space-charge class

Arguments:
   nr (int): number of space-charge mesh slices in r direction
   nz (int): number of space-charge mesh slices in z direction
   interval (double): maximum spacing between space charge kicks
   adj_bunch (int): number of adj_bunch used in s.c. calc
   type (str, optional): "scheff" by default and is the only option
```

```
                                    at the moment
```
get_adj_bunch(self)
```
        Return the number of adjacent bunches in space charge calculation
```
get_adj_bunch_cutoff_w(self)
```
        Return the cutoff energy (MeV) above which the adjacent bunches are
        no longer used in space charge calculation and s.c. mesh region base
        d
        upon beam size, i.e. 3*sigmas. This enables automatic transition to
        faster s.c. calc once adjacent bunches need no longer be considered
```
get_interval(self)
```
        Returns the interval, i.e. maximum drift distance (m) between
        space-charge kicks
```
get_mesh_size(self)
```
        Return a list of floats representing the r,z mesh size
```
get_mesh_size_cutoff_w(self)
```
        Return the cutoff energy for the beam at which the mesh size will
        decrease by nr/2 and nz/2 and interval increase by 4.This enables
        automatic transition to faster s.c. calc.
```
get_remesh_threshold(self)
```
        Get the remeshing factor (default is 0.05) where
        0 => remesh before every space-charge kick
        >0 => adaptive algorithm determines how much beam shape can change
        before mesh must be redone
```
set_adj_bunch(self, adj_bunch)
```
        Set the number of adjacent bunches used in space charge calculation
        Argument:
           adj_bunch (int): number of adjacent bunches to use in s.c. calc.
```
set_adj_bunch_cutoff_w(self, w_cutoff)
```
        Set the cutoff energy (MeV) above which the adjacent bunchss are
        no longer used in space charge calculation and s.c. mesh region base
        d
        upon beam size, i.e. 3*sigmas. This enables automatic transition to
        faster s.c. calc once adjacent bunches need no longer be considered

        Argument:
           w_cutoff (double): threshold energy above which adjacent bunches
        are no
                              longer used in s.c. calc
```
set_interval(self, interval)
```
        Set maximum drift distance (m) between space-charge kick

        Argument:
           interval (double): maximum distance between space-charge kicks
```
set_mesh_size(self, nr, nz)
```
        Set the size of the mesh, i.e. nr, nz

        Arguments:
           nr (double): number of radial grid points
           nz (double): number of longitudinal grid points
```
set_mesh_size_cutoff_w(self, w_cutoff)
```
        Set the cutoff energy for decreasing the mesh by nr/2, nz/2 and
        increasing interval by 4. This enables automatic transition to
        faster s.c. calc.

        Arguments:
           w_cutoff (double): Threshold energy (MeV) where s.c. calc reduces

                              nr by factor 2, nz by factor 2 and interval by
         factor 4.
```
set_remesh_threshold(self, rm_thres)

```
Set the remeshing factor (default is 0.05) where
0 => remesh at before every space-charge kick
>0 => adaptive algorithm determines how much beam shape can change
before mesh must be redone

Arguments:
   rm_thres (double): the factor that determines if the s.c. grid is
 remeshed
                     or not.
```

Data and other attributes defined here:
spacecharge = ''

Functions

betalambda(mass, freq, w)

```
Return value of beta*lambda of beam.

Arguments:
   mass(double): mc^2 of beam particle in MeV
   freq(double): frequency in MHz
   w(double): Kinetic energy in MeV
```

get_beamline_direction(start_elem, stop_elem)

```
Returns +1 for stop_elem beyond start_elem or -
1 if stop_elem behind start_elem

Arguments:
   start_elem(str): beginning element name
   stop_elem(str): final element name
```

get_beamline_length(start, end)

```
Returns length of beamline from element 'start' to element 'end'
in hpsim base units (m). If start is after stop, then the length is < 0

Arguments:
   start(str): first element in list
   end(str): last element in list
```

get_beamline_midpoints()

```
Returns a list of the distance to the midpoint of each element in
the complete beamline, units (m).

Arguments: None
```

get_db_epics(pv_name)

```
Retrieve EPICS PV value in database

Arguments:
   pv_name (str): EPICS pv name string
   Note: DBConnection must be already be established
```

get_db_model(elem_name, field_name)

```
Retrieve model database parameter value given by table_name and field_na
me

Arguments:
   table_name (str): name of element in db table
   field_name (str): name of field to change of element in table
   Note: DBConnection and BeamLine must be already be established
```

get_element_length(elem_name)

```
Return length of beamline element in hpsim base units(m).

Arguments:
   elem_name(str): name of element
```

get_element_list(start_elem_name=None, end_elem_name=None, elem_type=None)

```
Retrieve a list containing the names of beamline elements from
'start_elem_name' to 'end_elem_name'

Arguments:
   start_elem_name(str): first element in list
   end_elem_name(str): last element in list
   elem_type(str): type of element (db type or C++ type) to retrieve
```

get_first_element()

```
Returns name of first element in connected database

Arguments: None
```

get_last_element()

```
Returns name of first element in connected database

Arguments: None
```

get_mmf(twiss1, twiss2)

```
Returns the MisMatch Factor between to sets of Twiss parameters
where Twiss is (alpha, beta, eps)

Arguments:
   twiss1(list of doubles): [alpha, beta, emittance]
   twiss2(list of doubles): [alpha, beta, emittance]
```

get_next_element(elem_name)

```
Returns the name of the next element in the connected databases
Arguments:
   elem_name(str): name of element
```

modulo_phase(phase_dist, ref_phs)

```
Return the phase coordinates of beam after modulo 360 deg wrt ref_phs
   has been applied

Arguments:

   phase_dist (Numpy vector, doubles): phase coordinates (deg)
   ref_phs (double): reference phase for modulo calc
```

most_freq_value(vals)

```
Returns an estimate of the most frequently occuring value
by first histograming the the Numpy array npvals in a histogram
with unit bins, then finding the peak, then averaging that along
with the adjacent bins to get an estimate of the value that
represents the most frequency value.

Arguments:
   vals(Numpy array): input 1D array

Returns:
   estimate of the value that occurs most frequently
```

set_db_epics(pv_name, value)

```
Change EPICS PV value in database

Arguments:

   pv_name(str): EPICS pv name string
   value(str or double): value to set Epics PV to
   Note: DBConnection and BeamLine must be already be established
```

set_db_model(table_name, field_name, value)

```
Change model database parameter value given by table_name and field_name

Arguments:
   table_name (str): name of element in db table
   field_name (str): name of field to change of element in table
```

```
        value (str or double): value to set db field to
        Note: DBConnection and BeamLine must be already be established
```

set_gpu(n)

```
    Set which GPU to use
    Arguments:
        n(int): number of GPU to use
```

Data

colorConverter = <matplotlib.colors.ColorConverter object>
lib_dir = '/Users/larry_r/Projects/HPSimulator Development/python scripts/bin'
par_dir = '/Users/larry_r/Projects/HPSimulator Development/python scripts'
pkg_dir = '/Users/larry_r/Projects/HPSimulator Development/python scripts/pylib'

Lcsutil.py

lcsutil

[.](#)
[lcsutil.py](#)

```
# lcsutil_v2.py
# Collection of python functions for accessing LANSCE Control System (LCS)   info
```

Functions

expand_pv(pv)

```
Returns the LCS Process Variable in full syntax AADDDNNNCMM as a string
where AA is two character area,
      DD is device type,
      NNN is device number,
      C  is channel type,
      MM is channel number
Returns partial PVs when input string is incomplete.

Argument:
   pv(str): LCS PV to expane
```

get_neg_beams()

```
Returns list of names associated with H- beam
['TB', 'H-', 'LB', '-']
```

get_pos_beams()

```
Returns list of names associated with H+ beam:
['TA', 'H+', 'LA', 'IP', '+']
```

get_pv_area(pv)

```
Returns two character area for PV

Arguments:
   pv(str): name of LCS PV
```

get_pv_asp(n, beam='+')

```
Returns EPICS/LCS PV name (string) for the amplitude set point of
module n or buncher n,beam '+' or '-', n can be a string
e.g. 'TA' or '05', or a number 5.

Arguments:
   n(str or int): module or buncher number
   beam(str, optional): beam species
```

get_pv_asp_n(n)

```
Returns ASP PV associated with neg(-) beam species, either -.
Required for Main Buncher.

Arguments:
   n(str or int): module or buncher number
```

get_pv_asp_p(n)

```
Returns ASP PV associated with pos(+) beam species, either +.
Required for Main Buncher.

Arguments:
   n(str or int): module or buncher number
```

get_pv_device(pv)

```
Returns two character device for PV

Arguments:
   pv(str): name of LCS PV
```

get_pv_psp(n, beam='+')

```
Returns EPICS/LCS PV name (string) for the phase set point of
module n or buncher n can be a string e.g. 'TA' or '05', or a number 5.

Arguments:
   n(str or int): module or buncher number
   beam(str, optional): beam species
```

get_pv_rf(n, beam='+')

```
Returns Epics/LCS PV name (string) for the on(intime)/off(delayed)
set point of module n or buncher n, where n can be a string
e.g. 'TA' or '05', or a number 5.

Arguments:
   n(str or int): module or buncher number
   beam(str, optional): beam species
```

get_pv_type(pv)

```
Returns one character type for PV

Arguments:
   pv(str): name of LCS PV
```

rf_off_val(n)

```
Returns RF 'OFF' value for bunchers, DTL or CCL rf,
n can be a string e.g. 'TA' or '05', or a number 5.

Arguments:
   n(str or int): module or buncher number
```

rf_on_val(n)

```
Returns RF 'ON' value for bunchers, DTL or CCL rf,
n can be a string e.g. 'TA' or '05', or a number 5.

Arguments:
   n(str or int): module or buncher number
```

Nputil.py

nputil_v3

index
nputil_v3.py

```
#nputil_v3.py
#python scripts for manipulating Numpy arrays
```

Modules

numpy sys

Functions

apply_limits(xy, limits)

```
Returns new Numpy array of same shape
where limits[0] <= x value <= limits[1]
for either 1D or xy-pair Numpy array

Arguments:
   xy(numpy array): input array
   limits(list double): [xmin, xmax]
```

apply_threshold(xy, thres)

```
Returns new Numpy array of same shape where values below threshold are
replaced with zeros 1D array and xy-pair Numpy array where y-
value >= thres.

Arguments:
   xy(numpy array): input array
   thres(int or double): threshold value
```

apply_threshold_old(xy, thres)

```
Returns new Numpy array of same shape
where values are above threshold in 1D array
xy-pair Numpy array where y-value >= thres

Arguments:
   xy(numpy array): input array
   thres(int or double): threshold value
```

get_fwhm(xy)

```
Returns FWHM (deg) of xy-pair Numpy array.

Arguments:
   xy(numpy array): input array

Returns:
   phase diff between upper and lower half-max points
```

get_halfmax_x(xy, half)

```
Returns the interpolated x coord value corresponding to the halfmax_val
   in either the lower or upper half of the xy-pair Numpy array.

Arguments:
   xy(numpy array): input array
   half (str): 'left' or 'lower' or 'right' or 'upper'

Returns:
```

```
         Interpolated x coord corresponding to half_max val
```

get_max_index(xy)

```
      Returns the Numpy array index of xy-pair with maximum y.

      Arguments:
         xy(numpy array): input array
```

get_max_pair(xy)

```
      Returns an xy-pair from a Numpy array where y has the max value.

      Arguments:
         xy(numpy array): input array
```

get_max_val(xy)

```
      Returns the maximum y value in the xy-pair array.
      Arguments:
         xy(numpy array): input array
```

get_min_index(xy)

```
      Returns the Numpy array index of xy-pair with maximum y.
      Arguments:
         xy(numpy array): input array
```

get_min_pair(xy)

```
      Returns an xy-pair from a Numpy array where y is minimum.
      Arguments:
         xy(numpy array): input array
```

get_min_val(xy)

```
      Returns the minimum y value in the xy-pair array.
      Arguments:
         xy(numpy array): input array
```

get_pvr(xy)

```
      Returns the y Peak to valley ratio for xy-pair Numpy array
      The minimum must be > 0.
      Arguments:
         xy(numpy array): input array
```

get_x(xy)

```
      Returns the x-vector from xy-pair Numpy array.
      Arguments:
         xy(numpy array): input array
```

get_x_y(xy)

```
      Returns two Numpy vectors taken from the xy-pair Numpy array.
      Arguments:
         xy(numpy array): input array
```

get_xy(x, y)

```
      Returns the ordered xy-pairs in Numpy array.
      Arguments:
         xy(numpy array): input array
```

get_y(xy)

```
      Returns the y-vector from xy-pair Numpy array.

      Arguments:
         xy(numpy array): input array
```

remove_zero_pairs(xy)

```
      Returns new xy-pair Numpy array where x=y=0 pairs have been removed

      Arguments:
         xy(numpy array): input array
```

remove_zeros(xy)

```
      Returns new Numpy array of same shape
      where zeros have been removed from 1D array
      or x=y=0 pairs have been removed from xy-pair Numpy array

      Arguments:
```

```
      xy(numpy array): input array
```

runsum(array, dir=0)

```
Returns a Numpy array containing the running sum in direction dir
0 (default) along a row and 1 along a column

Arguments:
   array(numpy 2d array): input array
```

swap_columns(array, i, j)

```
Returns a copy of the Numpy array with columns i and j being swapped
Arguments:
   array(numpy 2d array): input array
   i(int): column number to swap
   j(int): column number to swap
```

Sqldb.py

sqldb [sqldb.py]

```
Python access to the linac databases for quicker read access
to various quantities, e.g. element length
```

Modules

sqlite3 os sys

Classes

Db_bl

class **Db_bl**

```
An class for accessing the given beamline databases through SQLite
```

Methods defined here:

__init__(self, db_dir, db_list)

```
Creates database beamline object that contains dir and names
of beamline db files
db_dir,    // path to db directory
db_list    // list containing names of databases
```

get_bl_elem_len(self)

```
Returns a sorted list of lists containing the
view_index, element_name, element_length, cumulative_length
```

Example HPSim Scripts (Offline mode)

The main method of interacting with HPSim in an offline mode is via Python scripts. The offline mode is useful for many purposes. Here are a few of them: 1) trying to simulate beam measurements, e.g. emittance, profiles, phase scans, etc., 2) extracting model calibration constants from a comparison/fit of model to measurement, 3) optimizing the machine tune, 4) investigating new modes or operation or beam performance under specific conditions. The following are just a few example Python scripts along with explanations to show how the Python packages are used to configure and run HPSim to perform various simulations.

In the offline mode, HPSim enables a user to simulate a beam from point A to point B in beam line under specific conditions that don't require the LCS EPICS connection. To perform this kind of simulation requires the following: 1) input beam distribution, 2) beam line description in SQL db files, 3) a workstation (tested on Linux) with an NVIDIA GPU (tested with Fermi and Kepler architectures) and 4) Python and the packages and related ones mentioned above for scripting.

Boilerplate code

Most scripts will need to follow the same set of initial steps leading up to the actual simulation. They are the following: 1) Import requisite modules, 2) Establish DB connnections, 3) Create Beamline object, 4) Create input beam object, 5) Create space-charge object and 6) Create Simulator Object. Once these steps are complete, then the user is ready to perform a typical simulation. (N.B. You only need to establish DB connections to the database files that represent the problem space that you are simulating. However, the database files must be in the correct order, i.e. linac start to finish, as they are loaded into memory sequentially!) The boilerplate section is given here:

`Code is shown in Courier font`

Python comments begin with "#"

Description of the sections is given in blue italic

HEADER

This is a comment section

```
#!/usr/bin/env python
#
# sim-lbeg.py
# for simulating LBEG beam from point A to B in the LANSCE
linac
#
```

IMPORT MODULE SECTION

Here we define paths for importing our packages and libraries, then we import them.

```
import sys
import os
# define directory to packages and append to $PATH
par_dir = os.path.abspath(os.path.pardir)
print par_dir
lib_dir = os.path.join(par_dir,"bin")
print lib_dir
sys.path.append(lib_dir)
pkg_dir = os.path.join(par_dir,"pylib")
print pkg_dir
sys.path.append(pkg_dir)

#import additional python packages
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import math
# import the HPSim simulation package
import hpsim as hps
# use next line to select GPU device
GPU = 0
hps.set_gpu(GPU)
# import additional simulation packages
import lcsutil as lcs
import nputil as npu
import sqldb as pydb
```

CREATE DATABASE CONNECTION SECTION

Here we create a list of the database filies that represent TB, TD, DTL, TRST and CCL and then connect them to HPSim

N.B. the database files must be placed in the correct order!

```
##########################################################
# install db's and connect to beamline
db_dir = par_dir + '/db'
lib_dir = par_dir + '/db/lib'
dbs = ['tbtd.db','dtl.db','trst.db','ccl.db']
dbconn1 = hps.DBConnection(db_dir, dbs, lib_dir,
'libsqliteext.so')
dbconn1.print_dbs()
dbconn1.clear_model_index()
print "*** dB connection established ***"
```

CREATE BEAMLINE SECTION
Here we create the beam line object

```
############################################################
# create beamline
bl = hps.BeamLine()
beamline = hps.get_element_list()
print "*** Beamline created ***"
```

CREATE INPUT BEAM OBJECT SECTION
*Here we create the input beam distribution. It is a DC waterbag located at the H- prebuncher and is based upon the actual beam measurements at a nearby emittance station. To make the beam always arrive at the same phase, it is offset by -360° * number of period in βλ, wrt BLZ, the start of the DTL. The reference particle is also shifted by this amount. Finally, the input beam is saved to allow it to be restored for later use in repetitive simulation requiring the identical input distribution.*

```
############################################################
# create H- beam
##beam = hps.Beam(file='zzz.txt')
SIM_START = "TBDB02"
beam = hps.Beam(mass=939.294, charge=-1.0, current=0.015,
num=1024*16) #H- beam
beam.set_dc(0.095, 47.0, 0.00327,  -0.102, 60.0, 0.002514,
180.0, 0.0, 0.7518) #TBDB02 20140901
beam.set_frequency(201.25)
betalambda = hps.betalambda(mass = beam.get_mass(),
freq=beam.get_frequency(), w=0.750)
phi_offset = -hps.get_beamline_length('TBDB02','BLZ') /
betalambda * 360
beam.set_ref_w(0.750)
beam.set_ref_phi(phi_offset)
beam.translate('phi', phi_offset)
beam.save_initial_beam()
print "*** H- Beam created ***"
```

CREATE SPACECHARGE OBJECT SECTION
Here we create the space charge object, which includes the s.c. grid, the maximum interval that s.c. is applied in a drift, and the number of adjacent bunches. We also set the cutoff energy to stop using adjacent bunches, early in the dtl, and also the scale factor the determines how when the bunch size has changed enough to warrant remeshing the beam.

```
############################################################
# create spacecharge
```

```
spch = hps.SpaceCharge(nr = 32, nz = 128, interval = 0.025,
adj_bunch = 3)
print "spch interval=", spch.get_interval()
print "adj_bunch=", spch.get_adj_bunch()
# define at what energy simulation stops using adjacent
bunches in SC calc
spch.set_adj_bunch_cutoff_w(0.8)
# remeshing factor determines how ofter the mesh gets
recalc vs scaled for SC kick
spch.set_remesh_threshold(0.02)
print "cutoff w=", spch.get_adj_bunch_cutoff_w()
print "*** Space Charge Initialized ***"
```

CREATE SIMULATOR OBJECT SECTION
Here we create the simulator object and turn on space charge for the simulation.

```
##########################################################
# create simulator
sim = hps.Simulator(beam)
sim.set_space_charge('on')
print "*** Simulator Initialized ***"
```

Example 1: Simulate LBEG H- beam from prebuncher to 48DT

This HPSim offline example script "sim-lbeg.py" (Appendix A) is used to simulate a realistic H- LBEG beam from the LEBT Transport B prebuncher to the CCL module 48 delta-T loop located after the last linac module. The boilerplate code is required but not shown below. The remainder of the script is given below. Shown in Figures 19 and 20 are the printed output beam info and plot results, respectively, produced by HPSim with this script.

```
# define the start and stop locations and a energy_cutoff
for later analysis
SIM_STOP = '48DT' #delta-t loop after last accel module
ENERGY_CUTOFF = 790.0 #MeV

# print out input beam info
print "*** Input Beam ***"
print "w/user units"
beam.print_results()

# perform simulation
print "*** Starting Simulation ***\n"
sim.simulate(SIM_START, SIM_STOP)
```

```
# define mask for later analysis and plotting that will
only include ‘good’ i.e. not lost particles whose energy is
greater than the cutoff defined above
wmask = beam.get_mask_with_limits('w', lolim =
ENERGY_CUTOFF)
gmask = beam.get_good_mask(wmask)
mask = gmask

# print info on output beam with masked applied
print "*** Output Beam ***"
print "w/user units"
beam.print_results(mask)

# create various plots of output beam with masked applied
plot = hps.BeamPlot(nrow=4, ncol=3, hsize=16, vsize=12)
plot.title(SIM_STOP)
plot.iso_phase_space('xxp', beam, mask, 1)
plot.iso_phase_space('yyp', beam, mask, 2)
plot.iso_phase_space('phiw', beam, mask, 3)
plot.hist2d_phase_space('xxp', beam, mask, 4)
plot.hist2d_phase_space('yyp', beam, mask, 5)
plot.hist2d_phase_space('phiw', beam, mask, 6)
plot.profile('x', beam, mask, 7, 'g-')
plot.profile('y', beam, mask, 8, 'g-')
plot.profile('phi', beam, mask, 9, 'g-')
plot.profile('xp', beam, mask, 10, 'g-')
plot.profile('yp', beam, mask, 11, 'g-')
plot.profile('w', beam, mask, 12, 'g-')
plot.show()
exit()
```

To execute a HPSim Python script in the offline mode, one should enter the word python followed by the script name at the command prompt in a terminal window:

python sim-lbeg.py

This is done in the pytest directory that contains the pythons scripts and in this case will envoke the python interpreter on the example script sim-lbeg.py .

The masking operations used in the script examples enables to user to select a subset of the beam distribution for analysis or plotting that satisfies some set of criteria. For example, creating a mask with a lower energy cutoff and combining it with the “good particles” is a way to select only those macro particles that were not lost on a transverse aperture, i.e. “good” and had energies greater than the lower energy cutoff. A mask equal to “None” can be used to analyze/plot all particles.

```
*** Starting Simulation ***
Change beam frequency from 201.25 to 805, ratio = 4, current change to 0.06
*** Output Beam ***
w/user units
Distribution Analysis Results (w/user units)
Mass = 939.2940
Charge/|e| = -1
Ib = 45.77 mA
Number of macroparticles = 799883
Frequency = 805.000 MHz
Ref part.
 phi = 1284811.1142 deg
   w =   799.7095 MeV

Centroids and RMS sizes
            Avg          Sigma
x   :     0.3674        0.3857 cm
xp  :     0.1595        0.3040 mr
y   :    -0.0001        0.2124 cm
yp  :    -0.0000        0.1682 mr
phi: 1284811.7820         6.1954 deg
w   :   801.0263        1.0127 MeV

Twiss parameters
         Alpha       Beta       Eurms       Enrms
x :     -0.5469     1.4458     0.1029     0.16046
y :     -0.8553     1.6615     0.0271     0.04235
z :     -0.0128     6.1184     6.2735     6.27347
```

Figure 19: Masked LBEG output beam information at 48DT .

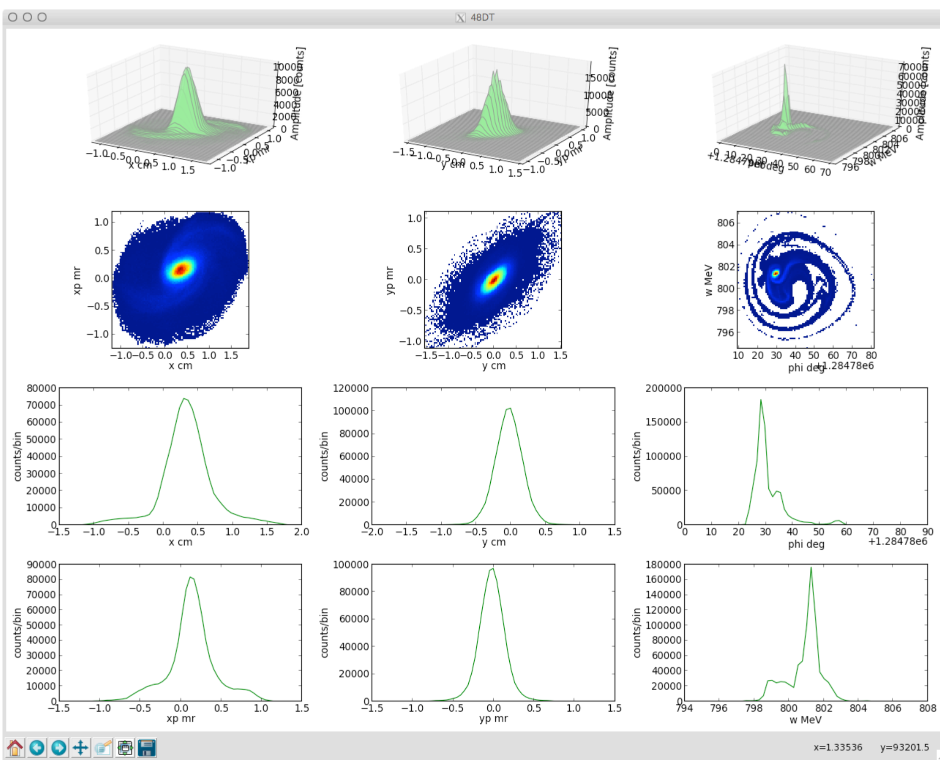

Figure 20: Plot results from sim-lbeg.py HPSim script showing beam phase space and profiles at 800 MeV.

Example 2: Scan a LINAC Control Parameter, i.e. EPICS channel

This HPSim offline example script “sim-lbeg-scan-pv.py” (Appendix B) is used to scan a linac control parameter, in this case DTL module 3 amplitude set point, while simulating a realistic H- LBEG beam from the Transport B prebuncher to the CCL module 48 delta-T loop located after that module. The boilerplate code required for the script was shown above. The remainder of the script is given below. Shown in Figures 20 and 21 are beam plots at an intermediate step and the beam parameters versus Module 3 amplitude set point, respectively, produced by HPSim with this script.

```
##########################################################
SIM_STOP = '48DT'
ENERGY_CUTOFF = 790.0 #MeV; should be less than the nominal
beam output energy
PV_TYPE = 'AMP' #'AMP' or 'PHS' or 'OTHER' or 'NONE'
AREA = 'TD'
BEAM = '-'

if PV_TYPE == 'OTHER':
    # if OTHER then define EPICS_PV here and adjust scan
range below
    EPICS_PV = 'MRPH001D01' #master reference phase for
whole CCL

if PV_TYPE == None:
    EPICS_PV = None
```

Here we define the range and step size of the scan to be performed.

```
if PV_TYPE == 'PHS':
# EPICS pv to scan
    EPICS_PV = lcs.get_pv_psp(AREA, beam=BEAM)
    # Initial pv PHS value; will restore after scan
    PV_INIT_VAL = hps.get_db_epics(EPICS_PV)
    D_PV = 15.0 #degree
    PV_MIN = PV_INIT_VAL - D_PV
    PV_MAX = PV_INIT_VAL + D_PV
    PV_STEP = 2.0 #degree

elif PV_TYPE == 'AMP':
    EPICS_PV = lcs.get_pv_asp(AREA, beam=BEAM)
    # Initial pv AMP value; will restore after scan
    PV_INIT_VAL = hps.get_db_epics(EPICS_PV)
    D_PV = 5.0 #percent of initial value
    PV_MIN = PV_INIT_VAL * (1.0 - D_PV/100.)
    PV_MAX = PV_INIT_VAL * (1.0 + D_PV/100.)
```

```
    STEP_SIZE = 2.5 #percent of initial value
    PV_STEP = STEP_SIZE/100.0 * PV_INIT_VAL
    print np.arange(PV_MIN, PV_MAX, PV_STEP)

elif PV_TYPE == 'OTHER':
    PV_INIT_VAL = hps.get_db_epics(EPICS_PV)
    D_PV = 120.
    PV_MIN = PV_INIT_VAL - D_PV
    PV_MAX = PV_INIT_VAL + D_PV
    PV_STEP = 10.0

elif PV_TYPE == None:
    EPICS_PV = None
    print PV_TYPE

else:
    print 'Error with pv_type'
    exit()

print '{0} starting value is {1}'.format(EPICS_PV,
PV_INIT_VAL)
```

Here we 'try' to scan. If something bad happens and the script aborts it should exit gracefully.

```
try: #use 'try' to allow graceful exit if simulation
crashes, so that PV is restored to init val
    plt.ion() #interactive mode ON
    plot = hps.DistPlot(nrow=4, ncol=3, hsize=16, vsize=12)
    output = []
    bll = []

    for val in np.arange(PV_MIN, PV_MAX, PV_STEP):
        # restore initial input beam distribution before
          starting next step
        beam.restore_initial_beam()
        # set Module 3 ASP to next value
        hps.set_db_epics(EPICS_PV, val)
        # simulate here
        sim.simulate(SIM_START,SIM_STOP)

        wmask =
beam.get_mask_with_limits('w',lolim=ENERGY_CUTOFF)
        mask = beam.get_good_mask(wmask)

        if len(mask) > 0:
            try:
                dist = hps.Distribution(beam, mask)
```

```
            # beam.print_results(mask)
            # save npart, avgW, sigW, avgPHI, sigPHI beam
quantities to output list
                output.append([val, dist.get_size(),
                               dist.get_avg('w'),
                               dist.get_sig('w'),
                               dist.get_avg('phi'),
                               dist.get_sig('phi')])

                plot.clear()
     # create intermediate results to plot for each scan
step
                plot.iso_phase_space('xxp', dist, 1)
                plot.iso_phase_space('yyp', dist, 2)
                plot.iso_phase_space('phiw', dist, 1)
                plot.hist2d_phase_space('xxp', dist, 4)
                plot.hist2d_phase_space('yyp', dist, 5)
                plot.hist2d_phase_space('phiw', dist, 2)
                plot.profile('x', dist, 7, 'r-')
                plot.profile('y', dist, 8, 'r-')
                plot.profile('phi', dist, 4, 'r-')
                plot.profile('xp', dist, 10, 'r-')
                plot.profile('yp', dist, 11, 'r-')
                plot.profile('w', dist, 3, 'r-')
                title = "H{0} from {1} to {2}; {3} {4} =
{5}".format(\
                    BEAM, SIM_START, SIM_STOP, PV_TYPE,
EPICS_PV, val)
                plot.title(title)
     # plot intermediate results
                plot.draw()
            except:
                print " Warning - No output for PV at this
value"
finally:
    plt.ioff()
    # restore scan parameter to original value
    hps.set_db_epics(EPICS_PV, PV_INIT_VAL)
    print '{0} restore to original value
{1}'.format(EPICS_PV, PV_INIT_VAL)

    # print scan result quantities in table
    for item in output:
        print item


# create plots of output list quantities versus channel
scanned
```

```
    pv_val, npart, wout, sig_w, avg_phi, sig_phi =
zip(*output)

    fig2 = plt.figure()
    title = "H{0} from {1} to {2}; scan of {3} {4}".format(\
            BEAM, SIM_START, SIM_STOP, PV_TYPE, EPICS_PV)
    fig2.canvas.set_window_title(title)

    a1 = fig2.add_subplot(511)
    a1.plot(pv_val, npart, 'b-')
    a1.set_ylabel('Num part')
    npm = max(npart)
    a1.set_ylim([0.95*npm, 1.05*npm])

    a2 = fig2.add_subplot(512)
    a2.plot(pv_val, wout, 'b-')
    a2.set_ylabel('W out (Mev)')

    a3 = fig2.add_subplot(513)
    a3.plot(pv_val, sig_w, 'b-')
    a3.set_ylabel('Sigma W (MeV)')

    a4 = fig2.add_subplot(514)
    a4.plot(pv_val, avg_phi, 'b-')
    a4.set_ylabel('Avg Phi (deg)')

    a5 = fig2.add_subplot(515)
    a5.plot(pv_val, sig_phi, 'b-')
    a5.set_ylabel('Sigma phi (deg)')
    a5.set_xlabel(EPICS_PV)

    # plot results
    plt.show()
exit()
```

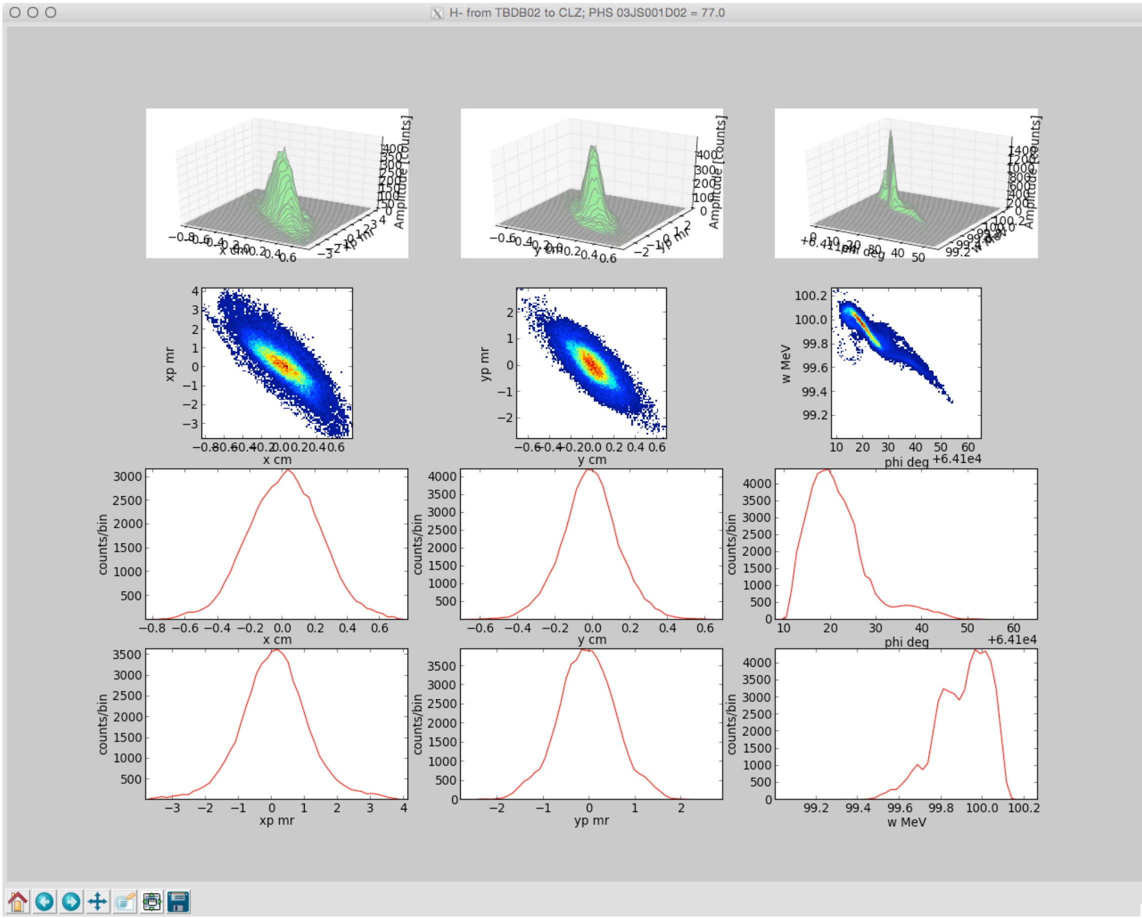

Figure 21: Masked LBEG output beam information at entrance to CCL at intermediate point in scan of Module 3 phase set point.

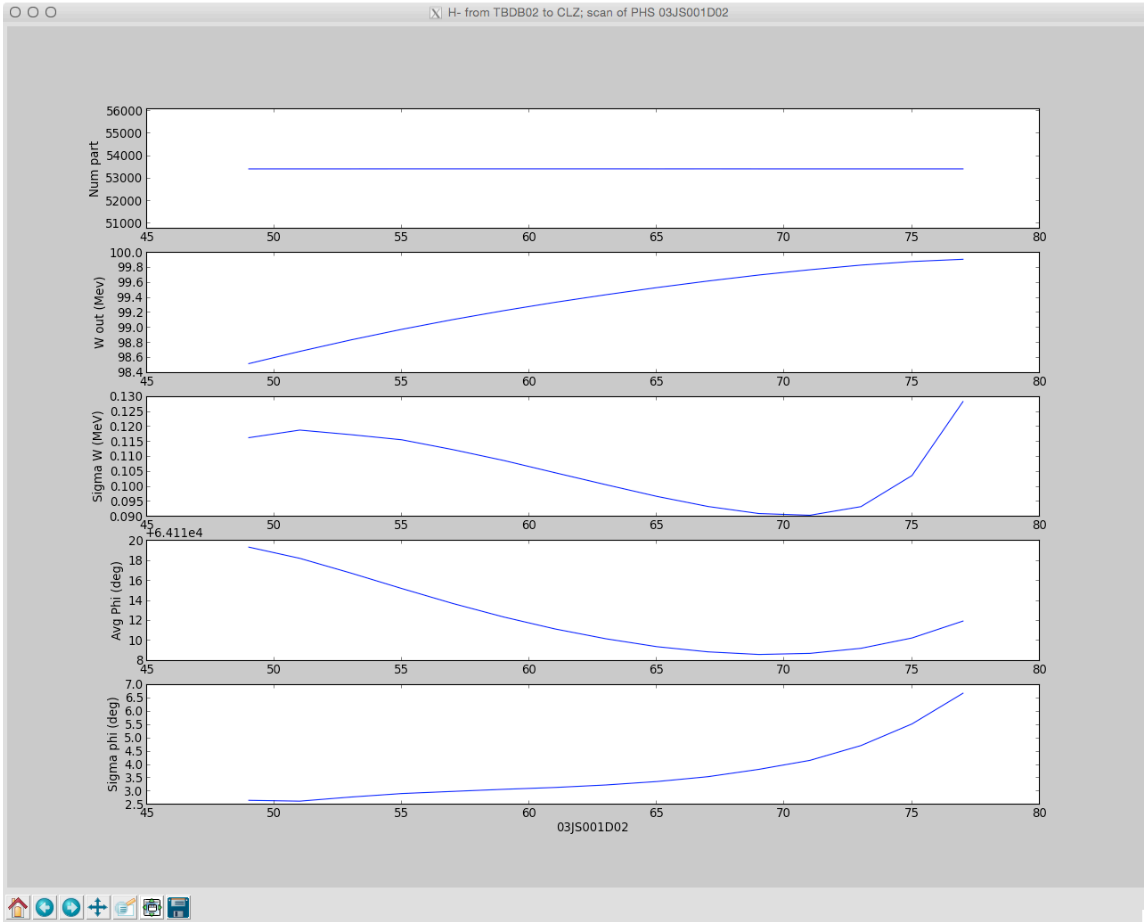

Figure 22: Scan results showing variation in LBEG beam at entrance to CCL versus Module 3 Phase set point, 03JS001D02.

Performing the same scan but now taking the beam to the end of the CCL and apply an energy cutoff of 790 MeV produces the results shown in Figures 23 and 24.

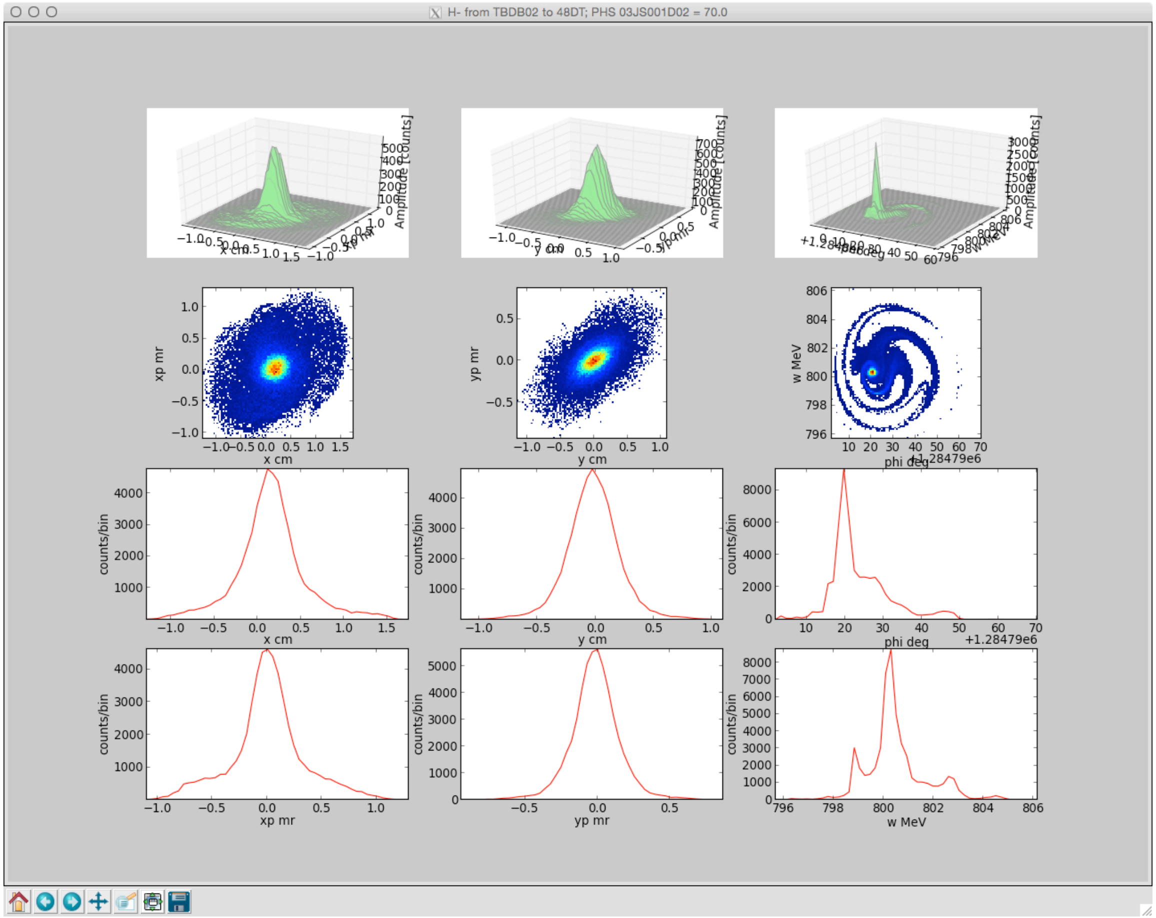

Figure 23: Masked LBEG output beam information at the exit of CCL at intermediate point in scan of Module 3 phase set point.

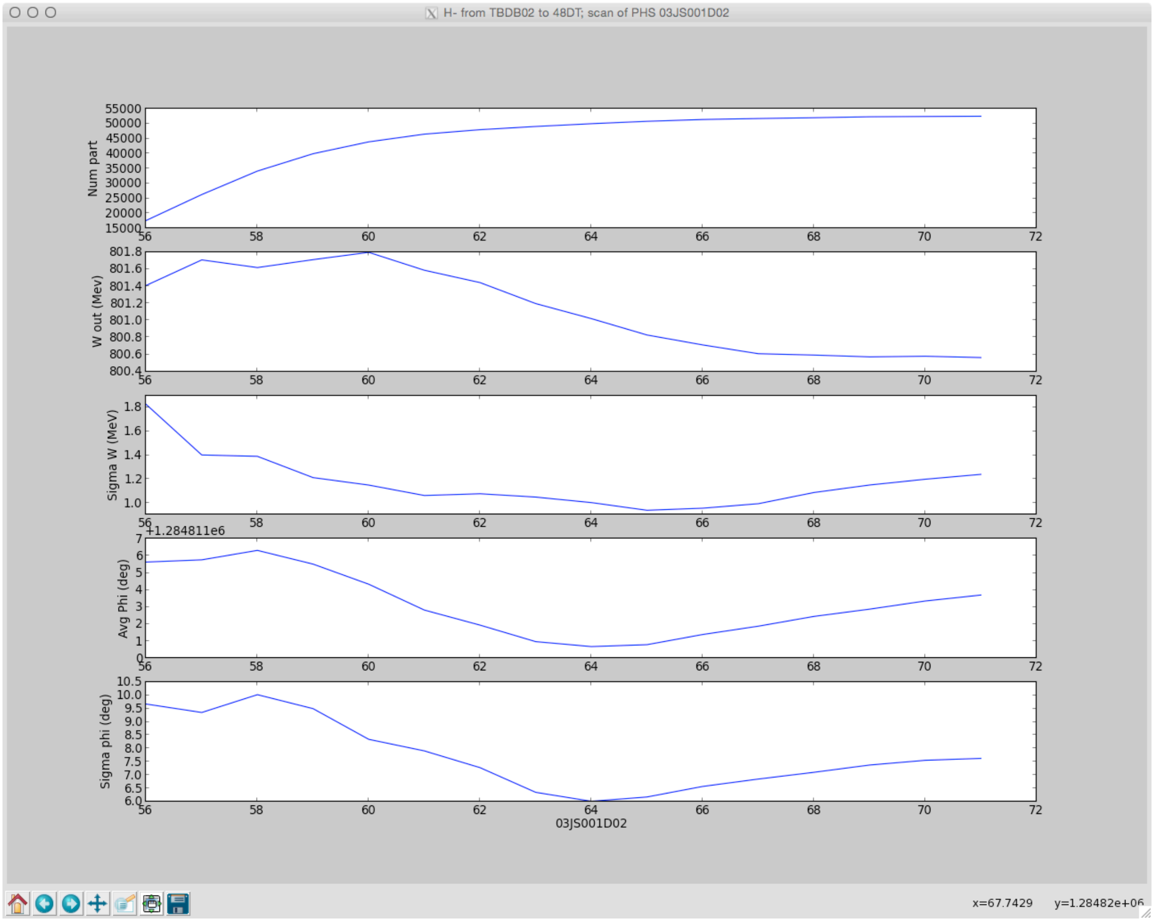

Figure 24: Scan results showing variation in LBEG beam at exit of the CCL versus Module 3 Phase set point, 03JS001D02.

Comments about Model Calibration

Any HPSim model database file will likely contain calibrations curves and factors needed to transform EPICS PV setpoint values used to operate a beamline device to physics model quantities needed to calculate the effect of this device on the beam. For beamline magnets, e.g. quadrupole, an excitation curve, Integral Gdl vs. I, can be created from magnet mapping data and stored in the database. This allows the model to transform an EPICS PV associated with the excitation current in the magnet field strength. RF cavities are a bit different.

Each RF accelerating/buncher structure nominally requires two calibration functions/constants, which are associated with the cavity field amplitude and phase and their relationship to their respective EPICS PVs. We have successfully used a combination of a Superfish EM model and measured cavity Q and RF power vs. EPICS amplitude set point to create a calibration function relating buncher gap voltage to the EPICS amplitude set point PV [10]. The phase-offset calibration constant requires a beam-based measurement that is fitted using HPSim to extract the value. We have successfully user an absorber-collector beam-based measurement to extract these values. For large, multicell accelerating structures, e.g. a drift-tube linac (DTL) or coupled-cavity linac (CCL) a phase scan method using either an absorber-collector or ΔT technique has been successfully used [1, 10] to extract calibration constants. Python scripts were written to simulate the actual phase-scan measurements and extract the calibration constants. Whatever technique is employed, the scripts should be written to simulate the measurement so that the calibration constants can be extract from an HPSim fit to the data.

HPSim (Online mode)

The online version of HPSim is run from the command line using a file that contains the required definitions. There are two different versions for online use: 2D and 3D. Unlike the offline scripts that can be run across the network, the online version must be run from the workstation console.

2D Version

The 2D version produces plots of the output beam phase-space distributions, output profiles and centroid, rms size, emittances and losses along the linac. Shown in Figure 25 is the sample input file for the 2D online model. The contents of this file include the list of databases to be used, the file containing the input beam

```
[larry_r@aothpsim hpsim_old]$ more start2d.txt
db: ./db/online-tbtd.db ./db/online-201.db
beam: ./python-scripts/TBDB02_input_beam_64k.dat
start: TBDB02
end: 04QM30
server: off
```

Figure 25: Sample 'start2d.txt' file used by the 2D online version of HPSim.

distribution, the start and end points in the simulation and the state of the EPICS data server. When the model is run it will create a continuously updating set of plots as shown in Figure 26.

The command to start the 2d version is "./start2d start2d.txt".

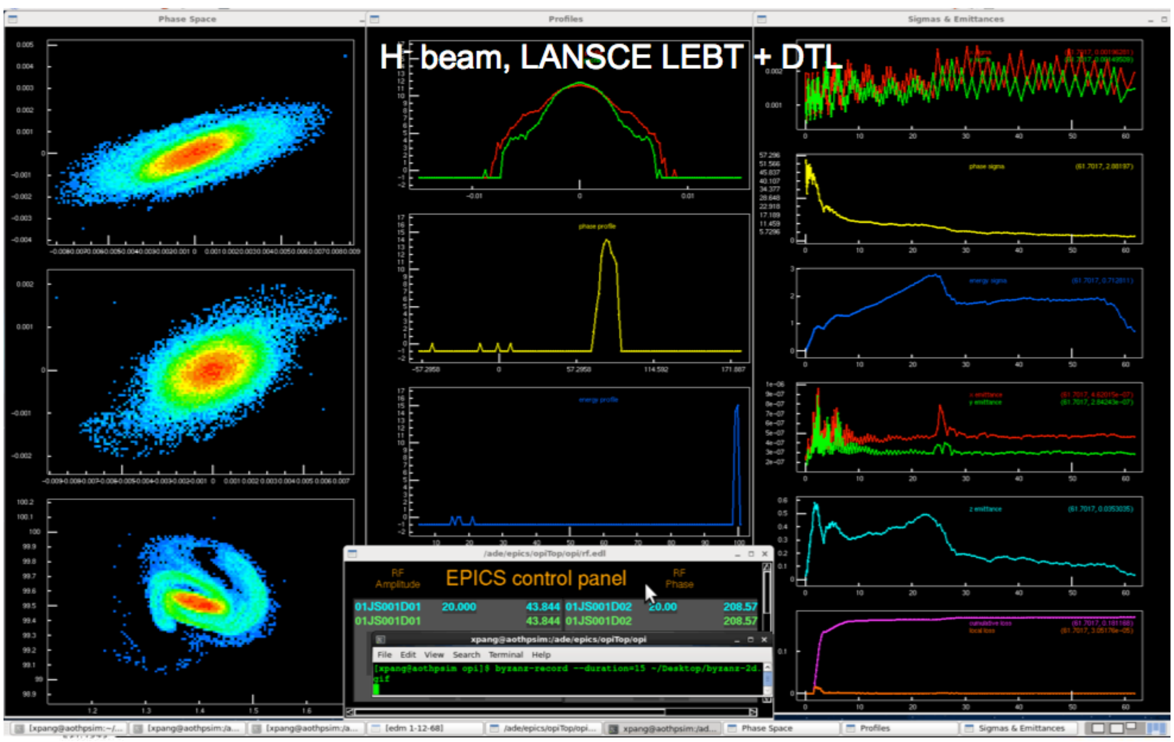

Figure 26: Sample graphical output generated by 2D online version of HPSim.

3D Version

The 3D version uses a similar input file for db, beam, and end point definition as to the 2D version. However, the 3D version creates the beam phase-space plots and a 3D-like view of the beam distribution in real space as it progresses along the linac. This is shown in Figure 27.

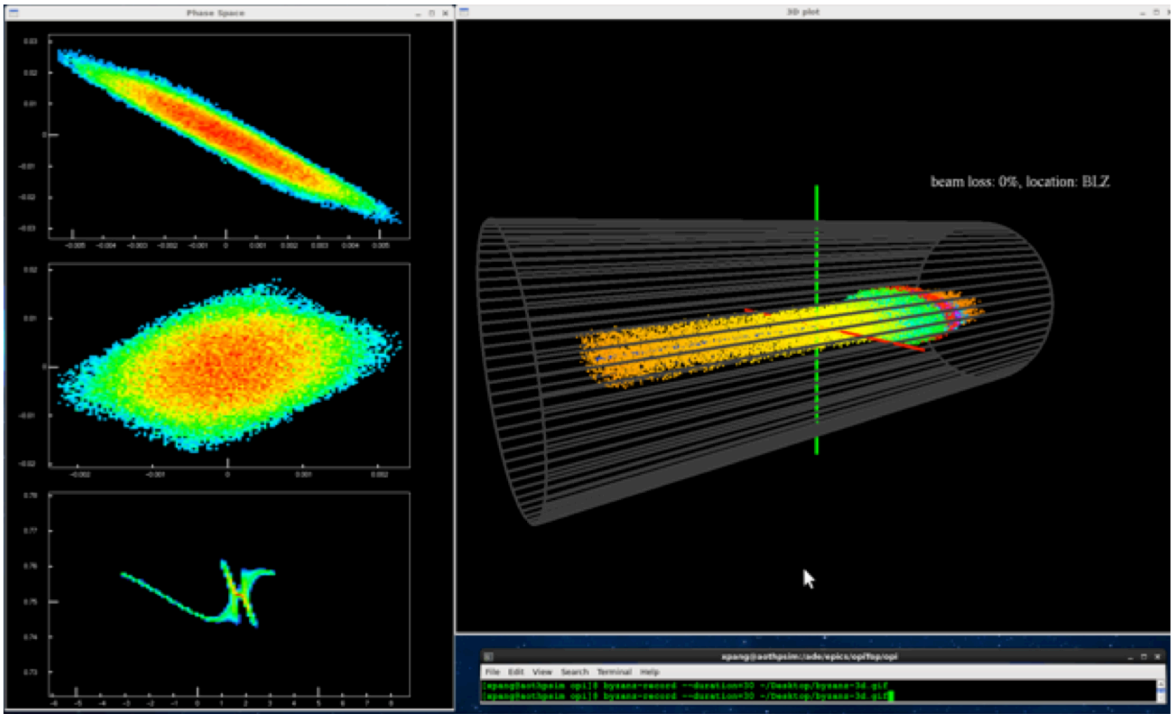

Figure 27: Sample graphical output generated by 3D online version of HPSim.

References

[1] X. Pang, "Advances in Proton Online Modeling," Proceedings of IPAC2015, Richmond, May 3-8, 2015, pp. 2423-2427.

[2] H. Takeda and J. Billen, "PARMILA Manual," Los Alamos Report, LA-UR-98-4478, Revised July 2005.

[3] EPICS: Experimental Physics and Industrial Control System, http://www.aps.anl.gov/epics/license/index.php

[4] SQLite, https://www.sqlite.org/about.html

[5] NVIDIA, NVIDIA CUDA C Programming Guide, Version 4.0, 2011

[6] Python, https://www.python.org/

[7] J.H. Billen and L.M Young, "Poisson Superfish," LA-UR-96-1834, revised 13 January 2006.

[8] X. Pang and L. Rybarcyk, "Multi-Objective Particle Swarm and Genetic Algorithm for Optimization of the LANSCE Linac Operation," Nucl. Instr. Meth. A 741, pp. 124-129, 2014.

[9] A. Scheinker et al, "Model Independent Particle Accelerator Tuning," Phys. Rev. ST Accel. Beams, 16, 102803, 2013.

[10] L. J. Rybarcyk and X. Pang, "Application and Calibration Aspects of a New High-Performance Beam-Dynamics Simulator for the LANSCE Linac," proceedings of the PAC2013 conference, Pasedena, CA, Sept. 29-Oct 4, 2013, pp. 676-678.

Appendixes

A. sim-lbeg.py

The following is a listing of the sim-lbeg.py script that is used to simulate the LANSCE LBEG H- beam from the 750 keV LEBT to anywhere up to the end of the CCL.

```
#!/usr/bin/env python
#
# sim-lbeg.py
# for simulating LBEG beam from point A to B in the LANSCE linac
#
import sys
import os
# define directory to packages and append to $PATH
par_dir = os.path.abspath(os.path.pardir)
print par_dir
lib_dir = os.path.join(par_dir,"bin")
print lib_dir
sys.path.append(lib_dir)
pkg_dir = os.path.join(par_dir,"pylib")
print pkg_dir
sys.path.append(pkg_dir)

#import additional python packages
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import math
# import additional simulation packages
import hpsim as hps
import HPSim as HPSim
# use next line to select either GPU 0 or 2 on aothpsim
GPU = 0
hps.set_gpu(GPU)
import lcsutil as lcs
import nputil as npu
import sqldb as pydb

################################################################################
#
# install db's and connect to beamline
db_dir = par_dir + '/db'
lib_dir = par_dir + '/db/lib'
dbs = ['tbtd.db','dtl.db','trst.db','ccl.db']
dbconn1 = hps.DBConnection(db_dir, dbs, lib_dir, 'libsqliteext.so')
dbconn1.print_dbs()
dbconn1.clear_model_index()
print "*** dB connection established ***"

################################################################################
#
# create beamline
bl = hps.BeamLine()
beamline = hps.get_element_list()
print "*** Beamline created ***"

################################################################################
#
# create table of beamline elements at lengths
```

```
pybl = pydb.Db_bl(db_dir, dbs)
py_beamline = pybl.get_bl_elem_len()
print "*** PySQLite Beamline created ***"

################################################################################
#
# create H- beam
SIM_START = "TBDB02" #defined by input beam location
beam = hps.Beam(mass=939.294, charge=-1.0, current=0.015, num=1024*256) #H-
beam
beam.set_dc(0.095, 47.0, 0.00327,  -0.102, 60.0, 0.002514, 180.0, 0.0, 0.7518)
#TBDB02 20140901
beam.set_frequency(201.25)
betalambda = hps.betalambda(mass = beam.get_mass(), freq=beam.get_frequency(),
w=0.750)
phi_offset = -hps.get_beamline_length(SIM_START,'BLZ')/betalambda *360
beam.set_ref_w(0.750)
beam.set_ref_phi(phi_offset)
beam.translate('phi', phi_offset)
beam.save_initial_beam()
print "*** H- Beam created ***"

################################################################################
#
# create spacecharge
spch = hps.SpaceCharge(nr = 32, nz = 128, interval = 0.025, adj_bunch = 3)
print "spch interval=", spch.get_interval()
print "adj_bunch=", spch.get_adj_bunch()
# define at what energy simulation stops using adjacent bunches in SC calc
spch.set_adj_bunch_cutoff_w(0.8)
# remeshing factor determines how ofter the mesh gets recalc vs scaled for SC
kick
spch.set_remesh_threshold(0.02)
print "cutoff w=", spch.get_adj_bunch_cutoff_w()
print "*** Space Charge Initialized ***"

################################################################################
#
# create simulator
sim = hps.Simulator(beam)
sim.set_space_charge('on')
print "*** Simulator Initialized ***"

################################################################################
#
# STANDARD AND REQUIRED STUFF ABOVE THIS LINE
################################################################################
#

SIM_STOP = 'BLZ'
ENERGY_CUTOFF = 0.0
mask = gmask = beam.get_good_mask()

print "*** Input Beam ***"
print SIM_START
print "w/user units"
beam.print_results()

print "*** Starting Simulation ***\n"
sim.simulate(SIM_START, SIM_STOP)

# determine mask of particles used in analysis and plotting
wmask = beam.get_mask_with_limits('w', lolim = ENERGY_CUTOFF)
gmask = beam.get_good_mask(wmask)
mask = gmask
```

```
print "*** Output Beam ***"
print SIM_STOP
print "w/user units"
beam.print_results(mask)

# create output plot
plot = hps.BeamPlot(nrow=4, ncol=3, hsize=16, vsize=12)
plot.title(SIM_STOP)
plot.iso_phase_space('xxp', beam, mask, 1)
plot.iso_phase_space('yyp', beam, mask, 2)
plot.iso_phase_space('phiw', beam, mask, 3 )
plot.hist2d_phase_space('xxp', beam, mask, 4)
plot.hist2d_phase_space('yyp', beam, mask, 5)
plot.hist2d_phase_space('phiw', beam, mask, 6)
plot.profile('x', beam, mask, 7, 'g-')
plot.profile('y', beam, mask, 8, 'g-')
plot.profile('phi', beam, mask, 9, 'g-')
plot.profile('xp', beam, mask, 10, 'g-')
plot.profile('yp', beam, mask, 11, 'g-')
plot.profile('w', beam, mask, 12, 'g-')
plot.show()
exit()
```

B. sim-lbeg-scan-pv.py

The following is a listing of the sim-lbeg-scan-pv.py script that is used to simulate the LANSCE LBEG H- beam from the 750 keV LEBT to somewhere in the linac up to the end of the CCL while varying an EPICS PV used to control a linac machine parameter.

```
#!/usr/bin/env python
# sim-lbeg-scan-pv.py
# simulate lbeg H- beam through the LANSCE beamline while varying EPICS PV over
range of vals

import sys
import os
# define directory to packages and append to $PATH
par_dir = os.path.abspath(os.path.pardir)
print par_dir
lib_dir = os.path.join(par_dir,"bin")
print lib_dir
sys.path.append(lib_dir)
pkg_dir = os.path.join(par_dir,"pylib")
print pkg_dir
sys.path.append(pkg_dir)

#import additional python packages
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import math
# import additional simulation packages
import hpsim as hps
import HPSim as HPSim
# use next line to select either GPU 0 or 1
hps.set_gpu(2)
import lcsutil as lcs
import nputil as npu

################################################################################
#
# install db's and connect to beamline
db_dir = par_dir + '/db'
lib_dir = par_dir + '/db/lib'
dbs = ['tbtd.db','dtl.db','trst.db','ccl.db']
dbconn1 = hps.DBConnection(db_dir, dbs, lib_dir, 'libsqliteext.so')
dbconn1.print_dbs()
dbconn1.clear_model_index()
print "*** dB connection established ***"

################################################################################
#
# create beamline
bl = hps.BeamLine()
beamline = hps.get_element_list()
print "*** Beamline created ***"

################################################################################
#
# create H- beam
SIM_START = "TBDB02" #defined by input beam location
beam = hps.Beam(mass=939.294, charge=-1.0, current=0.015, num=1024*64) #H- beam
```

```
beam.set_dc(0.095, 47.0, 0.00327,  -0.102, 60.0, 0.002514, 180.0, 0.0, 0.7518)
#TBDB02 20140901
beam.set_frequency(201.25)
betalambda = hps.betalambda(mass = beam.get_mass(), freq=beam.get_frequency(),
w=0.750)
phi_offset = -hps.get_beamline_length('TBDB02','BLZ')/betalambda *360
beam.set_ref_w(0.750)
beam.set_ref_phi(phi_offset)
beam.translate('phi', phi_offset)
beam.save_initial_beam()
print "*** H- Beam created ***"

################################################################################
#
# create spacecharge
spch = hps.SpaceCharge(nr = 32, nz = 128, interval = 0.025, adj_bunch = 3)
print "spch interval=", spch.get_interval()
print "adj_bunch=", spch.get_adj_bunch()
# define at what energy simulation stops using adjacent bunches in SC calc
spch.set_adj_bunch_cutoff_w(0.8)
# remeshing factor determines how ofter the mesh gets recalc vs scaled for SC
kick
spch.set_remesh_threshold(0.02)
print "cutoff w=", spch.get_adj_bunch_cutoff_w()
print "*** Space Charge Initialized ***"

################################################################################
#
# create simulator
sim = hps.Simulator(beam)
sim.set_space_charge('on')
print "*** Simulator Initialized ***"

################################################################################
#
# STANDARD AND REQUIRED STUFF ABOVE THIS LINE
################################################################################
#

SIM_STOP = 'TREM01'
ENERGY_CUTOFF = 95.0 #MeV; should be less than the nominal beam output energy
PV_TYPE = 'AMP' #'AMP' or 'PHS' or 'OTHER' or 'NONE'
AREA = 'TD'
BEAM = '-'

if PV_TYPE == 'OTHER':
    # if OTHER then define EPICS_PV here and adjust scan range below
    EPICS_PV = 'MRPH001D01' #master reference phase for whole CCL

if PV_TYPE == None:
    EPICS_PV = None

if PV_TYPE == 'PHS':
# EPICS pv to scan
    EPICS_PV = lcs.get_pv_psp(AREA, beam=BEAM)
    # Initial pv PHS value; will restore after scan
    PV_INIT_VAL = hps.get_db_epics(EPICS_PV)
    D_PV = 20.0 #degrees
    PV_MIN =PV_INIT_VAL - D_PV
    PV_MAX = PV_INIT_VAL + D_PV
    PV_STEP = 5.0 #degree

elif PV_TYPE == 'AMP':
    EPICS_PV = lcs.get_pv_asp(AREA, beam=BEAM)
    # Initial pv AMP value; will restore after scan
```

```
    PV_INIT_VAL = hps.get_db_epics(EPICS_PV)
    D_PV = 5.0 #percent of initial value
    PV_MIN = PV_INIT_VAL * (1.0 - D_PV/100.)
    PV_MAX = PV_INIT_VAL * (1.0 + D_PV/100.)
    STEP_SIZE = 2.5 #percent of initial value
    PV_STEP = STEP_SIZE/100.0 * PV_INIT_VAL
    print np.arange(PV_MIN, PV_MAX, PV_STEP)

elif PV_TYPE == 'OTHER':
    PV_INIT_VAL = hps.get_db_epics(EPICS_PV)
    D_PV = 120.
    PV_MIN = PV_INIT_VAL - D_PV
    PV_MAX = PV_INIT_VAL + D_PV
    PV_STEP = 10.0

elif PV_TYPE == None:
    EPICS_PV = None
    print PV_TYPE

else:
    print 'Error with pv_type'
    exit()

print '{0} starting value is {1}'.format(EPICS_PV, PV_INIT_VAL)

try: #use try to allow graceful exit if simulation crashes, so that PV is
restored to init val
    plt.ion() #interactive mode ON
    plot = hps.DistPlot(nrow=4, ncol=3, hsize=16, vsize=12)
    output = []
    bll = []

    for val in np.arange(PV_MIN, PV_MAX, PV_STEP):
        beam.restore_initial_beam()
        hps.set_db_epics(EPICS_PV, val)
        print EPICS_PV, "is now", hps.get_db_epics(EPICS_PV)
    # simulate here
        sim.simulate(SIM_START, SIM_STOP)

        wmask = beam.get_mask_with_limits('w',lolim=ENERGY_CUTOFF)
        mask = beam.get_good_mask(wmask)

        if len(mask) > 0:
            try:
                dist = hps.Distribution(beam, mask)
            # beam.print_results(mask)
            # save npart, avgW, sigW, avgPHI, sigPHI beam quantities to output
list
                output.append([val, dist.get_size(),
                               dist.get_avg('w'), \
                               dist.get_sig('w'), \
                               dist.get_avg('phi'), \
                               dist.get_sig('phi')])

                plot.clear()
                # create intermediate results to plot for each scan step
                plot.iso_phase_space('xxp', dist, 1)
                plot.iso_phase_space('yyp', dist, 2)
                plot.iso_phase_space('phiw', dist, 3)
                plot.hist2d_phase_space('xxp', dist, 4)
                plot.hist2d_phase_space('yyp', dist, 5)
                plot.hist2d_phase_space('phiw', dist, 6)
                plot.profile('x', dist, 7, 'r-')
                plot.profile('y', dist, 8, 'r-')
                plot.profile('phi', dist, 9, 'r-')
```

```
                plot.profile('xp', dist, 10, 'r-')
                plot.profile('yp', dist, 11, 'r-')
                plot.profile('w', dist, 12, 'r-')
                title = "H{0} from {1} to {2}; {3} {4} = {5}".format(\
                    BEAM, SIM_START, SIM_STOP, PV_TYPE, EPICS_PV, val)
                plot.title(title)
                plot.draw()
            except:
                print " Warning - No output for PV at this value"
finally:
    plt.ioff()
    hps.set_db_epics(EPICS_PV, PV_INIT_VAL)
    print '{0} restore to original value {1}'.format(EPICS_PV, PV_INIT_VAL)

    # print results
    for item in output:
        print item

# plot output list quantities
    pv_val, npart, wout, sig_w, avg_phi, sig_phi = zip(*output)

    fig2 = plt.figure()
    title = "H{0} from {1} to {2}; scan of {3} {4}".format(\
        BEAM, SIM_START, SIM_STOP, PV_TYPE, EPICS_PV)
    fig2.canvas.set_window_title(title)

    a1 = fig2.add_subplot(511)
    a1.plot(pv_val, npart, 'b-')
    a1.set_ylabel('Num part')

    a2 = fig2.add_subplot(512)
    a2.plot(pv_val, wout, 'b-')
    a2.set_ylabel('W out (Mev)')

    a3 = fig2.add_subplot(513)
    a3.plot(pv_val, sig_w, 'b-')
    a3.set_ylabel('Sigma W (MeV)')

    a4 = fig2.add_subplot(514)
    a4.plot(pv_val, avg_phi, 'b-')
    a4.set_ylabel('Avg Phi (deg)')

    a5 = fig2.add_subplot(515)
    a5.plot(pv_val, sig_phi, 'b-')
    a5.set_ylabel('Sigma phi (deg)')
    a5.set_xlabel(EPICS_PV)

    plt.show()
exit()
```