CERN – EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

CLIC – Note – 1064

LINEAR ACCELERATOR SIMULATION FRAMEWORK WITH PLACET AND GUINEA-PIG.

Jochem Snuverink and Jurgen Pfingstner

European Organization for Nuclear Research, Geneva, Switzerland,
Royal Holloway, University of London & University of Oslo, Norway

Many good tracking tools are available for simulations for linear accelerators. However, several simple tasks need to be performed repeatedly, like lattice definitions, beam setup, output storage, etc. In addition, complex simulations can become unmanageable quite easily. A high level layer would therefore be beneficial. We propose LinSim, a linear accelerator framework with the codes PLACET and GUINEA-PIG. It provides a documented well-debugged high level layer of functionality. Users only need to provide the input settings and essential code and / or use some of the many implemented imperfections and algorithms. It can be especially useful for first-time users. Currently the following accelerators are implemented: ATF2, ILC, CLIC and FACET. This note is the comprehensive manual, discusses the framework design and shows its strength in some condensed examples.

Geneva, Switzerland
14th December 2015

LinSim

Linear Accelerator Simulation Framework with PLACET and GUINEA-PIG

Jochem Snuverink and Jürgen Pfingstner

December 14, 2015

Contents

1. Introduction and purpose

When simulating linear accelerators or transport lines, one encounters repeating tasks that are basically the same for each simulation as:

- Setting up the model of the beamline and the beam
- Specifying and saving simulation parameters
- Implementing a simulation structure to simulate typical scenarios
- Implementing of correction techniques
- Implementing imperfections as ground motion and component imperfections
- Maintain scripts to ease parallel computing on a server farm
- Evaluating the results
- Performing backups and keeping track of changes

All these tasks are usually repeated for each simulation project that is started. As a result, there are a large number of implementations of very similar code in each beam physics team, which reduces the productivity significantly.

Instead of this approach, a unified simulation framework for PLACET [1] and GUINEA-PIG [2], named LinSim, is described in this document. It automates the mentioned tasks and takes its (re)implementation burden of the developer. The work of the user is simplified to specify simulation settings and to add the specific code for the given problem. This makes LinSim easy to use and still flexible enough to perform basically every test case. Additionally, such an unified framework can include more tools than would usually be implemented, e.g. consistency checks for certain parameters, automated settings storing to be able to reproduce the results at a later stage, code management via SVN [3], and many others. This helps to increase the productivity of the user. Especially (but not only) for newcomers, LinSim is an enormous starting help, since they don't have to know all the details of the already provided simulation scripts. Other advantages are that correction techniques (e.g. orbit feedbacks, IP feedbacks, dispersion free steering, wakefield alignment, and many other) only have to implemented once, which increases productivity drastically and leads to well debugged code.

The structure of the framework is illustrated in Fig. 1. It consists of a set of scripts written in Tcl [4] and Octave [5] that interface different simulation codes: PLACET is used for the beam tracking, a ground motion simulator [6] (which has been ported to C++ and is now included in PLACET) generates realistic element misalignments, and GUINEA-PIG facilitates beam-beam simulations. All these interfaced codes are written in C++ [7]. Several types of imperfections are implemented and can be turned on and off via simulation parameters (see Apps. C.6, C.7, and C.8). Also, many algorithms for the correction of these imperfections, e.g. beam-based alignment and orbit feedbacks, have been put in place (see Apps. C.10, C.11, and C.12). Historically, LinSim was developed

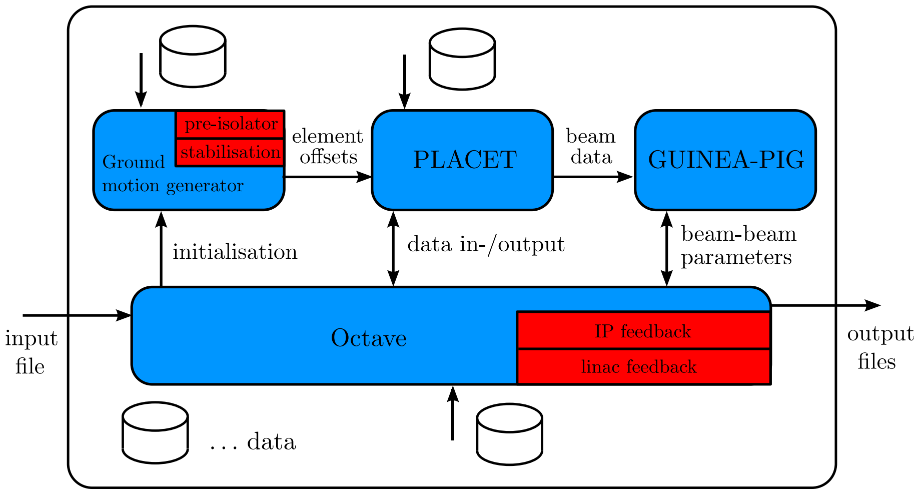

Figure 1: Internal structure of LinSim, where scripts (written in Tcl and Octave) interfaces the simulation codes PLACET, a ground motion generator, and GUINEA-PIG. An input file is used to control the simulations that use additional data as lattice files and created standardised output files.

for orbit feedback simulations for CLIC, but has then been extended to a universal tool. LinSim also provides the lattice, beam and wake field information of the accelerators ATF2, CLIC, FACET, and ILC (even though they are not directly an element of the code). All of these accelerators can be simulated by simply specifying their name as the parameter `machine_name` (see App. C.1).

2. Simulation structure

A typical simulation consists of two components as depicted in Fig. 2. The first part is a set of provided scripts that make up LinSim itself, together with provided lattice files and additional data as, e.g. orbit response matrices. The entry point to LinSim is the script `run.tcl`, which is executed with PLACET. The second element of a simulation is the user specific test, which consists of two files: a settings file and a code file (e.g `test_settings.tcl` and `test_code.m` in Fig. 2). These files control the execution of LinSim and therefore determine the specific simulation. In the settings file, parameters and settings can be specified that overwrite the initial ones. In the code file, code can be added that is linked into the execution of the framework at three different point to make LinSim as flexible as possible. Examples for test files are given in App. B.

The structure of each simulation can be divided in four parts depicted with different colours in Fig. 2. In the code, the status variable `sim_mode` indicates which part is currently executed, which is useful, e.g. for the logging of results:

- Initial setup (yellow): `sim_mode = setup_mode`
- Seed loop (orange): `sim_mode = static_mode`

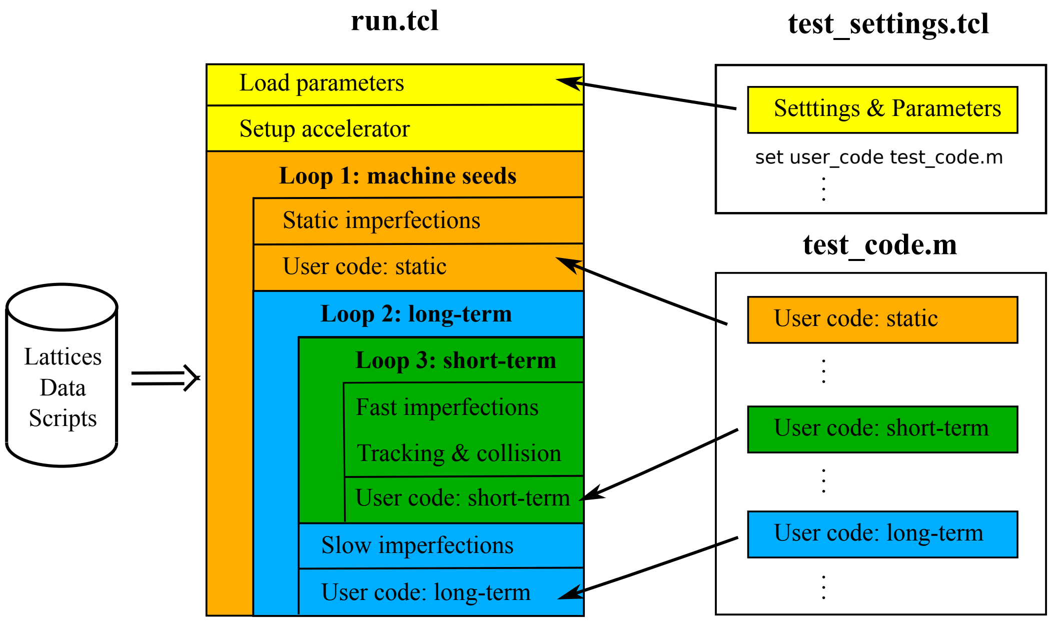

Figure 2: Structure of a simulation with LinSim (entry point is `run.tcl`), which used external data and is controlled via two test file, e.g. `test_settings.tcl` and `test_code.m`. Code from the test files is linked into the four different parts of the execution, which increases the flexibility significantly.

- Long-term loop (blue): `sim_mode = long_term_mode`
- Short-term loop (green): `sim_mode = short_term_mode`

Note that most simulations will only consist of a subset of these four parts as, e.g. initial setup and short-term loop. The specific structure depends on the parameter choice. These four parts give the user a flexible simulation environment. The first part of LinSim, which is the initial setup, will always be executed (yellow in Fig. 2). It consists of loading the initial settings and overwriting them with the user specific ones. Then, the accelerator model and the according beam(s) are set up (e.g. for CLIC). There are many parameters related to the setup of the accelerator model, which will be explained in App. C.2 and App. C.4. Here only a few important parameters are discussed necessary for the following explanations:

- `machine_name`: The accelerator to be simulated: `ATF2`, `CLIC`, `FACET`, or `ILC`.
- `use_only_one_arm`: For a collider (CLIC and ILC), usually the e^- and the e^+ parts are simulated and the created beam can be used for beam-beam simulations. If the parameter is `1` then only the e^- arm is simulated. The beam can be collided with itself (`0` or `1`).
- `use_rtml`: The ring to main linac transfer (RTML) of the accelerator is used (values `0` or `1`).

- `use_main_linac`: The main linac of the accelerator is used (values `0` or `1`).
- `use_bds`: The beam delivery system of the accelerator is used (values `0` or `1`).
- `use_beam_beam`: The beam-beam simulations are performed (values `0` or `1`).

Clearly not all combinations of the above parameters are possible. For example, beam-beam simulations make no sense, if the beam delivery system(s) are not used. Such impossible combinations are detected by initial checks and the simulation is stopped.

After the initial setup, the code enters the first of the three interleaved loops. This first loop (orange in Fig. 2) iterates over a number of seeds (specified via `nr_machines`). Each seed corresponds to a different setup of the initial imperfections (misalignments and static component imperfections), which are created via random number generators. Iterating over many seeds is usually necessary to be able to calculate an average statistical behaviour. The use of many random seeds within one simulation is especially useful, if the individual simulations take only very short simulation time. If the individual simulations take longer, it is more appropriate to distribute the different seeds on different computers of a computer farm (see Sec. 3.4) and simply set `nr_machines` to `1`. Within the simulation of one seed, the initial misalignment and the static imperfections are set up. Then, the user specified code in the section "User code: static" of the test file is executed. This code allows to create user-specific imperfections and to test according mitigation methods. This section can correspond to the start up of the accelerator, or its reactivation after a shutdown.

As a next step, the execution enters a double loop consisting of a short-term loop (green in Fig. 2) and a long-term loop (blue in Fig. 2). Each step of the short-time loop corresponds to one beam arrival and therefore to one beam tracking. In case of CLIC this occurs every 20 ms in real time. The number of iterations can be controlled via the parameters `nr_time_steps_short`. Within each step of the short-term loop the following activities are performed:

- Apply the specified dynamic imperfections, e.g. ground motion
- Track the beam(s) through the accelerator
- Collide the two beams (if specified)
- Save the results
- Apply already implemented correction methods (if specified)
- Execute the user code in the test file under "User code: short-term"

The short-term loop is the right place to test train-to-train effects and systems as orbit feedback systems and system identification schemes in full detail.

Since real time simulations are computationally expensive, it is often not possible to simulate effects on larger time scales in full detail. Therefore, the long-term loop can be used. This loop starts with the creation of imperfection that corresponds to the time specified in the variable `delta_T_long`. Then the short-term loop is executed, and finally the user specific code in the test file under "User code: long-term" is performed. The long-term loop is repeated `nr_time_steps_long` times. It is the right place to test long-term effects such as ground motion and mitigation methods such as dispersion free steering and IP feedback.

3. Usage

3.1. Basic usage

LinSim is started in a terminal with the command

```
placet run.tcl test_settings.tcl param1 param2 ...
```

As can be seen, the code PLACET interprets the script `run.tcl`. The script `run.tcl` reads the test settings file `test_settings.tcl` in which the user has specified the simulation settings. Note that only the test settings file is passed, since the test code file is specified in the test settings file in the variable `user_code`. An arbitrary number of parameters, e.g. `param1`, can be passed by the user. These parameters are available in `test_settings.tcl` as Tcl variables. The number of parameters is stored in `argc`, and the parameters itself can be accessed via the list `argv`. The first parameter `param1` can, e.g., be used to initialise the ground motion seed with the Tcl command

```
set groundmotion(seed) [lindex $argv 1]
```

More examples for the passing of external parameters can be found in App. B.

3.2. Adding test code

The test file consists of four sections. While the first section settings of the simulation can be specified, the other three sections are used to add user code that is executed at different points in LinSim (see Sec. 2 for more explanation). The first part of code corresponds to initial misalignments, static imperfections, and initial correction methods, and has to be filled between the lines

```
if(sim_mode == static_mode)
    ...
end
```

The user code executed at the end of each step of the short-term loop has to be placed within

```
if(sim_mode == short_term_mode)
    ...
end
```

Finally, the user code executed at the end of each step of the long-term loop has to be put into

```
    if(sim_mode == long_term_mode)
        ...
    end
```

3.2.1. Simple Example

This example does a scan over different rolls of the last quadrupole QD0 for CLIC. It can be found in `LinSim/tests`.

Settings file `qd0_rollscan_settings.tcl`:

```
% code file
set user_code "tests/qd0_rollscan_code.m"
% machine
set machine_name "CLIC"
% scan values in microrad
set values_to_scan {-100 -75 -50 -25 0 25 50 75 100 0}
% number of steps, equal to values
set nr_time_steps_short [llength $values_to_scan]
% number of long time steps
set nr_time_steps_long 1
% use BDS and beam beam interaction
set use_main_linac 0
set use_bds 1
set use_beam_beam 1
% output directory
set dir_name "QD0_rollscan"
```

Code file `qd0_rollscan_code.m`

```
% static mode, store initial roll
if (sim_mode == static_mode)
    qd0_roll_start = placet_element_get_attribute('electron',
                                                  index_qd0(electron),
                                                  'roll')
end

% short time mode, apply new roll setting
if (sim_mode == short_term_mode)
    % new roll setting
    qd0_roll = values_to_scan(time_step_index_short) + qd0_roll_start
    % apply to beamline
    placet_element_set_attribute('electron',
                                 index_qd0(electron),
                                 'roll',
                                          qd0_roll)
end

if (sim_mode == long_term_mode)
    % nothing to be done here
end
```

To start the simulation in PLACET

```
  placet run.tcl tests/qd0_rollscan_settings.tcl
```

The data can be plotted with Python scripts that are provided, see section 3.5.2 for a full description.

```
  python
  import TrackingAnalysis
  a=TrackingAnalysis.MeasurementStation(directory="../QD0_rollscan/")
  a.lumiScanPlot(-100,100,25,label='QD0 Roll scan [$\mu$rad]',
plotname='QD0Roll')
```

The output is shown in Fig. 3.

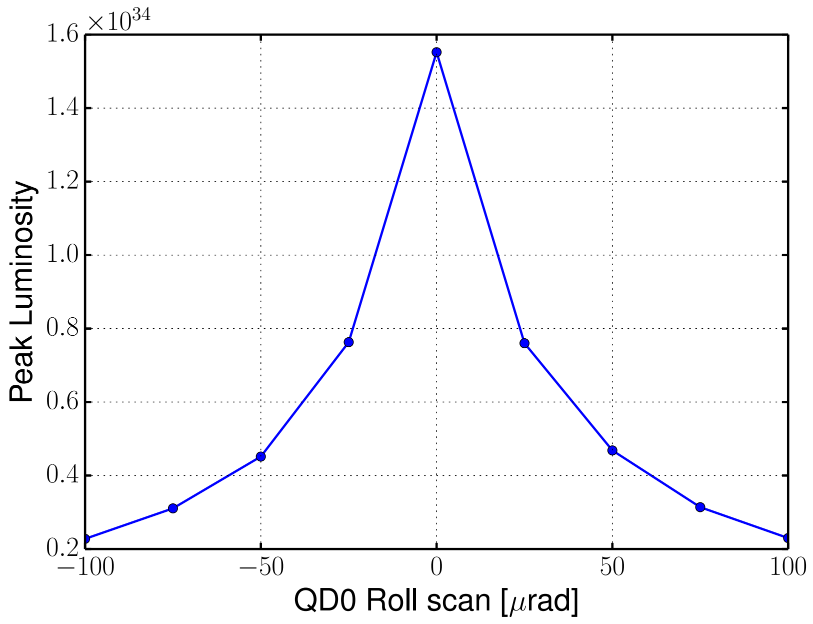

Figure 3: Peak luminosity versus QD0 roll.

More examples of test files can be found in Appendix B and `LinSim/tests/`, where also the template file `test_template.tcl` is provided.

3.3. Logging

In LinSim, the task of logging is simplified and automated as much as possible for the user. At maximum (depending on the beamline setup), there are four data sets defined that consist of the most important values. These data sets are stored automatically to according log files whenever the Octave script `postprocess_time_step.m` is called. Each data set corresponds to a different location along the accelerator (measurement stations):

- End of e^- RTML: `meas_station_0_machine_index.dat`
- End of e^- main linac: `meas_station_1_machine_index.dat`
- Final doublet of e^- beam delivery system: `meas_station_2_machine_index.dat`
- Interaction point: `meas_station_3_machine_index.dat`

The variable `machine_index` corresponds to the index in the seed loop, which means that for each seed a different set of files is created. Logging is not performed in the e^+ part of colliders. If an accelerator does not have a certain subsystem, e.g. FACET does not have a beam delivery system nor an interaction point, the according log file is not created.

The data sets are stored at all important simulations steps: after lattice setup, after static misalignment, after each short-term step, and after each long-term step. If logging is also necessary at a different location, it can be added in the user code via a simple call of `postprocess_time_step.m`. Each call creates one line of data in the log files, and each column corresponds to one value of the predefined data sets. A description of the logged values is written as a header into the log files. If additional values are needed to be stored, they have to be added in `postprocess_time_step.m`.

When analysing the simulations results, it is necessary to know which line in the log files corresponds to which simulation step. Therefore, the first three columns are the values of `time_step_index_long`, `sim_mode` and `global_time`, where `global_time` is a variable that corresponds to the duration of the simulation in real-time (as in the real machine). Additional to the described standard logging implementation, there are several other possibilities to store specific data for special analysis. The available options are described in App. C.5.

3.4. Parallel computing

In most cases, it will not be sufficient to run only one simulation, since e.g. the used imperfections are randomly distributed. Many simulation results (corresponding to different seeds) should be averaged. If the simulation of each seed takes long time (which is usually the case), the simulations of the different seeds have to be distributed over several computers (parallel computing). In this section, it is shown how this can be done using the job scripts in `LinSim/jobs/`. Here, the focus is laid on the lxbatch computing service of CERN. Also, it is assumed that LinSim is installed on a user directory on AFS.

The first step for the user is to create a bash script called `job_universal.sh`, which is the job that is passed to the lxbatch system. To created such a file, the provided template file `job_universal_template.sh` can be used. Only the two paths `basedir` and `homedir` have to be specified. The directory `basedir` is the basis directory containing also LinSim (see Sec. 4.1 for more information). The directory `homedir` is the directory where the results of the simulation should be copied. The definition of these paths usually takes a form similar to

```
export basedir=/afs/cern.ch/work/$USER/clicsim/trunk/
export homedir=/afs/cern.ch/work/$USER/sim_results/
```

The file `job_universal.sh` usually only has to be created once at the installation of LinSim. Then, it can be used to start a job by typing

```
bsub -q queue_name job_universal.sh settings_name param1 param2 ...
```

Here, `queue_name` is the queue where the job should be submitted to, e.g. `2nd` (two "normalised days"), `settings_name` is the main settings file to be used, and the parameters are values that are passed to the test file. There are many more options for lxbatch, e.g. for acquiring information about running jobs via `bpeek` and for killing running jobs via `bkill`. For more information about lxplus and AFS please refer to [8].

The typical application of LinSim on a parallel computing system is to perform the simulation of different seeds and/or parameter scans. Therefore, it is convenient not to start every job individually, but via a loop implemented in the following template scripts:

- `submit_jobs_seed_scan.sh`: Start jobs for a range of seeds.
- `submit_jobs_param_scan_1d.sh`: Start jobs for a parameter scan of one variable.
- `submit_jobs_param_scan_2d.sh`: Start jobs for a parameter scan of two variables.
- `submit_jobs_seed_param_scan.sh`: Start jobs for a range of seeds and at the same time a parameter scan of one variable.

A typical example would be the evaluation of the ground motion effects in a linear accelerator for ten different seeds. Such simulations can be started via the command

```
./submit_jobs_seed_scan_param_scan.sh tests/ground_motion_test.tcl 2 11
```

As a result, LinSim is started on ten different computers of the lxbatch system with seeds from 2 to 11. The test file `LinSim/ground_motion_test.tcl` is used for each simulation. Note that for scans in two parameters the number of started jobs grows rapidly. Also note that each job file has to have executable rights with respect to their file permissions, which can be assigned with the command

```
chmod u+x file_name.sh
```

3.5. Data evaluation tools

3.5.1. Evaluation scripts

To evaluate the results of seed and/or parameter scans the individual results have to be combined. The following templates for this propose can be found in the directory `LinSim/jobs/`:

- `eval_seed_scan.m`
- `eval_param_scan.m`
- `eval_param_scan_param_scan.m`
- `eval_param_scan_seed_scan.m`
- `eval_param_scan_seed_scan_iter_scan.m`

The usual procedure is to copy the appropriate template and assign a more specific file name. Then the first few open parameter of the script have to be specified and the data are evaluated by starting the scripts with Octave. The names are mainly self-explaining, where for long-term simulations the quantities to evaluate are only evaluated at one specified long-term step. If these quantities should be evaluated at several long-term steps, `eval_param_scan_seed_scan_iter_scan.m` can be used.

3.5.2. Plotting scripts

3.6. Tips and tricks

- One can rerun a simulation (when the exact settings are forgotten) by using the output file settings.dat:

  ```
  placet run.tcl settings.dat
  ```

- Additional command line arguments can be added in the personalised settings file, e.g.:

  ```
  set ground_motion_x [lindex $argv 1]
  set ground_motion_y [lindex $argv 2]
  ```

  This will run with ground motion on in the horizontal direction and off in the vertical direction:

  ```
  placet run.tcl my_settings.tcl 1 0
  ```

- All settings variables are available in both Tcl and Octave. They are constant in Tcl and cannot be changed. It is advised not to change them in Octave. New Octave or Tcl variables will be in principle not available in the other language. With the octave command Tcl_SetVar, variables can be transported:

```
Tcl_SetVar("machine_index", "1");
```

- To increase the number of threads/cores that should be used on a machine, add the following PLACET command to the settings:

  ```
  ParallelThreads -num 2
  ```

- To debug a new script faster, reduce the number of particles.

4. Code structure

In the following, a short overview about some aspects of the code structure of LinSim is given. Since the explanations are not a full code documentation the user is referred to the source code for aspects that are not covered.

4.1. Directory structure

The source code of LinSim is stored as part of a larger SVN directory that also includes lattice and beam descriptions of different accelerators, and other simulation setups. Not all of these directories are needed for LinSim.

The SVN base directory is `./clicsim/trunk/`. Its sub-directories correspond to the different available machines: `ATF2/`, `CLIC/`, `FACET/`, and `ILC/`. Each accelerator directory contains the sub-directories:

- `Lattices`: The lattice files of the different machines
- `Common`: Additional model information as beam creation and wake field properties
- `Frameworks`: Different simulation frameworks

The directory `./clicsim/trunk/LinSim/` contains LinSim. To use a certain accelerator, e.g. ATF2, only the directories `./clicsim/trunk/ATF/Common` and `./clicsim/trunk/ATF/Lattices` have to be checked out. An according description of the installation is given in App. A.3. Within the LinSim directory, there are the following sub-directories:

- `analysis`: Scripts to analyse the results of simulations
- `data`: Data files of smaller size, e.g. different ground motion models, stabilisation system transfer functions, and typical initial misalignments of tuned machines
- `doc`: Documentation of LinSim
- `jobs`: Scripts to facilitate parallel computing with the lxbatch system
- `scripts`: Source code of LinSim
- `tests`: User tests for LinSim

Two additional comments have to be made. Firstly, only relatively small data files are stored directly within LinSim in `data/`. More data are available, if needed, on AFS in `/afs/cern.ch/eng/clic/machine/LinSim/`. If, e.g., LinSim is installed locally and there is no direct access to lxplus, the necessary data can be copied to the local computer via ssh. The directory where the copied (or newly created) data are located has to be specified via parameter `data_dir`.

Secondly, tests files in `tests` are stored there to archive them. If they are used for simulation, a different directory structure should be implemented. For each test a separate directory should be created in `LinSim/` with the same name as the test file (apart from the ending). This setup allows to start several simulations in parallel and to store the results in separate directories. It is also the structure expected by the scripts in `jobs/` to facilitate efficient parallel computing on lxbatch.

4.2. Beamline setup

In the following it is explained how a new accelerator can be added to LinSim in addition to the already available machines ATF2, CLIC, FACET and ILC. It is assumed that this new accelerator is called ERL and that its lattice and beam models are already available in `./trunk/ERL/Common/` and `./trunk/ERL/Lattices/`. In particular, files with the following names have to be available `./trunk/ERL/Common/scripts/`:

- `wf_long_phase_ml.tcl`: Acceleration voltage phase with respect to the beam phase and long-range wakefield information
- `wf_short_ml.tcl`: Short-range wakefield model of the accelerating cavities

If the according information are not used for the ERL accelerator the files can be left empty, but still have to be created. As a next step, the following files have to be created (`LinSim/` is assumed to be the base directory) and filled according to the example of an existing accelerator:

- `./settings_erl.tcl`: Settings specific to the ERL
- `./scripts/test_settings_erl.tcl`: Parameters check with respect to their constancy (especially since not all methods are implemented for all accelerators)
- `./scripts/add_erl_lattice.tcl`: Lattice setup
- `./scripts/define_beampars_erl.tcl`: Beam parameter setup (usually defined in `./trunk/ERL/Common/scripts/beam_pars.tcl`)
- `./scripts/create_element_indices_erl.m`: Define the needed element indices specific for ERL
- `./scripts/define_instrumentation_hardware_erl.m`: Define the hardware indices used for different instrumentation methods

Calls to the files have to be added with `if-else`-statements at the following locations according (follow the examples of already existent accelerators):

- In `./scripts/load_settings.tcl` add `settings_erl.tcl`
- In `./scripts/test_settings.tcl` add `test_settings_erl.tcl`
- In `./scripts/accelerator_setup.tcl` add `add_erl_lattice.tcl` (two times) and `define_beampars_erl.tcl`:
- In `./scripts/create_bpm_corr.m` add `./scripts/create_element_indices_erl.m` and `./scripts/define_instrumentation_hardware_erl.m`

5. Support and code extension

Users of LinSim are very welcome to contact the developers:

- Jochem Snuverink (jochem.snuverink@rhul.ac.uk)
- Jürgen Pfingstner (juergen.pfingstner@cern.ch)

with their questions, suggestions for improvements, bug findings and all other issues concerning the framework. The framework LinSim is not a finished project, but should be extended and improved in the future. Every user is therefore welcome to contribute. Especially, please add your test files so that also others can profit from your expertise. Also changes in LinSim itself are welcome, but please coordinate such modifications with the developers.

For some additional information see the IPAC paper [9] and the LCWS presentation [10]. The latest version of this manual can be downloaded from the SVN server at `https://svnweb.cern.ch/cern/wsvn/clicsim/trunk/LinSim/doc/Framework_doc.pdf` (see also Section A.3) or (with a CERN userid) from `http://clicsw.web.cern.ch/clicsw/LinSim`. LinSim has nightly tests to ensure the latest version is in a good state. The status can be viewed at `http://abp-cdash.web.cern.ch/abp-cdash/index.php?project=LinSim`

A. Installation

To be able to use LinSim, it is necessary to install PLACET. If beam-beam simulations should be performed, also the GUINEA-PIG has to available. A detailed description of the usage as well as the installation of PLACET and GUINEA-PIG can be found at `http://clicsw.web.cern.ch/clicsw/` [1] and `http://clicsw.web.cern.ch/clicsw` [2], respectively. Here, only some minimal information for the installation if theses codes is extracted.

For the case, the simulations are started on AFS, it is sufficient to run the script

```
source /afs/cern.ch/eng/sl/clic-code/setup.sh
```

to set the correct paths to used PLACET and GUINEA-PIG. For the case, the simulations are started on a local computer, the necessary steps for the installation of these codes are summarised in the following. To check out source codes, the user has to have access rights to the according SVN directories. Therefore, please contact Andrea Latina (andrea.latina@cern.ch) or the developer of LinSim.

A.1. PLACET

As a requirement for the installation of PLACET several software packages have to be installed. The use of the MacPorts system [11] on a Macintosh computer is assumed in the following. For an installation on Linux, please install the according packages with an appropriate package management system. The following packages have to be installed:

- octave
- python (optional)
- tcl
- gsl (GNU Scientific Library)

Please also make sure that the latest compiler version is installed. This can be ensured easiest by typing

```
sudo port selfupdate
sudo port upgrade outdated
```

Also make sure the latest version of the "Xcode command line tools" is installed by executing

```
xcode-select --install
```

in the command line. Then, the latest versions of the code PLACET can be checked out via SVN with the expression, with CERN userid,

```
svn co svn+ssh://$USER@svn.cern.ch/reps/clicsw/trunk/placet
```

or without CERN userid:

```
svn co http://svn.cern.ch/guest/clicsw/trunk/placet
```

Finally PLACET can be installed by the series of commands

```
cd placet
./configure --prefix=[install_dir] --enable-octave
./make
./make install
```

where `[install_dir]` should be a directory in the search path and in which the user has write rights. If also the Python interface and the code parallelisation should be installed, add the parameter `--enable-python` and `--enable-mpi` to the `./configure` command.

A.2. GUINEA-PIG

To be able to install GUINEA-PIG, please install first the fftw package, which stands for "Fastest Fourier Transform in the West". Then, the C++ version of GUINEA-PIG can be checked out via, with CERN userid,

```
svn co svn+ssh://$USER@svn.cern.ch/reps/clicsw/trunk/guinea
```

or without CERN userid:

```
svn co http://svn.cern.ch/guest/clicsw/trunk/guinea
```

The code can be installed via

```
cd guinea
./configure --prefix=[install_dir] --enable-fftw3
make
make install
```

A.3. Framework

The LinSim framework is stored as part of a larger SVN directory that also includes lattice and beam models of different accelerators, and other simulation setups. To check out the full directory, execute the following command, with CERN userid,

```
svn co svn+ssh://$USER@svn.cern.ch/reps/clicsim/trunk/
```

or without CERN userid:

```
svn co http://svn.cern.ch/guest/clicsim/trunk/
```

Since the checkout includes a lot of unused files, it is suggested to follow a different approach of creating the necessary directory structure by hand and checking out only the necessary parts. This can be achieved by the following commands

```
mkdir -p clicsim/trunk
cd clicsim/trunk
svn co svn+ssh://$USER@svn.cern.ch/reps/clicsim/trunk/LinSim
mkdir CLIC
cd CLIC
svn co svn+ssh://$USER@svn.cern.ch/reps/clicsim/trunk/CLIC/Common
svn co svn+ssh://$USER@svn.cern.ch/reps/clicsim/trunk/CLIC/Lattices
```

To use also other accelerators beside CLIC the according directories have to be checked out. For the example of ILC takes the form

```
cd ..
mkdir ILC
cd ILC
svn co svn+ssh://$USER@svn.cern.ch/reps/clicsim/trunk/ILC/Common
svn co svn+ssh://$USER@svn.cern.ch/reps/clicsim/trunk/ILC/Lattices
```

The framework can also be browsed at `https://svnweb.cern.ch/cern/wsvn/clicsim/trunk/LinSim`

A.3.1. Machine without AFS

The framework assumes by default that the data directory is located on AFS in the directory `/afs/cern.ch/eng/clic/machine/LinSim/` as explained in Sec. 4.1. In case there is no direct access to AFS, the necessary data have to be copied to the local computer via ssh and the new data directory path has to be specified in the settings variable `data_dir`. However, the user is advised to not change this variable directly in the file `settings_common.tcl`, since in this change would be checked in with the next SVN commit and the AFS path would be lost as the standard. Instead, create in the directory `LinSim/` the file `my_computer.tcl` (where settings unique to the computer can be such set as the directory structure) and add the following line to specify the data directory

```
set data_dir [path_to_local_data_dir]
```

If the data directory is not needed leave `my_computer.tcl` empty.

B. Examples

Some more examples, for the simple QD0 roll example see Section 3.2.1.

B.1. Calculation of orbit response matrix

This example calculates the response matrix of the FACET beamline. It can be found `LinSim/tests`.
Settings file `facet_response_matrix_demo_settings.tcl`

```
set user_code "tests/facet_response_matrix_demo_code.m"

set machine_name "FACET"

# first argument is seed number
if {$argc > 1} {
    set global_seed [lindex $argv 1]
}

set nr_time_steps_short 23

set nr_time_steps_long 2

# static misalignment
set use_misalignment 1
set alignment_bpm_sigma 0 ;# [um]

# ground motion generator short term
set ground_motion_x 1
set ground_motion_y 1
set groundmotion(model) "B"

# ATL motion
set ground_motion_long_x 1
set ground_motion_long_y 1
set delta_T_long 3600

# Response parameters
set dipole_correctors 1
set corr_step 1 ;# [m]

set dir_name "FACETResponseMatrix"

# FACET specific
set use_only_one_arm 1
set use_rtml 0
set use_main_linac 1
set use_bds 0
set use_beam_beam 0
set use_bpm_set_value "bpm_center"
```

Code file facet_response_matrix_demo_code.m

```
if (sim_mode == static_mode)
    % apply misalignment --> already in Framework!
    R_x = zeros(nr_bpm, nr_corr);
end

if (sim_mode == short_term_mode)
    % set corrector
    placet_element_set_attribute('electron',
                             corr_index(10*(time_step_index_short)),
                             'strength_x',
                             corr_step);
    if (time_step_index_short > 1)
        placet_element_set_attribute('electron',
                             corr_index(10*(time_step_index_short-1)),
                             'strength_x',
                             0);
    end

    if (time_step_index_short == 1)
        % ref orbit
        ref_orbit_x = bpm_readings(:,x,electron);
    else
        % store results
        R_x(:,time_step_index_short-1) =
              (bpm_readings(:,x,electron) - ref_orbit_x)/corr_step;
    end
end

if (sim_mode == long_term_mode)
    % apply ATL misalignment --> already in Framework!

    % save response matrix
    filename = ['R_x', int2str(time_step_index_long), '.dat' ]
    save(filename, 'R_x', '-ascii');
    % reset for next iteration
    R_x = zeros(nr_bpm, nr_corr);
end
```

To start the simulation in PLACET with seed 3:

```
placet run.tcl tests/facet_response_matrix_demo_settings.tcl 3
```

Or to run a 20 consecutive seeds on lxbatch to the 8 hour queue:

```
cd jobs
./submit_jobs_seed_scan.sh
     tests/facet_response_matrix_settings.tcl 8nh 1 20
```

This can generate a simple response matrix plot. see Fig. 4:

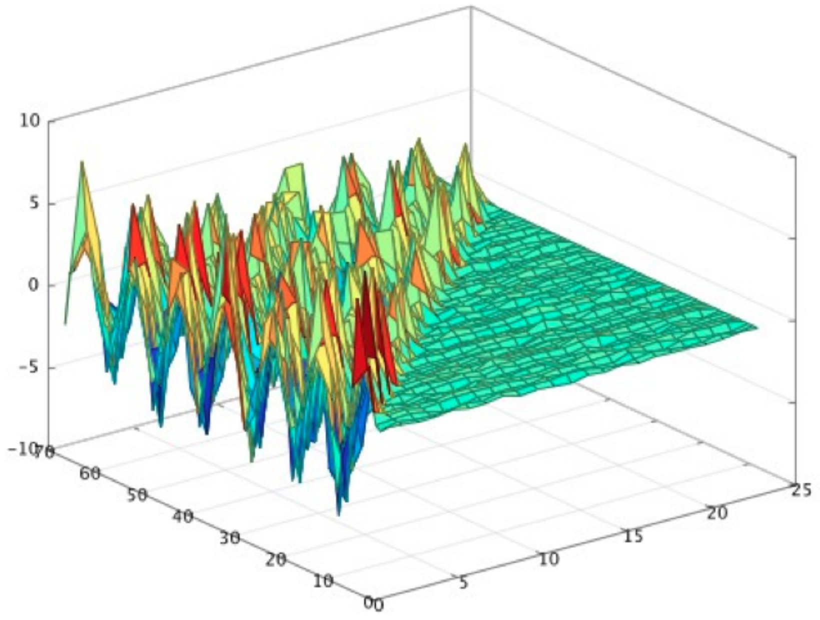

Figure 4: FACET response matrix.

C. Simulation parameter and options

In LinSim the options are divided in a settings_common.tcl for parameters common to all linear accelerators. In settings_longterm.tcl there are options specific for the long time step. For each accelerator there is also an accelerator specific file in e.g. settings_clic.tcl. In this section all options are briefly described. They are sorted by functionality.

C.1. General Simulation Parameters

- `data_dir` directory that holds large data files.
  Default value: /afs/cern.ch/eng/clic/machine/LinSim

  When no permanent AFS access, download the above directory (currently about 2.5 GB) to the local computer and set data_dir (best set in the file my_computer.tcl)

- `parallel_threads` Number of threads/cores that should be used on a machine. Default value: 1

- `dir_name` Output directory. Default value: default_output

- `machine_name` Accelerator, possible values are: CLIC, ILC, FACET and ATF2. Default value: CLIC

- `nr_machines` Number of machines (seeds) to simulate. Default value: 1

- `nr_time_steps_long` Number of iterations with a longer time per machine. Default value: 1

- `nr_time_steps_short` Number of pulse to pulse iterations (time_steps) per machine per long time step. Default value: 10

- `user_code` Individual user test file.
  Default value: "tests/user_test_template_code.m"

- `debug` Produce additional files and printout, and don't delete intermediate files. Default value: 0

- `global_seed` Global seed. Default value: 1

- `specific_random_stream` Set a specific random seed stream in PLACET. Possible values are "Misalignments", "Instrumentation", "Groundmotion", "Cavity", "User", "Default", "Survey", "Select", "Radiation". See the PLACET manual for details. Default value: "non" (no specific random seed)

- `specific_seed` Specific random seed. Only set when specific_random_stream is set to a value different than "non". Default value: "non"

C.2. Lattice Parameters

- `use_rtml` Use Ring to Main Linac lattice.
- `use_main_linac` Use Main Linac lattice.
- `use_bds` Use Beam Delivery System lattice.
- `use_only_one_arm` For colliders. Use one beamline or two (electron and positron). Default value: 0

C.3. Beam Parameters

Default values are for CLIC accelerator.

- `produce_beam_from_main_linac` If Main Linac not used, then option to produce realistic initial beam from ML lattice. Default value: 0
- `params(ch)` Default value: 3.72e9
- `params(emitt_x)` Default value: 6.6
- `params(emitt_y)` Default value: 0.2
- `params(sigma_z)` Default value: 44.0
- `params(e_spread)` Default value: 1.3
- `params(offset)` Default value: 0.0
- `params(waist_y)` Default value: 0.0
- `params(linerr)` Default value: everything
- `params(wgt)` Default value: 1000.0
- `params(quad_misalign)` Default value: 0.000
- `params(qjitter)` Default value: 0.0
- `lattice_def_file_name` Default value: ”phase.def”
- `phase` Default value: 8a
- `e_initial_linac` Default value: 9.0
- `e_initial_bds` Default value: 1500.0
- `n_slice` Default value: 25
- `n_part` Default value: 15

- `frac_lambda` Default value: 0.25
- `n_total` Default value: 50000
- `n_bunches` Default value: 1
- `n` Default value: [expr $n_total/$n_slice]
- `scale` Default value: 1.0
- `bds_version` CLIC BDS version. Current options are v_10_01_25 and v_10_10_11. Default value: "v_10_01_25"
- `use_linac_scaling` Linac scaling. Default value: 0
- `linac_scaling_factor` Scaling factor. Default value: 1
- `use_long_wf_ml` Use of longitudinal wake fields in main linac accelerating structures. Default value: 1
- `use_trans_wf_ml` Use of transverse wake fields in main linac accelerating structures. Default value: 1

C.3.1. Phase configuration

Parameter to vary the phase configuration of the main linac. The value in experimental_phase_param are only set if use_experimental_phase = 1. The gradient is scaled such that the correct energy is achieved approx. in the middle part of the accelerator.

- `use_experimental_phase_ml` Default value: 0
- `experimental_phase_params(nr_cav_0)` Default value: 144
- `experimental_phase_params(phase_0)` Default value: 8.0
- `experimental_phase_params(nr_cav_1)` Default value: 54940
- `experimental_phase_params(phase_1)` Default value: 8.0
- `experimental_phase_params(nr_cav_2)` Default value: 1000000
- `experimental_phase_params(phase_2)` Default value: 30.0
- `experimental_phase_params(gradient)` Default value: 0.0954853244727

C.3.2. Beam-beam

- `use_beam_beam` Use beam-beam interaction from GUINEA-PIG.
- `use_centred_beam_lumi` For beam-beam interaction centre beam in position. Default value: 0
- `use_centred_and_non_centred_lumi` In case of artificially centering the beam also calculate the luminosity for the non-centering case. Default value: 0

C.3.3. GUINEA-PIG

GUINEA-PIG parameters. See GUINEA-PIG manual for details.

- `gp_param(energy)` Default value: 1500.0
- `gp_param(cut_x)` Default value: 400.0
- `gp_param(cut_y)` Default value: 15.0
- `gp_param(n_x)` Default value: 128
- `gp_param(n_y)` Default value: 256
- `gp_param(do_coherent)` Default value: 1
- `gp_param(n_t)` Default value: 1
- `gp_param(charge_sign)` 0: no beam-beam force. Default value: -1.0
- `gp_param(waist_y)` Default value: $params(waist_y)
- `gp_param(particles)` Default value: [expr $params(ch)*1e-10]
- `gp_param(sigmaz)` Default value: $params(sigma_z)
- `gp_param(emitt_x)` Default value: [expr $params(emitt_x)/10.]
- `gp_param(ecm_min)` Minimum energy for lumi_high to 99% of the nominal centre-of-mass. Default value: [expr 2.0*$gp_param(energy)*0.99]
- `gp_param(seed)` Default value: [expr $global_seed*123 + 13]
- `gp_reset_seed` Guinea-Pig seed (overrides GUINEA-PIG rndm_load setting!). Default value: 1

C.4. Machine specific parameters

The machine specific parameters are found in an accelerator specific settings file e.g. settings_clic.tcl. These define the beam parameters of the machine and directory structure.

- `delta_T_short` Repetition rate in s.
- `delta_T_long` Long term time step in s.

C.4.1. ATF2

BPM resolutions

- `stripline` Resolution of stripline BPMs in μm, type 1. Default value: 1.0
- `stripline2` Resolution of stripline BPMs in μm, type 2. Default value: 1.0
- `font_stripline_resolution` Resolution of FONT stripline BPMs in μm. Default value: 0.5
- `cband` Resolution of c-band cavity BPMs in μm. Default value: 0.2
- `cband_noatt` Resolution of c-band non attenuated cavity BPMs. Default value: 0.05
- `sband` Resolution of s-band cavity BPMs in μm. Default value: 1.0
- `ipbpm` Resolution of IP BPMs in μm. Default value: 0.1
- `ipbpma` Resolution of IP BPM A in μm. Default value: value of `ipbpm`
- `ipbpmb` Resolution of IP BPM B in μm. Default value: value of `ipbpm`
- `ipbpmc` Resolution of IP BPM C in μm. Default value: value of `ipbpm`

  BPM resolution is dependent on charge with the formula $R * \sqrt(A^2/q^2 + 1)$, where R is the usual resolution and A, in 1e10 number of particles, the charge dependent part. This behaviour can be added with the following parameters:
- `use_bpm_charge_dependence` Switch on charge dependent resolution.
  Default value: 0
- `stripline_charge_dependence` Charge dependent factor for stripline BPMs. Default value: 0.69 (for 1 μm resolution)
- `font_stripline_charge_dependence` Charge dependent factor for FONT stripline BPMs. Default value: 0;(unknown)
- `cband_charge_dependence` Charge dependent factor for c-band cavity BPMs. Default value: 0.46 (for 200 nm resolution
- `cband_noatt_charge_dependence` Charge dependent factor for c-band non attenuated BPMs. Default value: 0.16 (for 30 nm resolution)
- `sband_charge_dependence` Charge dependent factor for s-band cavity BPMs. Default value: 0 (unknown)
- `ipbpm_charge_dependence` Charge dependent factor for IP BPMs. Default value: 0 (unknown)

  Add 3 IPBPMs between QF21X and QM16FF (for nanoBPM study)

- `nanoBPMSetup` Default value: 0
- `iptbpm1` Default value: `ipbpm`
- `iptbpm2` Default value: `ipbpm`
- `iptbpm3` Default value: `ipbpm`
- `skew_tilt` tilt angle of skew quadrupoles (only for lattice version v.4.0). Default value: $\pi/180 * 45$
- `beamline_version_name` Beamline version. Options are: "ATF2" (v5.2), "ATF2_v5.1", "ATF2_v4.0", "UL" (UltraLow (v4.2)) Default value: "ATF2_v5.2"
- `optics` Optics settings. Can be changed only for lattice version 5. Options are (analog to Mad8): "nominal" or "BX1BY1m" (beta*40mm x 0.1mm), BX10BY1m (beta* 4mm x 0.1mm), BX10BY1nl (beta* 4mm x 0.1mm, nonlinear optimisation by G. White). Default value: "BX10BY1nl"
- `sext` Switch sextupoles on or off. For lattice version 4.0. Default value: 1
- `use_sextupole` Switch sextupoles on or off. For lattice versions 5. Default value: 1
- `use_skewsextupoles` Switch skew sextupoles on or off. Strength is set to zero. Default value: 1
- `use_multipole_components` Switch on/off multipole components (all multipoles with length 0). Default value: 1
- `e0_start` Reference beamline energy (can be different from beam energy) Default value: 1.3 (GeV)
- `use_set_file` Set machine according to ATF2 'set' file. Default value: 0
- `set_file_name` Set file name e.g. "set13feb22_0718.dat", to be put in directory 'data'. Default value: ""
- `bunch_spacing` Bunch spacing for multi-bunch operation. Default value: 150e-9 (s)
- `use_dispersion` Add initial dispersion to beam. Default value: 0
- `dispersion_x` 0 [um]
- `dispersion_xp` 0 [urad]
- `dispersion_y` 0 [um]
- `dispersion_yp` 0 [urad]

- `use_xycoupling` Add initial xy-coupling. Default value: 0
- `xycoupling_angle` 0 [rad]
- `use_energy_jitter` Use energy jitter. Default value: 0
- `energy_jitter` 2e-4 (relative)

Values for FONT experiment

- `use_font` 0 Kicker strength in GeV*urad (for fontk1 10um change in direction fontp1 bpm corresponds to about 23.4 urad*GeV (keV))
- `fontk1_strengthx` Default value: 0.0
- `fontk1_strengthy` Default value: 0.0
- `fontk2_strengthx` Default value: 0.0
- `fontk2_strengthy` Default value: 0.0
- `ipkicker_strengthx` Default value: 0.0
- `ipkicker_strengthy` Default value: 0.0
- `use_font_jitter` Default value: 0
- `fontk1_strengthx_sigma` Default value: 0.0
- `fontk1_strengthy_sigma` Default value: 0.0
- `fontk2_strengthx_sigma` Default value: 0.0
- `fontk2_strengthy_sigma` Default value: 0.0
- `mfb2ffkicker_strengthx_sigma` Default value: 0.0
- `mfb2ffkicker_strengthy_sigma` Default value: 0.0
- `ipkicker_strengthx_sigma` Default value: 0.0
- `ipkicker_strengthy_sigma` Default value: 0.0
- `use_qd0ff_strength` Input QD0 strength. Default value: 0
- `qd0ff_strength` Default value: -0.88554 [GeV/m]
- `use_qd0ff_current` input QD0 strength in current, overrides QD0 strength setting. Default value: 0
- `qd0ff_current` Default value: 131.631 [A]

- `use_qd0ff_position` Set QD0 position. Default value: 0
- `qd0ff_positionx` Default value: 0.0 [μ m]
- `qd0ff_positiony` Default value: 0.0 [μ m]
- `qd0ff_roll` Default value: 0.0 [μ rad]
- `use_qf1ff_strength` Input QF1FF strength setting. Default value: 0
- `qf1ff_strength` Default value: 0.48034 [GeV/m]
- `use_qf1ff_current` Input QF1FF strength in current, overrides QF1FF strength setting. Default value: 0
- `qf1ff_current` Default value: 124.376 [A]
- `use_qf1ff_current_scan` Default value: 0
- `qf1ff_current_scan_step` Default value: 0.1 [A]
- `use_qf1ff_position` Default value: 0
- `qf1ff_positionx` Default value: 0.0 [μm]
- `qf1ff_positiony` Default value: 0.0 [μm]
- `qf1ff_roll` Default value: 0.0 [μrad]
- `FONTFeedback` FONT feedback in FONT region, needs use_font 1. Default value: 0
- `FONTFeedbackX` FONT feedback for horizontal direction. Default value: 0
- `FONTFeedbackY` FONT feedback for vertical direction. Default value: 1
- `FONTFeedbackP1` simple FONTfeedback on BPM FONTP1. Default value: 1
- `FONTFeedbackP2` simple FONTfeedback on BPM FONTP2. Default value: 0

  Feedback relation between bpm reading and kicker strengths.
- `FONTP1slope` FONT kicker 1 to FONTP1 [μrad*GeV/μm].
  Default value: `e_initial`/0.548 (e0 / distance K1 - P1)
- `FONTP2slope` FONT kicker 2 to FONTP2 [μrad*GeV/μm].
  Default value: `e_initial`/0.525 (e0 / distance K2 - P2)

  IP feedback with kicker at IP using IPA and IPB
- `IPFeedback` Switch on/off IP feedback. Default value: 0
- `IPFeedbackX` Switch on/off IP feedback in horizontal direction. Default value: 0

- `IPFeedbackY` Switch on/off IP feedback in horizontal direction. Default value: 1

  Feedback objective, can also be used to correct systematic bunch offset
- `IPFeedbackXObjective` Default value: 0.0 [μm]
- `IPFeedbackYObjective` Default value: 0.0 [μm]
- `IPAslope` IP kicker gain to IPA/IPB [μrad*GeV/μm]. Default value depends on lattice version (e0 / (distance kicker - IPA)
- `IPBslope` IP kicker to IPB. Default value: e0 / (distance kicker - IPB)
- `IPFeedbackIPA` Simple IPfeedback on BPM IPA. Default value: 1
- `IPFeedbackIPB` Simple IPfeedback on BPM IPB. Default value: 0

Imperfections - Dynamic and Static

- `use_beam_jitter` Switch on/off beam jitter. Default value: 0
- `beam_jitter_x` Default value: 0 [μm]
- `beam_jitter_y` Default value: 0 [μm]
- `beam_jitter_xp` Default value: 0 [μrad]
- `beam_jitter_yp` Default value: 0 [μrad]
- `use_bunch_jitter` Switch on/off bunch jitter. Default value: 0
- `bunch_jitter_x` Default value: 0 [μm]
- `bunch_jitter_y` Default value: 0 [μm]
- `bunch_jitter_xp` Default value: 0 [μrad]
- `bunch_jitter_yp` Default value: 0 [μrad]
- `use_bunch_correlation` Bunch to bunch correlation. Default value: 0
- `bunchTobunchCorrelation` Default value:. 1.0
- `use_bunch_off` Bunch offset. Default value: 0
- `bunch_offset_x` Default value: 0 [μm]
- `bunch_offset_y` Default value: 0 [μm]
- `bunch_offset_xp` Default value: 0 [μrad]
- `bunch_offset_yp` Default value: 0 [μrad]

CBPM Wakefield experiment

- `use_wakefield` Default value: 0
- `use_wakefield_shortbunch` Use short bunch approximation to calculate wakefield. Default value: 1
- `wakefield_shortbunch` Default value: 300 [μm]
- `wakeFieldSetup` Add wake field setup between QD10BFF and QD10AFF. Default value: 0 Select one of the following setups:
- `oneRefCavity` 1 reference cavity (November 2012). Default value: 0
- `twoRefCavity` 2 reference cavities (December 2012 - April 2013). Default value: 0
- `bellowWakeStudy` Bellow setup instead of 2 reference cavities. `http://atf.kek.jp/twiki/bin/view/ATFlogbook/Log20130419s`. Default value: 0
- `oneBellow` 1 bellow (May - June 2013). Default value: 0
- `oneBellowShielded` 1 bellow (June 2013) - no wakefield. Default value: 0
- `wakeFieldSetupTiltedBellow` Use tilted bellow description instead of moving bellows halfway. Default value: 0
- `wakeFieldSetupQuadMove` Set move subsequent quadrupoles also a little. Default value: 0
- `wakeFieldSetup_pos` Wakefield setup position. Default value: 0 [mm]. Should be between -4.6mm and +4.6mm
- `wakeFieldSetup_scan` Increase position by 1.0 mm per time step in y. Default value: 0

Steering

Response matrix calculation

- `response_matrix_calc` Calculate response matrix. Default value: 0
- `step_size_R_x` Horizontal step size to calculate response matrix. Default value: 1
- `step_size_R_y` Vertical step size to calculate response matrix. Default value: 1
- `response_matrix_load` Load response matrix. Default value: 0
- `response_matrix_file` File name of loaded response matrix.
  Default value: "../Response/Response.dat"

- `use_bpm_set_value` Value has to be ”bpm_center” for ATF2.
- `use_steering` Apply any steering (need to choose one or more). Default value: 0
- `steering_iter` Number of iterations. Default value: 3
- `steering_select_bpms` Option to select a subset of bpms used for DFS or WFS (see machine_setup.m). Default value: 0
- `steering_beta` Beta parameter, cuts on singular values (the higher, the harder the cut). Default value: 6
- `use_one_to_one_steering` One to one steering. Default value: 0
- `one_to_one_steering_weight` 1-to-1 steering weight. Default value: 5
- `one_to_one_steering_start` Bunch number start for 1-to-1 steering. Default value: -1
- `one_to_one_steering_end` Bunch number end. Default value: 9999
- `use_dfs` Dispersion free steering. Default value: 0
- `dfs_dE` Relative energy of second beam. Default value: 0.995
- `dfs_weight` Default value: 5
- `use_wfs` Wakefield free steering. Default value: 0
- `wfs_dcharge` Relative charge of second beam. Default value: 0.8
- `wfs_weight` Omega weight, dependent on bpm_resolution and alignment. Default value: 5

System identification

- `identification_switch` Default value: 0
- `alpha_x` Alpha has for the ATF2 simulations the meaning of a projected beam size growth in percent. Default value: 5
- `alpha_y` Default value: 5
- `lambda_x` Default value: 0.997
- `lambda_y` Default value: 0.997
- `rls_algo` 1 ... standard algo, 2 ... modified algo, 3 ... full algo. Default value: 3
- `init_P_algo` 1 ... diag(0.1), 2 ... find_init_P function. Default value: 2

- `init_P_value` Default value: 0.1
- `excit_type` 1 ... random excit, 2 ... one corrector after the other excit (necessary for rls_algo 6), note the excit_type 3 which is an excitation with constant maximal BPM offset due to the excitation, is not supported so far. 4 ... corresponds to a test case for the initial scaling of the emittance growth (determination of factors). Default value: 1

Postprocessing Parameters

- `save_bpm_readings_nonoise` Save BPM readings without noise. Default value: 0
- `bpm_readings_file_name_x` Default value: "bpmreadingsx.dat"
- `bpm_readings_file_name_y` Default value: "bpmreadingsy.dat"
- `magnetPosition_readings_file_name_x` Default value: "magnetPositionreadingsx.dat"
- `magnetPosition_readings_file_name_y` Default value: "magnetPositionreadingsy.dat"
- `ip_feedback_file_name` Default value: "IPFeedback.dat"
- `meas_station_file_name` Default value: "meas_station.dat"
- `save_beam_ff_start` Save beam at start of Final Focus (before MQM16FF). Default value: 1
- `meas_station_ff_start_file_name` Default value: "meas_station_ff_start.dat"
- `save_beam_mfb2ff` Save beam at MFB2FF. Default value: 1
- `meas_station_mfb2ff_file_name` Default value: "meas_station_mfb2ff.dat"
- `save_beam_ip` Save beam at IP. Default value: 1
- `meas_station_ip_file_name` Default value: "meas_station_ip.dat"
- `use_disp_free_emittance` Emittance calculated as if dispersion is corrected. Default value: 0

C.4.2. CLIC

- `use_collimators` For BDS version 10_01_25 only. Default value: 0
- `use_nonlinear_collimation` For BDS version 10_01_25 only. Default value: 0
- `wmetode` Wake method calculation 0: analytical formula, 1: wakefield tables. Default value: 0

- `taperl` taperl defines the length of the tapered part of the spoiler (here it is defined to conform 88 mrad taper angle). Default value: 0.09

C.4.3. FACET

No specific parameters are defined for FACET at the moment.

C.4.4. ILC

- `tdr_bds_params_elec` and `tdr_bds_params_posi` Technical Design Report parameters of the BDS.

C.5. Logging Parameters

- `multipulse_nr` Number of bunch trains that are used to calculated the multipulse quantities. Default value: 10
- `save_meas_stations` Store data at the measurement stations (after RTML, after ML, before FD and at IP). Switching off will remove the first initial tracking step. Default value: 1
- `add_header` Add headers to file. Default value: 1
- `save_string` Additional string to be added to output filenames. Default value ""
- `save_beam_data` Defines if in general, the beams itself and the according correlation data should be saved. Sometimes this is not necessary and/or would corrupt the multi-pulse calculation. Default value: 1
- `save_beam_file_ip` Save beam files at IP (careful 10 MB per iteration!). Default value: 0
- `save_beam_file_freq` Frequency of saving beam files (once every x iterations). Default value: 50
- `twiss_output` Output twiss parameter file. Default value: 0
- `eval_beam_shift` Include beam offsets and angles to calculate multipulse emittance. Default value: 1
- `use_bpm_storing` Default value: 0
- `bpm_storing_no_noise` Default value: 1
- `use_gm_storing` Default value: 0
- `use_ip_corr_storing` Default value: 0
- `use_ml_corr_storing` Default value: 0

- `use_machine_status_storing` Default value: 0
- `load_machine` Load machine status (to be saved with use_machine_status_storing). Default value: 0
- `load_machine_electron` Electron beamline, absolute path.
  Default value: "${script_dir}/machine_status_electron_1.dat"
- `load_machine_positron` Positron beamline, absolute path.
  Default value: "${script_dir}/machine_status_positron_1.dat"
- `use_bpm_callback` Callback function in BPMs, to store the beam in specific BPMs for beam physics analysis. Default value: 0
- `bpm_callback_start_index` First BPM to call callback function. Default value: 1
- `bpm_callback_end_index` Last BPM to call callback function. If it is < 0.5 it means: go to the end. Default value: -1
- `bpm_callback_function` Callback function. Values are:
  1: store the full beam
  2: store the beam parameters.
  Default value: 1
- `use_qp_callback` Callback function in quadrupoles, to store the beam in specific quadrupoles to calculate the multi-pulse emittance. Default value: 0
- `qp_callback_start_index` First quad to call callback function. Default value: 1
- `qp_callback_end_index` Last quad to call callback function. Default value: -1
- `qp_callback_function` Callback function. Values are:
  1: store the full beam
  2: store the beam parameters.
  Default value: 1
- `use_storage_in_qps` Options to store beam information (different than BPM readings) all along the beam line (in the QPs). Default value: 0
- `use_storage_in_qps_mpe` Multipulse emittance. Default value: 0
- `use_storage_in_qps_spe` Single pulse emittance. Default value: 0
- `use_storage_in_qps_sigmaE` Sigma energy. Default value: 0
- `use_storage_in_qps_E` Energy. Default value: 0
- `tracking_output_string` Output for tracking in each element. See PLACET manual for details. Default value: "%s %ex %sex %x 0 %ey %sey %y %Env 0 %n" (s position; emittance x; beamsize x; x position; 0; emittance y; beamsize y; energy; number of particles)

C.6. Imperfections

- `bpm_noise` Use the BPM resolution, if 0 then 0.0 resolution. Default value: 1
- `bpm_resolution_ml` BPM resolution Main Linac in μmeter. Default value: 0.1
- `bpm_resolution_bds` BPM resolution BDS in μmeter. Default value: 0.05
- `stabilization_noise_x` Default value: 0
- `stabilization_noise_y` Default value: 0
- `stabilization_noise_type` Default value: "white_uniform"
- `stabilization_noise_param` Default value: 0.001
- `stabilization_noise_section` Parameter could be 'ALL', 'ML', 'BDS' . Default value: "ALL"
- `stabilization_scaling_error_x` Default value: 0
- `stabilization_scaling_error_y` Default value: 0
- `stabilization_scaling_sigma` Default value: 0.001
- `stabilization_scaling_section` Parameter could be 'ALL', 'ML', 'BDS'. Default value: "ALL"
- `bpm_scaling_error_x` Default value: 0
- `bpm_scaling_error_y` Default value: 0
- `bpm_scaling_sigma` Default value: 0.001
- `bpm_scaling_section` Parameter could be 'ALL', 'ML', 'BDS'. Default value: "ALL"

C.6.1. RF jitter variables

Each default value corresponds to 1% lumi loss. Phi is in degree and grad in GV/m

- `use_rf_jitter` Default value: 0

  For random train to train jitter
- `phi_linac` Default value: 0.0
- `phi_decel` Default value: 0.0
- `grad_linac` Default value: 0.00000
- `grad_decel` Default value: 0.0000

  For random jitter constant for one machine

- `phi_linac_machine` Default value: 0.0
- `phi_decel_machine` Default value: 0.0
- `grad_linac_machine` Default value: 0.0
- `grad_decel_machine` Default value: 0.0
  For a deterministic acceleration change for one machine
- `phi_linac_machine_const` Default value: 0.0
- `grad_linac_machine_const` Default value: 0.0
  Procedure to remove the orbit created by dispersion
- `use_disp_suppression_x` Default value: 0
- `use_disp_suppression_y` Default value: 0

C.6.2. Initial beam jitter

Initial jitter: 1 ... beam offset/angle jitter, 2 ... energy jitter, 3 ... energy chirp, 13 ... beam offset/angle jitter and energy chirp

- `initial_imperfection_type_x` Default value: 0
- `initial_imperfection_type_y` Default value: 0
- `initial_jitter_x` in μm. Default value: 0
- `initial_jitter_y` in μm. Default value: 0
- `initial_energy_jitter` in GeV. Default value: 0
- `initial_energy_chirp` in %. Default value: 0

C.6.3. Constant initial beam energy variation

- `use_initial_beam_variation` Default value: 0
- `initial_energy_var_const` Default value: 0.0

C.6.4. Quadrupole strength jitter

- `use_quad_strength_jitter` Default value: 0
- `quad_strength_sigma_ml` Relative. Default value: 0
- `quad_strength_sigma_bds` Default value: 0
- `quad_strength_sigma_ff` Default value: 0

- `quad_strength_sigma_fd` Default value: 0
- `quad_strength_sigma_ml_machine` Relative. Default value: 0
- `quad_strength_sigma_bds_machine` Default value: 0
- `quad_strength_sigma_ff_machine` Default value: 0
- `quad_strength_sigma_fd_machine` Default value: 0

C.6.5. Quadrupole position jitter

- `use_quad_position_jitter` Default value: 0
- `quad_position_sigma_ml` in μm. Default value: 0
- `quad_position_sigma_bds` Default value: 0
- `quad_position_sigma_ff` Default value: 0
- `quad_position_sigma_fd` Default value: 0

C.6.6. Quadrupole roll jitter

- `use_quad_roll_jitter` Default value: 0
- `quad_roll_sigma_ml` in μm. Default value: 0
- `quad_roll_sigma_bds` Default value: 0
- `quad_roll_sigma_ff` Default value: 0
- `quad_roll_sigma_fd` Default value: 0
- `use_quad_roll_actuation` Default value: 0
- `quad_roll_x_const` in μrad/μm. Default value: 5.15

C.6.7. System Identification parameters

- `identification_switch` Default value: 0
- `perfect_controller_matrix` Default value: 0

C.6.8. BPM drifts

- `use_bpm_drift` Default value: 0
- `bpm_drift_sigma_ml` in μm. Default value: 0.44
- `bpm_drift_sigma_bds` in μm. Default value: 0.44
- `bpm_drift_sigma_ff` in μm. Default value: 0
- `bpm_drift_sigma_fd` in μm. Default value: 0

C.6.9. component failures (static)

Possible probability distributions, "random", "block_begin", "block_end", "block_end_ml", "block_begin_bds", "list"

- `use_bpm_failure` Default value: 0
- `bpm_failure(prob)` Chance of a failed BPM. Default value: 0
- `bpm_failure(list)` Element numbers of failed BPMs (needs distribution list). Default value:
- `bpm_failure(distribution)` Default value: "random"
- `use_corr_failure` Default value: 0
- `corr_failure(prob)` Chance of a failed corrector. Default value: 0
- `corr_failure(list)` Element numbers of failed correctors (needs distribution list). Default value:
- `corr_failure(distribution)` Default value: "random"
- `use_quad_stab_failure` Default value: 0
- `quad_stab_failure(prob)` Chance of a failed quadrupole. Default value: 0
- `quad_stab_failure(list)` Element numbers of failed quadrupoles (needs distribution list). Default value:
- `quad_stab_failure(distribution)` Default value: "random"

C.6.10. add noise to response matrix

- `use_noisy_matrix_x` Default value: 0
- `use_noisy_matrix_y` Default value: 0
- `R_file_name_x` Default value: "R_noisy_x.dat"
- `R_file_name_y` Default value: "R_noisy_y.dat"

C.6.11. L-FB gain errors

- `use_gain_error_x` Default value: 0
- `use_gain_error_y` Default value: 0
- `gain_error_std_x` Default value: 0.1
- `gain_error_std_y` Default value: 0.1

C.6.12. Wake field monitor offset

- `use_wakefield_monitor_imperfections` Default value: 0
- `wakefield_monitor_offset_x` in μm. Standard value: 3.5. Default value: 0
- `wakefield_monitor_offset_y` in μm. Standard value: 3.5. Default value: 0

C.7. Ground Motion

- `ground_motion_x` Default value: 0
- `ground_motion_y` Default value: 0
- `groundmotion(type)` 0: none, 1: ATL, 2: ground motion generator. Default value: 2
- `groundmotion(model)` Ground motion model. Options are "A", "B", "B10", "C", "D". Default value: "B"
- `groundmotion(filtertype)` 0: no filter, 1: use filters specified in x and y for the whole beamline. Default value: 1
- `groundmotion(filtertype_x)` Default value: "local_FB_lateral_v2.dat"
- `groundmotion(filtertype_y)` Default value: "local_FB_vertical_v2.dat"
- `groundmotion(preisolator)` 0: no preisolator, 1: simple version F. Ramos et al, 2: mech. feedback B. Caron et al., 3: F. Ramos et al. including tilt motion. Default value: 3
- `groundmotion(t_start)` Start after perfect beamline in s. Default value: 0
- `gm_ip0` Move ground with respect to IP. Default value: 1
- `gm_cav_stab` Move cavities with stabilisation. Default value: 0
- `ground_motion_long_x` Ground motion long term, ATL.Default value: 0
- `ground_motion_long_y` Default value: 0

- `use_initial_ATL` Initial static misalignment with ATL. Default value: 0
- `delta_T_static` Time for ATL static misalignment in s. Default value: 0
- `groundmotion_long(ml)` Default value: 1
- `groundmotion_long(bds)` Default value: 1
- `groundmotion_long(time_step)` Default value: $delta_T_long

C.8. Alignment

- `use_misalignment` Start from (non-)perfect machine. Default value: 0
  Misalignments in both x and y
- `alignment_bpm_sigma` in μm. Default value: 10
- `alignment_dipole_sigma` in μm.Default value: 10
- `alignment_quad_sigma` in μm.Default value: 10
- `alignment_quad_roll_sigma` in μm.Default value: 100
- `alignment_mult_sigma` in μrad. Default value: 100
- `alignment_struct_tilt_sigma` in μrad. Default value: 140

C.9. Tracking

Specific tracking options.

- `use_multipole` Switch multipoles (sextupoles and higher order) on or off. Default value: 0

C.9.1. Synchrotron radiation

Default is off for main linac and RTML for CPU reasons, on for BDS. Always on for sbends.

- `quad_synrad_rtml` Synchrotron radiation in quadrupoles in RTML. Default value: 0
- `mult_synrad_rtml` Synchrotron radiation in multipoles in RTML. Default value: 0
- `sbend_synrad_rtml` Synchrotron radiation in sbends in RTML. Default value: 0
- `quad_synrad_ml` Synchrotron radiation in quadrupoles in Main Linac. Default value: 0

- `mult_synrad_ml` Synchrotron radiation in multipoles in Main Linac. Default value: 0
- `sbend_synrad_ml` Synchrotron radiation in sbends in Main Linac. Default value: 0
- `quad_synrad_bds` Synchrotron radiation in quadrupoles in BDS. Default value: 0
- `mult_synrad_bds` Synchrotron radiation in multipoles in BDS. Default value: 0
- `sbend_synrad_bds` Synchrotron radiation in sbends in BDS. Default value: 0

C.10. Orbit feedback

Controller Parameters. See orbit controller paper for details.

- `use_controller_x` Default value: 0
- `use_controller_y` Default value: 0
- `dipole_correctors` Dipole correctors or quadrupole offsets as orbit correctors. Default value: 0
- `controller_type_spatial_x` Controller type 1: full matrix, 2: non-scaling case, 3: scaling case, 4: loaded coefficients, 5: experimental case. Default value: 4
- `controller_type_spatial_y` Default value: 4
- `gain_file_name_x` File name of gains in x.
  Default value: "gains_B_V2_x_V2_load.dat"
- `gain_file_name_y` File name of gains in y.
  Default value: "gains_B_V2_y_V2_load.dat"
- `max_gain_x` Default value: 0.01
- `max_gain_y` Default value: 0.01
- `nr_sv1_x` Default value: 16
- `nr_sv1_y` Default value: 16
- `nr_sv2_x` Default value: 300
- `nr_sv2_y` Default value: 300
- `controller_gain1_y` Default value: 1
- `controller_gain1_x` Default value: 1
- `controller_gain2_y` Default value: 0.05

- `controller_gain2_x` Default value: 0.05
- `controller_gain3_y` Default value: 0.0001
- `controller_gain3_x` Default value: 0.0001
- `controller_type_frequency_x` Controller type 1: integrator, 2: integrator, low-pass (no lead), 3: complex controller (see below). Default value: 3
- `controller_type_frequency_y` Default value: 3
- `T_low` Time constant for the low-pass. Default value: 0.1;
- `orbit_filter_algo_x` Orbit_filter_algo
  1: explicit but complicated implementation (at the moment just integrator, low-pass and lead).
  2: short and fast implementation (integrator, low-pass, lead element and 0.3 Hz peak).
  Default value: 2
- `orbit_filter_algo_y` Default value: 2
- `use_bpm_set_value` Value can be "dispersion_init" or "bpm_center". Default value: "dispersion_init"
- `use_bpm_scaling` Use a scaling of the BPM readings for the controller. Default value: 0

C.11. IP feedback Parameters

IP feedback correction is independent of orbit correction. There are two types, from train to train (intertrain) and within one train (intratrain). For historical reasons they are called ip_feedback and ip_intratrain_feedback

C.11.1. Intertrain feedback

- `use_ip_feedback` Default value: 0
- `use_ip_feedback_x` Default value: 1
- `use_ip_feedback_y` Default value: 1

  Different ip feedback: linear ip feedback, second order (pid) or Annecy (LAPP) optimised. For x always linear feedback is used.
- `use_ip_feedback_linear` Default value: 0
- `gain_bb_x` Gain factor for linear feedback. Default value: 0.5
- `gain_bb_y` Default value: 0.5

- `use_ip_feedback_pid` Default value: 1
- `pid_gain_p` Default value: 0.3
- `pid_gain_d` Default value: 0.5
- `use_ip_feedback_annecy` Default value: 0
- `ip_feedback_annecy_choice` 0: fb 1:fba 2:pid 3:pida. Default value: 1

Extra Annecy options

- `external_ground_cms` Default value: 0
- `use_defl_angle` Default value: 1
- `outputfile_ip` Default value: 0
- `outputfile_ip_controller` Default value: "Beam_correction_log_4"

Properties of the BPM after the IP

- `bpm_ip_dist` Distance BPM-IP in m. Default value: 3.0
- `bpm_ip_noise` Use BPM resolution. Default value: 0
- `bpm_ip_res` Resolution of BPM at IP in μm. Default value: 1.0

C.11.2. Intratrain feedback

- `use_ip_intra_feedback` Default value: 0
- `ip_intra_latency` Latency time of system in ns. Default value: 37
- `gain_bb_intra_x` Gain factors for linear feedback. Default value: 0.28
- `gain_bb_intra_y` Gain factors for linear feedback. Default value: 0.28

Properties of the BPM after the IP

- `bpm_ip_intra_dist` Distance BPM-IP in m. Default value: 3.0
- `bpm_ip_intra_noise` Use BPM resolution. Default value: 0
- `bpm_ip_intra_res` Resolution of BPM at IP μm. Default value: 1.0

C.12. BDS Tuning

- `use_bds_knobs` Use of BDS knobs. Default value: 0
- `use_steering` Default value: 0

Response matrix can be loaded with R_file_name_x and R_file_name_y

C.12.1. Use of Beam based alignment

- `use_one_to_one_steering` Default value: 0
- `one_to_one_steering_weight` Default value: 0.5

C.12.2. Dispersion Free Steering

- `use_dfs` Default value: 0
- `dfs_deltae` Default value: 0.001
- `dfs_beta` Default value: 1
- `dfs_weight` Default value: 1.0

C.13. Long term steering

- `use_bds_knobs_longterm` Default value: 0

C.13.1. Basic steering

- `use_basic_steering` Default value: 0
- `basic_steering_method`
  1: 1-1 steering
  2: all to all steering.
  Default value: 2
- `basic_steering_option`
  1: corr everywhere
  2: corr ML
  3: corr ML - BPM ML
  Default value: 1
- `basic_steering_matrix_name_x` Default value: "R0_x_complete.dat"
- `basic_steering_matrix_name_y` Default value: "R0_y_complete.dat"
- `basic_steering_modulo` Default value: 1
- `basic_steering_beta` Default value: 1

C.13.2. Online dispersion estimation / Dispersion free steering

- `use_online_disp_estimation` Default value: 0
- `use_initial_rfalign` Default value: 1
- `use_dfs_local_excitation` Default value: 1
- `dfs_estimation_cycles` Default value: 25
- `use_disp_estimation_storing` Default value: 0
- `dfs_delta_grad` Default value: 0.01
- `dfs_delta_phi` Default value: 0.00
- `use_theoretical_dfs_omega` Default value: 1
- `dfs_omega_value` Default value: 1
- `dfs_beta_value` Default value: 1
- `dfs_nr_bins` Default value: 36
- `dfs_bin_overlap` 0.5: full overlap. Default value: 0.5
- `dfs_disp_noise` Default value: 0
- `dfs_matrix_noise` Default value: 0
- `dfs_nr_runs` Default value: 1
- `dfs_gain` Default value: 1
- `dfs_delta_T` Default value: $delta_T_long
- `use_dfs_imperfect_sys_knowledge` Default value: 0
- `dfs_imperfect_sys_knowledge_y_steering_file_name` Default value: "/energy_imperfection/grad_jitter_linac_decel/Delta_E_0_0/qp/y/R_qp_y_energy_jitter_linac_0001_decel_0005_seed_1.dat.gz"
- `dfs_imperfect_sys_knowledge_y_dfs_file_name` Default value: "/energy_imperfection/grad_jitter_linac_decel/Delta_E_0_001/qp/y/R_qp_y_energy_jitter_linac_0001_decel_0005_seed_1.dat.gz"

C.14. Specific Simulation Parameters

- `response_matrix_calc` Calculate response matrix mode, will exit after finished. Default value: 0
- `response_matrix_calc_longterm` Default value: 0
- `use_fd_model` Use Final Doublet model to predict beam position at IP. Default value: 0
- `load_beam_bds` Reload beam after bds. Default value: 0

D. Description of output files

- `acc.dat` A file for Guinea-Pig that holds information about how the collision was simulated.
- `beam.dat` The last beam that was tracked. This beam can be read into PLACET and used in a subsequent simulation.
- `girder_elec/positron.dat` A file that holds the offsets of all the girders in the elec/positron line.
- `meas_station_`*n*`_machine_`*n*`.dat` A file that has recorded several parameters at a certain location in the beamline. See section 3.3 for the default locations.
- `rndm.save` The initial state of the random number generator of Guinea-Pig.
- `settings.dat` The settings that were used for this simulation.
- `tracking_results_1.dat` A file that holds beam data at every quadrupole location.

  Additional output files can be saved with various options, see Section C.5.

E. Description of parameters available for user scripts

This section gives a selection of the parameters and variables readily available in user scripts. The list is certainly not exhaustive.

E.1. Element indices

The indices are a $n \times m$ matrix where n is the number of beamlines (1 or 2) and m the number of indices. These can be used to set properties of the element with `placet_set_element_attribute`.

General element types:

- `all_index` Indices of all elements.

- `bpm_index` Indices of BPMs.
- `corr_index` Indices of all correctors used in the orbit feedback (dipoles or quadrupoles as specified).
- `drift_index` Indices of drifts.
- `dipole_index` Indices of dipoles.
- `qp_index` Indices of quadrupoles.
- `mult_index` Indices of multipoles (sextupoles and higher).
- `cav_index` Indices of accelerating cavities.

Many specific elements indices are also defined. This list is not exhaustive, some examples:

- `index_bpm/dipole_rtml/ml/bds_start/end` The index of the first / last bpm / dipole of the RTML, ML or BDS.
- `index_qd0` The index of QD0 magnet.

E.2. Accelerator and simulation status

Many parameters about the current status of the accelerator are available. Here are some examples:

- `bpm_readings` The BPM readings including noise (if specified). A 3-d array (x,y,beamline).
- `tracking_results.Lumi(_high)` The (peak) luminosity of the last bunch crossing.
- `tracking_results.Angle_x/y` The horizontal/vertical beam deflection angle.
- `time_step_index_short/long` The short/long time step where the simulation is at.
- `sim_mode` The mode where the execution is in. This is explained in Section 2.

F. Python Analysis Documentation

TrackingAnalysis Documentation, Release 1.0

Tracking Analysis Classes

if used within PLACET use a non-interactive backend like Agg http://matplotlib.org/faq/usage_faq:

```
_mpl.use('Agg')
```

class `TrackingAnalysis.`**`BeamlineStudy`**(*directory='output/', fileNameX='gm_x_elec_machine_1.dat', fileNameY='gm_y_elec_machine_1.dat', begin=None, end=None, s=[], color='b', figureNumber=200, title=''*)

Class for plotting the beamline from a text file

Example:

```
import TrackingStudy
TrackingAnalysis.BeamlineStudy()
```

class `TrackingAnalysis.`**`MeasurementStation`**(*directory='output/', fileName='meas_station_3_machine_1.dat', label='', new=False*)

Class for plotting results from a text file

Example:

```
import TrackingAnalysis
TrackingAnalysis.MeasurementStation()
```

class `TrackingAnalysis.`**`MeasurementStationArray`**(*directory='', label='', color='b', parameter=1, figureNr=1000*)

Class for plotting results from a list of MeasurementStation

Example:

```
import TrackingAnalysis
TrackingAnalysis.MeasurementStationArray()
```

class `TrackingAnalysis.`**`OrbitStudy`**(*directory='output/', fileNameX='bpm_meas_x_elec_machine_1.dat', fileNameY='bpm_meas_y_elec_machine_1.dat', begin=None, end=None, onlyLast=True, s=[], color='b', figureNumber=100, title=''*)

Class for plotting the orbit from a text file

Example:: import TrackingStudy TrackingAnalysis.OrbitStudy()

class `TrackingAnalysis.`**`TrackingStudy`**(*directory='output/', fileName='tracking_results_1.mat', fileNameNominal='', label='tracking', machineNumber=1, positronLine=False, orbit=False, beamline=False, beamPlot=False*)

Class for plotting the tracking results from a text file

Example:

```
import TrackingAnalysis
TrackingAnalysis.TrackingStudy()
```

G. Brief introduction to scripting languages

G.1. Tcl

Tcl (Tool Command Language) is a scripting language created by John Ousterhout [4]. The following is taken from `http://en.wikipedia.org/wiki/Tcl#Features`:

A Tcl script consists of several command invocations. A command invocation is a list of words separated by whitespace and terminated by a newline or semicolon.

```
word0 word1 word2 ... wordN
```

The first word is the name of a command, which is not built into the language, but which is in the library. The following words are arguments. So we have:

```
commandName argument1 argument2 ... argumentN
```

An example, using the *puts* command to display a string on the host console, is:

```
puts "Hello, World!"
```

This sends the string ”Hello, World!” to the 'stdout' device, with an appended newline character. Variables and the results of other commands can be substituted inside strings too, such as in this example where we use *set* and *expr* to store a calculation result in a variable, and puts to print the result together with some explanatory text:

```
# expr evaluates text string as an expression
set sum [expr 1+2+3+4+5]
puts "The sum of the numbers 1..5 is $sum."

# with variables, it is faster to protect this string using curly braces
set x 1
set sum [expr {$x + 2 + 3 + 4 + 5}]
puts "The sum of the numbers 1..5 is $sum."

# without curly braces, variables are substituted even before parsing
# the expression
set x 2
set op *
set y 3
set res [expr $x$op$y]
puts "2 * 3 is $res."
```

There is one basic construct (the command) and a set of simple substitution rules.

Formally, words are either written as-is, with double-quotes around them (allowing whitespace characters to be embedded), or with curly- brace characters around them, which suppresses all substitutions inside (except for backslash-newline elimination). In bare and double- quoted words, three types of substitution occur (once, in a single left-to-right scan through the word):

- **Command substitution** replaces the contents of balanced square brackets with the result of evaluating the script contained inside. For example, [expr 1+2+3] is replaced with the result of evaluating the contained expression (i.e. 6) since that's what the expr command does.

- **Variable substitution** replaces a dollar-sign followed by the name of a variable with the contents of the variable. For example, "$foo" is replaced with the contents of the variable called "foo". The variable name may be surrounded in curly braces so as to delimit what is and isn't the variable name in otherwise ambiguous cases.
- **Backslash substitution** replaces a backslash followed by a letter with another character. For example, "\n" is replaced with a newline.

G.2. Octave

GNU Octave [5] is a high-level interpreted language, primarily intended for numerical computations. It provides capabilities for the numerical solution of linear and nonlinear problems, and for performing other numerical experiments. It also provides extensive graphics capabilities for data visualization and manipulation. The Octave language is quite similar to Matlab:

- Matrices as fundamental data type.
- Built-in support for complex numbers.
- Powerful built-in math functions and extensive function libraries.
- Extensibility in the form of user-defined functions.

G.3. Python

Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasises code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java. The language provides constructs intended to enable clear programs on both a small and large scale. Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library.

Python uses whitespace indentation, rather than curly braces or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Here is a simple python function that demonstrates this:

```
def fib(n):
    print 'n =', n
    if n > 1:
        return n * fib(n - 1)
    else:
        print 'end of the line'
        return 1
```

References

[1] D. Schulte et al., *The PLACET project*, `http://clicsw.web.cern.ch/clicsw`, CERN.

[2] D. Schulte. *Study of Electromagnetic and Hadronic Background in the Interaction Region of the TESLA Collider*. PhD thesis, Universität Hamburg, 1996.

[3] C. M. Pilato, B. Collins-Sussman and B. W. Fitzpatrick. *Version Control with Subversion*. O'Reilly. 2004. ISBN 0-596-00448-6. full book online.

[4] J. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley Professional Computing Series, 1994. ISBN: 0-201-63337-X.

[5] J. W. Eaton. *Octave*. Technical report. `https://www.gnu.org/software/octave`.

[6] Y. Renier, P. Bambade, and A. Sery. *Tuning of a 2D ground motion generator for ATF2*. Technical Report LAL/RT 08-18, CARE/ELAN document-2008-005, ATF-08-10, LAL, 2008.

[7] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley. 1997. ISBN 0-201-88954-4.

[8] *Batch Service*, `http://information-technology.web.cern.ch/services/batch`, CERN.

[9] J. Snuverink, J. Pfingstner, N. Fuster-Martinez, In Proc. of IPAC15, MOPJE029, 2015.

[10] J. Snuverink, J. Pfingstner, Linear Collider Workshop 2014. `http://agenda.linearcollider.org/event/6389/session/10/contribution/105/material/slides/0.pdf`.

[11] *The MacPorts Project*, `www.macports.org/`.