TRAIN Reference Manual

Ariadna Ribes Metidieri
Xavier Buffat

BE-ABP-HSC
CERN - Geneva, Switzerland

Abstract

This document is meant to be both a user and a developer guide to comprehend the structure and functioning of the program TRAIN, as well as the results that can be obtained through its usage.

The first chapter tries to provide an overview of the knowledge required to work with TRAIN as a user, without detailing the code. The information given here should be enough as to execute the code and understand the way it is working.

The following chapters detail all variables, functions and subroutins used and developed in the code. Those details may be useful for a developer. The last chapter details some modifications and improvements that can be done.

The code as detailed in this reference manual has been updated to the optics and beam parameters of the High Luminosity Large Hadron Collider (HL-LHC), although the previous version (`https://gitlab.cern.ch/agorzaws/trainTRAIN`) was developed for LHC. Unless stated otherwise, all the examples that will be provided are based on the nominal parameters detailed in [1].

* * *

Contents

CHAPTER 1

Introduction to TRAIN

In part of the straight sections of the LHC the two beams share a common beam tube. Therefore the bunches cross each other not only at the interaction point, but as well at many places on either side, with a typical transverse separation of 10 times the transverse beam size. These "parasitic" encounters lead to orbit distortions and tune shifts, in addition to higher order effects.

TRAIN solves a multivariate, self-consistent and non-linear fixed point problem in order to find the closed orbit of all bunches in both interacting beams once the Filling Scheme and the beam-beam interactions have been introduced. The Filling Scheme is the irregular filling pattern of bunches along the 3564 "buckets" around the machine that can be occupied. The existence of the Filling Scheme gives place to missing long range interactions (PACMAN) and even missing head-on interactions (super-PACMAN), causing different bunches to receive a different number of coherent beam-beam kicks depending on the number of PACMAN and super-PACMAN encounters that each bunch experiences. Consequently, bunch by bunch differences in orbit, tune and chromaticity, that cannot be corrected, will exist.

TRAIN is formed by a set of bash scripts written by A. Gorzawski in order to ease and grant the correct execution of the Fortran core developed and modified by F.C. Iselin, E. Keil, H. Grote, W. Herr and A. Gorzawski, M.Hostettler and A. Ribes-Metidieri.

* * *

1.1 Fast execution

The recommended method for operating TRAIN contemplates the usage of the bash scripts developed by A. Gorzawski and can be acchieve in three steps.

1. In the first set up, execute the command

   ```
   $ make
   ```

   that will generate the compiled executable file of the Fortran core.

2. Execute

   ```
   $ ./updateMadFiles.sh {OperationConfiguration}
   ```

   where OperationConfiguration refers to a MAD-X file (.mad) containing information about the crossing angles and planes, half parallel separation, the beam definitions, the optics...for generating the Twiss and map MAD-X input files that TRAIN will use. The files involved in creating the twiss and map files can be found in the folder *MAD_PART*. If a new configuration is to be described, the files *collisionPath*, *BeamDefinition* and *LHC_HL_<configuration>.dat* will have to be revised.

3. Execute the command

   ```
   $ ./runTrainWithFillingScheme.sh {fillingSchemeInputFile}{plot|noPlot}
   ```

   with fillingSchemeInputFile the name of the Filling Scheme that will be used in the simulation and the flag plot|noPlot that will trigger the appearance of the plots of the full separation in sigma of both beams at the Interaction Points 1 and 5, as well as the horizontal and vertical orbit as a function of the bunch id at these locations. This command will generate the file of input parameters *setup.input* and will trigger the executable generated in the first step with *setup.input* file. The input parameters can be varied through the usage of the file *input.temp*, used for generating *setup.input* once the information of the filling scheme has been added automatically while generating *setup.input*. The input file with the Filling Scheme information must be located in the folder *FillingScheme* and accepts two possible configurations:

   - *slotNb FilledB1 FilledB2 BCurrB1 BCurrB2*, with the slot number ranging from 1 to 3564, *FilledB1* and *FilledB2* indicating whether if the slot is occupied by a bunch or empty and *BCurrB1* and *BCurrB2* the normalized beam currents to the values of MAD-X.
   - *slotNb FilledB1 FilledB2 BCurrB1 BCurrB2 EmittXB1 EmittXB2 EmittYB1 EmittYB2*,, with the slot number ranging from 1 to 3564, *FilledB1* and *FilledB2* indicating whether if the slot is occupied by a bunch or empty and *BCurrB1, BCurrB2, EmittXB1, EmittXB2, EmittYB1* and *EmittYB2* the beam currents and horizontal and vertical emittances normalized to the values of MAD-X.

If the TRAIN input filling scheme is not available in the *FillingScheme* folder it can be generated using the python code *generateFillinScheme.py* with the description of the filling scheme (.csv file).

TRAIN can also be directly executed without using the bash scripts described above.

1. In the first set up, execute the command

   ```
   $ make
   ```

   in bash, which will generate the executable files from the Fortran code.

2. Execute

   ```
   $ ./amtrain < setup.input
   ```

   which executes TRAIN with the input parameters. The file *setup.input* must be generated before the execution.

This method is not recommended, since it doesn't offer a management of the input and output files generated through the execution of TRAIN.

* * *

1.2 How does TRAIN work?

The code has been updated for the usage of HL-LHC and the improvements and detailed functioning can be found in Train TWiki (`https://twiki.cern.ch/twiki/bin/view/ABPComputing/TrainWikiPage`). The schematic representation of the data flow in TRAIN can be seen in 1.1. In the first step, TRAIN updates the information of the Twiss parameters and the map files obtained with the program MAD-X of a single bunch in beam 1 and in beam 2 along the desired optics selected for HL-LHC (we have used the optic version HLLHC1.3 and the nominal and ultimate operational scenarios detailed in [1], unless specified otherwise). Secondly, TRAIN assembles the Filling Scheme information with the Twiss files in order to simulate the collision of all the bunches of beam 1 with the ones of beam 2, as schematically showed in Fig. 1.2.

Then, the closed orbits of beam 1 and beam 2 without beam-beam interactions are initialized using the first order maps. Next, the program proceeds to the introduction of the beam-beam interaction in the interaction points and a double loop iteration starts, so in the inner loop the closed orbit with

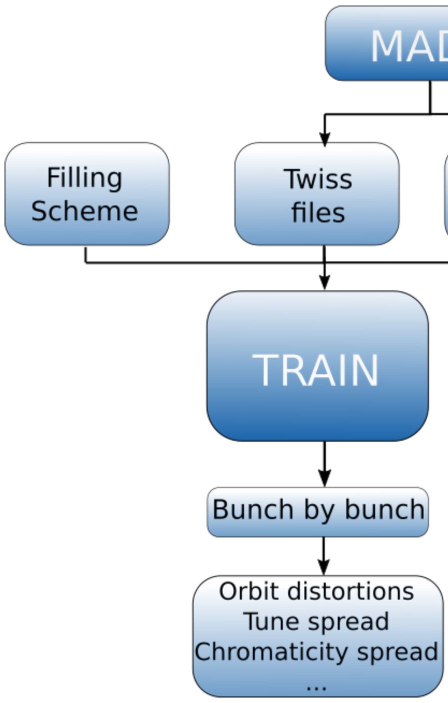

Figure 1.1: Representation of the data flow used in TRAIN. The input Twiss and map input files, jointly with the filling scheme are combined within TRAIN for computing the bunch by bunch orbit, tune and chromaticity among others.

fixed beam-beam kicks are found and in the outer one the bunch positions are updated until the convergence of both closed orbits into a stable solution. The last step is the tracking of every bunch pair with the second order maps in order to find out their tune, chromaticity and dispersion.

Conventions in TRAIN
Beam circulates clockwise in ring-1 and anti-clockwise in ring-2
The start of both machines is either IP3 or IP4
Slots (half-buckets) count clockwise from 0 at IP5
Buckets (filling scheme) count anti-clockwise from 0 at IP5 for ring-1
Buckets (filling scheme) count clockwise from 0 at IP5 for ring-2
Bunches count anti-clockwise from 1 at IP5 for ring-1
Bunches count clockwise from 1 at IP5 for ring-2
Beam-Beam points count clockwise from 1 at leftmost parasitic at IP5

Table 1.2: Conventions used by TRAIN in order to define the buckets, the slots and the interaction points.

* * *

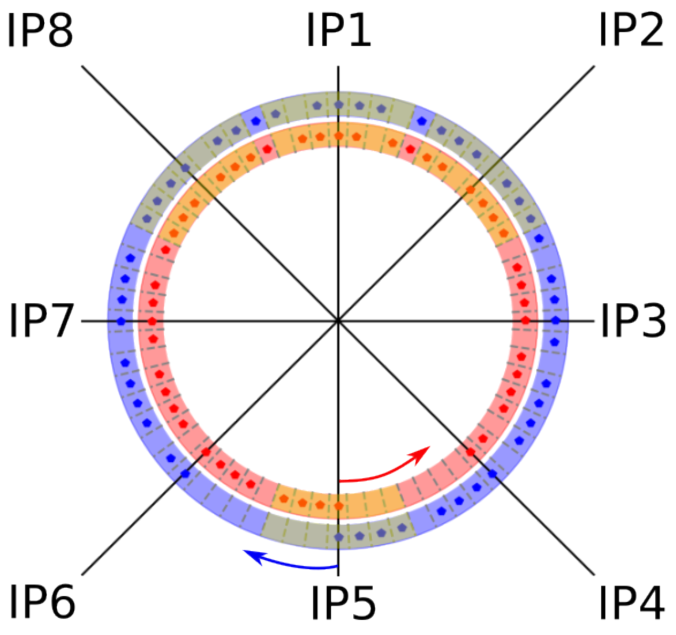

Figure 1.2: Schematic representation of the HL-LHC with the four interaction points 1,2,5 and 8. The buckets are marked by gray lines and the bunches represented by blue and red dots. The information of the Twiss files has been used to build the interaction regions up to D2 around the IPs, marked in yellow. The filling schemes are placed anti-clockwise for beam 1 (blue) and clockwise for beam 2 (red). Both beams circulates in the sense opposing how the filling schemes are placed. The scheme respects the numering conventions of TRAIN 1.2.

1.3 Outputs

Among others, TRAIN outputs results files for:

- Orbit effects:
  - Maximum peak-to-peak global orbit spread and global orbit spread root mean square at the files *fort.fort.ps*.
  - Horizontal and vertical orbit as a function of the bunch id for all the Interaction Points and the selected extra elements that have been introduced (i.e., crab cavities, triplets...) in the files *train.orb*, *hoff* and *voff*, and unperturbed normalized orbit at *train.orb*.

  - Horizontal and vertical separation at the Interaction Points in σ at *hsep_sig* and *vsep_sig* and in μm at *hsep_mu* and *vsep_mu*.

- Horizontal and vertical slop at the Interaction Points at *hslope* and *vslope*.

- Horizontal and vertical dispersion at the Interaction Points at *hdisp, vdisp, hsecdisp* and *vsecdisp*.

- Average luminosity as a function of the bunch id at the Interaction Points at Files *av_lumi* and average per bunch at *lumi*.

- Collision scheme and equivalent classes that have been computed. Files *alt.list, coll.count, freq_f.count, freq_b.count, equ_f.count, equ_b.count, set_f.list* and *set_f.list*.

- List of regular bunches in the file *reg.list*.

- Tunes and chomaticities:

  - Horizontal and vertical tune in the files *tune.list*.
  - Horizontal and vertical chromaticity in the files *tune.list*.
  - Beam current in the files *tune.list*.
  - Maximum peak to peak tune and chromaticity in the file *fort.closest_tune_ap*

- Global results at *train.list*

- Others at the files *fort*.

* * *

1.4 Changes in TRAIN

In the latest update of TRAIN several bugs have been corrected and some improvements have been made. In the following, the changes developed on top of the latest version modified by M. Hostettler are summarized.

	Previous version	Current version
①	• Adapted to LHC layout and optics	• Adapted to HL-LHC layout and optics.
②	• Fixed number of parasitic encounters without taking into account the length of the common chamber up to D1.	• Different number of parasitic interactions in different IR depending on the length of the common chamber up to D2.
③	• Symplecticity issues on the maps.	• Check of the symplecticity of the maps.
④	• Not possible to compute coupling and wrong tunes in the presence of coupling	• Tunes computed taking into account the linear coupling
⑤	• Beam 2 filling scheme overwritten with beam's 1 (bug).	• Independent filling schemes for beam 1 and beam 2.
⑥	• Outputs just possible at interaction regions.	• Possibility of obtaining results on other non interaction region positions, i.e., crab cavities, triplets, collimators...
⑦	• P5 mandatory a beam-beam interaction point. Rigidity of the code.	• Mandatory at least one beam-beam interaction point. Allows to disentangle the effect on different interaction points.
⑧	• Numerical divergence in the calculation of the second order beam-beam maps for round beams. Effect on the chromaticity.	• Solved bug imposing a tolerance in the calculation of the second order beam-beam maps for round beams.

Table 1.4: Summary of the changes developed in TRAIN in the latest version, built on top of the modified version of TRAIN developed by M. Hostettler.

① Modification of the MAD-X files. The layout, optics, beam parameters and number of interaction points have been changed in the files *LHC_HL_*.

② The number of parasitic encounters, defined in the file *npar.hllhc* has been changed according to the length of the common chamber up to the separation dipole D2 for the different interaction points.

③ The subroutine *symcheck* (Sec. 5.45) has been created and the symplecticity of the maps is checked on the subroutine *mketunemad2* (Sec.

4.15) before block diagonalizing the transfer maps.

4. The subroutine *mketunemad2* (Sec. 4.15) allows to compute the tunes in the presence of coupling using the block diagonalized one turn map as well as the coupling using the minimum tune approach (subroutine *Lnrcoup* in Sec. 5.43).

5. The code has been changed for accepting two different filling schemes for beam 1 and 2. The main modifications have been done in *collsch1* (Sec. 4.3) while reading the filling schemes and in *prcoll* (Sec. 4.10) in order to take into account which bunches of beam 1 are interacting with those of beam 2 if both beams have different filling scheme.

6. In order to enable the computation of the orbit in a location where the interactions are not taking place we have modified several parts. First of all, we have modified the MAD-X files, so maps at locations different from the beam-beam interaction points can be added. The introduction of those elements, for instance, the crab cavities and the triplets, can be controlled through the knob *on_ext* that is called in the file *commonBeamBeamPart.mad* and generates the map files *train.mapf* and *train.mapb* containing not only the maps in the beam-beam interaction points but also where these new elements are. Secondly, the subroutines responsible of the reading of the maps as well as those involved in the tracking have been modified to compute the one-turn map concatenating all maps but introducing the beam-beam maps in theose locations where beam-beam interactions take place.

7. The code has been modified so any interaction point is accepted independently of the presence of IP5. In order to correctly combine the information coming from the files *train.optf* and *train.optb* containing the information of the orbit, the dispersion and the β-function in the locations where the beam-beam interactions take place with the files *train.mapf* and *train. mapb* it is necessary that the files have a common origin. This common origin was achieved in the subroutine *getmask* rotating the optic data so the origin was at the leftmost beam-beam interaction point of IP5, since the first maps appearing in the map files are those of IP5 when IP5 is present independently of the presence of the other IPs. In *getmask2* (Sec. 5.3) the rotation is done to the leftmost beam-beam interaction of the first IP appearing in the map files, so it is not necessary to impose the presence of IP5 any more.

8 In some configurations, for instance, during the nominal collision of HL-LHC, we observed that the chromaticity spread of some bunches increased nonphysically. We also observed a big impact of the feed-down effect in this configuration while in similar ones, such as the ultimate collision, the impact of the occtupole feed-down was negligible. This nonphysical behavior came from the fact that a single beam-beam second order map diverged due to the term $\sqrt{2|\sigma_x - \sigma_y|}$ that divides some of the elements of the map. This divergence was a consequence of the usage of the non limit formula for round beams due to the restrictive condition imposed to apply this limit, i.e., the exact equality $\sigma_x = \sigma_y$ missed some cases of round beams. Although the numerical issue has been solved, the tolerance that has been imposed is arbitrary, so it may be necessary to come back to modify it. In case that a non physical chromaticity is obtained, the origin may be in the subroutine *Trbb* (Sec. 5.15) due to the condition of round beams, i.e., we must apply a limit formula in the case of round beams $\sigma_x = \sigma_y$ in order to avoid numerical divergences in the second order beam-beam maps.

CHAPTER 2

INPUT FILES

In this chapter, the main features of the input files required for TRAIN are described. As explained in chapter 1, through the bash execution of TRAIN the input files are automatically generated, and the parameters that should be changed depending on the desired configuration are explained here.

2.1 MAD-X input files

TRAIN uses as input files the information of the position and optic functions present in the Twiss files *train.optf* for the forward beam and *train.optb* for the backwards one; the position of the ideal orbit using the survey files *train.surf* and *train.surb* and the information of the first and second order sector maps present in the map sector files *train.manf* and *train.manb*. In order to generate these files, it is necessary to execute a main program *.mad* with the latest version of MAD-X (bug in the sectormaps fixed by M. Hostettler). In the main program the following information must be specified:

- LHC and HL-LHC sequences (files *lhc.seq* and *hllhc_sequence.madx*).
- Optic layout of the accelerator depending on the configuration that it is wished to run. Our simulations are based on the detailed parameters and optics (version HLLHC1.3) found in [1].
- Beam definition, specifying the energy of the beam, the number of particles per bunch, the emittance, the rms bunch length and the rms energy spread (q-Gaussian).
- If the octupoles are going to be activated (knob *on_oct*).
- The targeted tunes and chromaticities.

- The half crossing angles in all interaction points in μrad.
- The tilting in the transverse plane of the interaction points (we have used the knobs on_phi to analyze the effect of the transverse tilting on the linear coupling).
- The half parallel separation in all interaction points.
- Call to the file *commonBeamBeamPart.mad* that introduces the markers at the head-on and long-range interaction points and generates the input files of TRAIN.
- Select in the file *on_ext.hllhc* if extra elements have to be introduced in the maps (called in *commonBeamBeamPart.mad*).
- The active pits that are desired in the simulation can be changed in the file *collisionPath* (called from *commonBeamBeamPart.mad*) and allow to select if head-on and long-range interactions are going to be active for the different IPs.

These tables contain the information of a single bunch in beam 1 and beam 2 without beam-beam interactions. Further information about MAD-X can be found in [5].

* * *

2.2 *setup.input*

The information that must be specified in the input file *setup.input* is summarized in the following.

- Filling Scheme input file
- The write or execute flags
  - write: Indicates if all output files need to be generated.
  - coll: If True, the file *coll.count* is generated.
  - frequ: If true, the files *freq_f.count* and *freq_b.count* are generated.
  - equ: If true, the files *equ_f.count* and *equ_b.count* are generated.
  - set: If true, the files *set_f.list* and *set_b.list* are generated.
  - alt files: If true, the files *alt.list* and *reg.list* are generated.

- orbit: If true, introduces the tunes and recomputes the closed orbits.
- 2nd-order: If true, second order terms are introduced in the tracking.
- w_detail: If true, the files *equ_f.count* and *equ_b.count* are generated.
- b_curr: Base beam current. Depending on the input variable the beam currents are computed in different ways:
  * *beamc_f* = 1: It computes two alternate beam currents, if the bunch is even it computes the current as $beam\_current(n) = sigb \times bcurr$, whereas if it is odd, $beam\_current(n) = bcurr$, where *sigb* is the sigma of bunch current and $bcurr$ is the current per bunc. Both are input parameters.
  * *beamc_f* = 2: It uses the function *bc_fun* for beam currents, i.e., $beam\_current(n) = bcurr \times bc\_fun(sigb)$
  * *beamc_f* = 3: computes the beam current from the values read from a file, i.e., $beam\_current(n) = bcurr \times bc(n)$, where *bc* is read in subroutine *collsch1* from unit mucoll = 24 (for instance, file FillingSchemes/train25_5108_scan1.in).
- emitt: Emittances. Depending on the input parameter, the emittances are computes in different ways:
  * *emitt_f* = 1: Two alternate beam currents. It computes $\epsilon(n) = sigem \times \epsilon_0$ for the odd bunches and $\epsilon(n) = \epsilon_0$ for the even ones.
  * *emitt_f* = 2: Use the function *bc_fun* for the beam currents, i.e., $\epsilon(n) = \epsilon_0 \times bc\_fun(sigem)$
  * *emitt_f* = 3: The first half gets a factor $\epsilon(n) = sigem \times \epsilon_0$, while the second half does not.
  * *emitt_f* = 8: The emittance is computed from the values read in *collsch1* as $\epsilon = \epsilon_0 \times \epsilon_{b1}$ for x and y coordinates and both beams.
  * otherwise: Sets random values for the emittance choosing a positive value of $\epsilon_0(1 + sigem \times rand)$

• Other input parameters

- full_coll: Full collision flag. If True, all bunches are taken.
- nturn: Number of turns.

  - Debug: Debugging flag.
  - outbcnt: Number of out bunches. If *outbcnt* is 0, all the elements of *outblist* are 0.
  - outpos: Out position in half buckets.
  - outnorm: If 0, writes the orbits of *bunch1* and *bunch2* as a function of the turn. Otherwise, it writes the normalized x and y orbit coordinates and the square root of the radius in phase space of the out bunches.
  - xifact: Non head on collision factor.
  - hofact: Head on collision factor.
  - amp_bunch: Adds amplitudes to the bunches during the tracking. If zero, all bunches are added random amplitudes.
  - amp_fac: Start amplitude in sigmas (x + y); amp_fac=0: t_gauss, else value.
  - lumi_hist: Boolean. If true, generates the file *hist.list*.
  - b2_off: The offset of beam 2 with respect to beam 1 in half buckets.

- List of output bunches.
- Name of the orbit output file.
- Name of the general output file.
- Number of bunches for which the tune is going to be analyzed.
- Horizontal and vertical start amplitudes in σ.
- Beam current beam size σ_b and emittance size σ_e.
- Random seed. Used in the generator functions for the beam currents or the emittaces if the option of random intensities is selected.
- Extra elements flag. Allows to specify if there are maps corresponding to elements different from the interaction points.

All these parameters can be modified for customizing the properties of the beam. For instance, it would be possible to define two beams with bunches of random intensities or analyze the output in a given location of a list of 10 bunches.

In order to obtain the desired input file *setup.input*, the file *setup.temp* must be changed with the desired specifications and through the scrips execution specified in the subsection 1.1, the file *setup.input* is automatically generated.

* * *

2.3 Filling Scheme

TRAIN supports two file types of Filling Schemes depending on the parameters specified on *setup.input*, in particular on the value of *emitt*.

- *emitt_f* = 8: The values of the horizontal and vertical emittances are read from the Filling Scheme file. In this case, the input file for the Filling Scheme is assumed to have nine columns and the same number of rows as of buckets in the accelerator. The columns correspond to slot number, a 0 or a 1 indicating if that slot is filled for beam 1 and 2 respectively, the beam currents factor for beam 1 and 2; and the horizontal and vertical emittances for beams 1 and 2.

- *emitt_f* ≠ 8: The emittances are not read from the Filling Scheme input file, that is expected to have five columns indicating the slot number, if beam 1 and 2 are filled and the beam current factor for beam 1 and 2.

Several Filling Schemes files have been generated with this format. However, if the desired file is not generated it can be obtained using the auxiliar code *generateFillingScheme.py* and the csv file containing the description of the injection scheme.

CHAPTER 3

VARIABLES

The variables and parameters that are used in the main code of TRAIN or that are passed through common block structures are summarized here. For each variable, the type and dimension of the array are specified, jointly with a brief description. Local variables within subroutines have not been considered, since the inputs and outputs of Subroutines and functions are specified in sections 4,5 and 6.

The variables have been classified in 11 categories in order to ease the understanding of their usage as well as to finding them in these tables in a simplier way.

* * *

3.1 Parameters

General description. Variables that are not changed along the program. Used for defining array lengths among others.	
mbuck = 3564	*Integer*. Total number of buckets. The length of the ring is divided in small units with the equivalent distance that a bunch can travel in 25 ns. These units are called buckets.
mbunch = 3000	*Integer*.Maximum number of bunches in the accelerator.
mdslt = 2 × mbuck	*Integer*.Twice the number of buckets in the accelerator,i.e., interaction points (since interactions can occur every half bucket).
mpar = 50	*Integer*. Maximum number of one side parasitic interactions.

mpit = 4	*Integer.* Maximum number of active pits, i.e., points where the head-on interactions take place.
mcol = 2 * mpar * mpit + mpit	*Integer.* Maximum number of both head on and parasitic collisions.
melm = 10	*Integer.* maximum number of extra elements per one side interaction point.
mmaps = mcol + 2*mpit*melm	*Integer.* Maximum number of sector maps.
mlocal = 2 * mpar + 1	*Integer.* Maximum number of collisions per pit.
mdim = 4	*Integer.* Maximum number of phase space dimensions.
mvary = mdim $\times$ **mbunch** $\times$ **2**	*Integer.* Maximum number of variables, (used in mxmpy) with mvary = 6 (used in subroutine eigen).
max_list = 10	*Integer.* Maximum number of observed bunches during tracking.
erad = 2.8×10^{-15} **m**	*Float.* Classical electron radius.
emass = 0.51×10^{-3} **GeV**	*Float.* Rest mass of the electron
prad = 1.5×10^{-18} **m**	*Float.* Classical radius of the proton.
pmass = 0.938 **GeV**	*Float.* Rest mass of the proton.
ech = 1.6×10^{-19} **C**	*Float.* Charge of the electron.
Arrays size	*Integer.* maxseq = 20000, mcnam = 16 and msect = 259.
Unit parameters	*Integer.* iunit=11, orbout = 22, mulist = 23, mucoll = 24, +msep = 25, lumilist=26,mtrack = 30 and ustart = 50.

* * *

3.2 Input variables

General description. Input variables read at the subroutine *dialog2.*	
filename	*String.* Input Filling Scheme file name.
b2_off	*Integer.* The offset of beam 2 with respect to beam 1 in half buckets.
iseed	*Float.* Random seed for random numbers generators. It should be between 10^5 and 10^{10}.

amp_bunch	*Integer.* Adds amplitudes to the bunches during the tracking. If zero, all bunches are added random amplitudes.
amp_fac	*Integer(4).* Horizontal and vertical start amplitudes in σ.
ampx,ampy	*Float(2).* Start amplitude in sigmas x(1,2) + y(1,2).
sigb	*Float.* Beam current beam size σ_b.
sigem	*Float. Float.* Emittance size σ_{em}.
xifact	*Float.* Long-range collision factor.
hofact	*Float.* Head-on collision factor.
beamc_f	*Integer.* Options for computing the beam currents.
emitt_f	*Integer.* Options for computing the emittances.
nturns	*Integer.* Number of turns.
debug	*Integer.* Debugging flag, debugging mode if debug > 0.
outbcnt	*Integer.* Number of out bunches. If *outbcnt* is 0, all the elements of *outblist* are 0.
outblist	*Integer(10).* List of output bunches. Maximum of 10 elements.
outpos	*Integer.* Out position in half buckets.
outnorm	If 0, writes the orbits of *bunch1* and *bunch2* as a function of the turn. Otherwise, it writes the normalized x and y orbit coordinates and the square root of the radius in phase space of the out bunches.
c_tunes	*Integer.* Number of bunches for which the tune is going to be analyzed.
seed	*Float.* Random seed used in the generator functions for the beam currents or the emittances if the option of random intensities is selected.

Input flags

General description. Input variables read at the subroutine *dialog2.*	
all_write	*Boolean.* Indicates if all output files need to be generated.
w_coll	*Boolean.* If True, the file *coll.count* is generated.

w_equ	*Boolean.* If true, the files *equ_f.count* and *equ_b.count*, containing information about the equivalent classes, are generated.
w_frequ	*Boolean.* If true, the files *freq_f.count* and *freq_b.count* are generated.
w_set	*Boolean.* If true, the files *set_f.list* and *set_b.list*, with the bunch ids for each beam, are generated.
w_alt	*Boolean.*
w_alt	*Boolean.* If true, the files *alt.list* and *reg.list* are generated.
c_orbit	*Boolean.* If true, introduces the tunes and recomputes the closed orbits.
f_second	*Boolean.* If true, second order terms are introduced in the tracking.
w_detail	*Boolean.* If true, the files *equ_f.count* and *equ_b.count* are generated.
f_coll	*Boolean.* Full collision flag. If True, all bunches are taken.
lumi_hist	*Boolean.* If true, generates the file *hist.list.*

* * *

3.3 Global variables

General description. Variables passed through the common blocs *glob.* Provide general information about the characteristics of the beam and interaction regions.

npit	*Integer.* Number of active pits.
nbunch	*Integer.* Total number of bunches.
ninter	*Integer.* Total number of long-range and head-on active interaction points.
c_turn	*Integer.* Current turn.
npar	*Integer.* Number of one side parasitic interactions.
n_parasit	*Integer.* Index of the parasitic interaction.
iact	*Integer(8).* Indicates the number of active pits with a 1, while 0 refers to non activated pits.
epsx0	*Float.* Horizontal base emittance.

epsy0	*Float.* Vertical base emittance.
gev	*Float.* Energy of the beam (GeV).
bcurr	*Float.* Base beam current.
gamma	*Float.* Relativistic factor.
frev	*Float.* Synchrotron frequency.
circum	*Float.* Accelerator circumference (m).
tmass	*Float.* Rest mass of the particles the beam is formed of.
tradius	*Float.* Classical electron or proton radius.
xisign	*Float.* Sign of the beam-beam force depending on the types of interacting particles.
root2	*Float.* Square root of 2.
lumicnt	*Float.* Number of bunches contributing to luminosity.
lumiav	*Float.* Average luminosity.
lumifact	*Float.* Luminosity factor.
ippos	*Float(8).* Position (in m) of all interaction points with respect to IP3.
para_names	*String.* List of markers corresponding to parasitic interactions.
time1	*Float.* Not initialized.
time2	*Float.* Not initialized.
code	*Integer(mbunch).* External code for bunches.
ipit	*Integer(mpit).* List with the number of the active pits.
actlist	*Integer(mpit).* Sorted number of all pits in order of appearance in the map files.
nmaps	*Integer.* Total number of sector maps at the map files.
mapmask	*Integer(mmaps).* Mask indicating the position of the elements on the map list of elements that does not appear on the Twiss tables.
mname	*Integer(mmaps).* List of names of the elements present on the map files.

* * *

3.4 Optic functions

General description. Variables read from the Twiss and survey tables. Passed through the common blocks *opt*.	
particle	*String(2)*. Type of particle forming the beam.
seq_name	*String(2)*. Name of the used sequence.
name	*String*. Name of element in the Twiss tables.
betx	*Float(mcol,2)*. Horizontal beta function.
bety	*Float(mcol,2)*. Vertical beta function.
delta	*Float*. Difference between the reference momentum and the design momentum, divided by the design momentum.
deltap	*Float*. The relative energy spread (σ_E/E). The marker used in MAD-X for this variable is *sige*, not *deltap*.
dx	*Float(mcol,2)*. Horizontal first order dispersion.
dy	*Float(mcol,2)*. Vertical first order dispersion.
s	*Float(mcol,2)*. Arc length along the reference orbit (m) with respect to IP3.
epsx	*Float(mcol,2)*. Horizontal emittance.
epsy	*Float(mcol,2)*. Vertical emittance.
x	*Float(mcol,2)*. Horizontal position of the closed orbit, referred to the ideal orbit (m).
xmu	*Float(mcol,2)*. Horizontal phase advance.
y	*Float(mcol,2)*. Vertical position of the closed orbit, referred to the ideal orbit (m).
ymu	*Float(mcol,2)*. Vertical phase advance.
eiv1/eiv2	*Float(6,6,max_list)*. One turn map for out bunches.
orb0_1/orb0_2	*Float(6,max_list)*. Orbit for out bunches.
alfx	*Float(mcol,2)*. Horizontal α-function.
alfy	*Float(mcol,2)*. Vertical α-function.
survey_x	*Float(mcol,2)*. Horizontal reference orbit (m).
survey_y	*Float(mcol,2)*. Vertical reference orbit (m).
survey_z	*Float(mcol,2)*. Longitudinal reference orbit (m).
survey_x_ip	*Float(9,2)*. Horizontal reference orbit (m) at the head-on interaction points.
survey_y_ip	*Float(9,2)*. Vertical reference orbit (m) at the head-on interaction points.
survey_z_ip	*Float(9,2)*. Longitudinal reference orbit (m) at the head-on interaction points.
survey_sep_x	*Float(mcol)*. Horizontal survey separation.
survey_sep_y	*Float(mcol)*. Vertical survey separation.

collsk	*Integer(2,mbuck)*. Filling Scheme for beams 1 and 2. Indicates with a 1 the slots that are full.

* * *

3.5 Sector maps

General description. Variables obtained from the input map files.

sk1/sk2	*Float(6,mmaps+1)*. Kick for every interaction point and beams 1 or 2 respectively.
sr1/sr2	*Float(6,6,mmaps+1)*. First order map of dimension 6×6 for every interaction point and for beam 1 and 2 respectively.
st1/st2	Second order map of dimension $6 \times 6 \times 6$ for every interaction point and for beam 1 and 2 respectively.

* * *

3.6 One turn maps

General description. Information of the one-turn map. Passed through the common block *turn*.

tr1/tr2	*Float(6,6,mbunch)*. First order one-turn map.
tt1/tt2	*Float(6,6,mbunch)*. Second order one-turn map.

* * *

3.7 Orbit functions

General description. Information about the orbit. passed through the common blocks *corbit* and *mtcomm*.

z1/z2	*Float(6,mbunch,mmaps+2)*. Closed orbit 6 dimensional vector for each bunch and interaction point.
z1a/z2a	*Float(6,mbunch,mmaps)*. Closed orbit 6 dimensional vector for each bunch and interaction point after collision.
z1b/z2b	*Float(6,mbunch,mmaps)*. Closed orbit 6 dimensional vector for each bunch and interaction point before collision.
dd1/ dd2	*Float(6,mbunch,mmaps+2)*. Second order dispersion for each bunch and interaction point.
d1/d2	*Float(6,mbunch,mmaps+2)*. First order dispersion for each bunch and interaction point.
ztr	*Float(6,mbunch,2)*. Trajectory for each bunch and beam.
orb_amp	*Real(2,mbunch,2)*. Horizontal and vertical orbit amplitude for beam 1 and 2.
phix0	*Float*. Horizontal kick without change of orbit.
phix	*Float*. Horizontal kick once the orbit change has been considered.
phiy0	*Float*. Vertical kick without change of orbit.
phiy	*Float*. Vertical kick once the orbit change has been considered.

* * *

3.8 Bunch variables

General description. Information per bunch. Passed through the common block *bunchf*.	
bcurr1 /bcurr2	*Float(mbunch)*. Bunch current.
qx1 / qx2	*Float(mbunch)*. Horizontal tune.
qy1 /qy2	*Float(mbunch)*. Vertical tune.
qxp1/qxp2	*Float(mbunch)*. Horizontal chromaticity.
qyp1/qyp2	*Float(mbunch)*. Vertical chromaticity.
bc1/bc2	*Float(mbunch)*. Input bunch current for beam 1 and 2 respectively.

epsxb1/epsxb2	*Float(mbunch)*. Horizontal emittance of the bunches of beam 1 and 2 respectively.
epsyb1/epsyb2	*Float(mbunch)*. Vertical emittance of the bunches of beam 1 and 2 respectively.

* * *

3.9 Collision schedules

General description. Variables referring to the collision schedule built in *prcoll*. Passed through the common block *sched*.	
ibcnt1/ibcnt2	*Integer(mbunch,mcol)*. Bunch in beam 2/1 colliding with a given bunch from beam 1/2 at a given collision point.
maskm	*Integer(mdslt+1)*. For collision point i, $maskm(i)$ is the slot number.
present	*Integer(mbunch,2)*. Indicates with a 1 if a given bunch of beam 1 or 2 is still present.
maskmi	*Integer(mdslt+1)*. For slot i, $maskmi(i)$ is 0 or the number of the collision point.
maskmp	*Integer(mdslt+1)*. For slot i, $maskmp(i)$ is the number of previous or current collision points.
maskmn	*Integer(mdslt+1)*. For slot i, $maskmn(i)$ is the number of next or current collision points.
ibnch1/ibnch2	*Integer(mdslt+1)*. For collision point i, $maskm(i)$ is the slot number.

* * *

3.10 Equivalent classes

General description. Variables related to the equivalent classes built in *equ_class*.

hitlist_f/hitlist_b	*Integer(mbuck).* Bunch slot mask for all equivalent class bunches of beam 1/2.
colcnt_f/colcnt_b	*Integer(mbuck).* Number of collisions of each bunch of beam 2/1 with the bunches of beam 1/2.
list_f/list_b	*Integer(mcol,mbuck).* The collision point numbers as a function of the number of the collision and the bunch number.
part_f/part_b	*Integer(mcol,mbuck).* Stores the number of the bunch of beam 2/1 as a function of the collision number of the given bunch of beam 1/2 and the number of the bunch of beam 1/2 with which it is colliding.
cequl_f/cequl_b	*Integer(mcol).* Number of collisions of each bunch of one beam with the bunches of the other beam.
nequl_f/nequl_b	*Integer(mcol).* The number of buckets within the class *i* that have a different number of collisions which have collided in one point.
lequl_f/lequl_b	Id of the class of the buckets that have collided at the same point.
cordl_f/cordl_b	*Integer(mcol).* Number of collisions of a given class.
nordl_f/nordl_b	*Integer(mcol).* Number of buckets belonging to a given class.
lordl_f/lordl_b	*Integer(mcol,mbuck).* Number of the bucket as a function of the buckets where collisions occur and the id of the class.
ntotal_f/ntotal_b	*Integer.* Total number of bunches on the forward/backward beam.
ctotal_f/ctotal_b	*Integer.* Total number of equivalent classes.
nset_f/nset_b	*Integer.* Total number of elements in *set.*
equl_f/equl_b	*Integer.* Number of ordered collision lists.
ordl_f/ordl_b	*Integer.* Number of ordered collision lists
set_f/set_b	*Integer(mbuck).* Bunch id.
colpnt	*Integer.* Total number of collision points.
tcount	*Integer(mbuck,2).* Interaction point index for each bunch and beam.

3.11 Unused variables

General description. Defined and initialized variables that are passed using the common blocks but that are never used in the program flow.	
bcfile	*Boolean*. Unused.
title	*String*. Unused.
type	*String*. Unused.
date	*String*. Unused.
hour	*String*. Unused.
timew	*String*. Unused.
nlocal	*Integer*. Number of one side parasitic interactions. Unused.
arad	*Float*. Unused.
partno	*Float*. Unused.
q11	*Float(mbunch)*. Unused.
q21	*Float(mbunch)*. Unused.
q12	*Float(mbunch)*. Unused.
q22	*Float(mbunch)*. Unused.
occur	*Integer(mcol,2)*. Number of occurrences. Array of 1's never used.

* * *

CHAPTER 4

MAIN PROGRAM

The subroutines used in the main Fortran program are briefly described as they show up sequentially, and the functions and subroutines that each one calls are also listed in order of appearance in chapters Subroutines and Functions.

Some variables are initialized in *hajimeru* and all the data asked to the user is read in *dialog*.

The collision schedule is read and stored in *collsch1*. The optic files for beam 1 and beam 2 are read. It also checks that the maximum number of interaction points *nint* is smaller than the maximum number of collisions and that the number of interaction points match for both beams. It also identifies the head-on collision points, in the points referred as 'pits' and saves the name, the number, the s position and either if the pit is active or not (a pit is considered active if there are collisions in it).

In the next step, it tries to associate the parasitic interactions (long-range interactions) to the different pits and count the number of interaction in each pit. Initializes and reads the map files (the s position of the optic elements, the kick vector k and the first r and second order t transfer maps).

The collisions are simulated and the nominal, PACMAN and super-PACMAN bunches are found. The currents are set and the initial guess of the closed orbits is found. The beam-beam maps are introduced and the closed orbit of all bunches and both beams are found imposing that the orbits converge to a stable solution.

Finally, all bunch pairs are tracked using also the second order maps in order to find the tune, chromaticity and dispersion.

* * *

4.1 Hajimeru

General description. Initializes several variables and arrays.	
Detailed description	Initializes *lumicnt,lumiav* and *n_parasit*. Sets to 0 *iact* and to -1000. *ippos*, two arrays of length 8. Generates files *fort.* and enumerates the fourth column of the files from 0 to *mbunch*. The definition of the mentioned variables is detailed in chapter Variables.

* * *

4.2 Dialog2

General description. Reads the input file *setup.input* and store the input parameters in variables.	
Detailed description	Reads the input file *setup.input* and stores the non-commented lines of the input file in different variables. As specified in chapter Input files, the information contained in the file *setup.input* specifies the Filling Scheme that it is going to be used, the flags *all_write*, *w_coll*, *w_equ*, *w_set*, *w_alt*, *c_orbit,f_second* and *w_detail* detailing which output files are going to be generated, the option that will be used for computing the beam currents and the emittances, the number of turns and of bunches that are going to be tracked at a certain position, the head-on and long-range factor, the bunch amplitude and factor, the beam offset, the number of bunches for which the coherent tune will be computed, the amplitude of σ_x and σ_y, the random seed and the flag *extraelem*, specifying if the maps are going to be given in other positions of the accelerator different from the interaction regions.
Dependencies	Uses the function *nxline*.

* * *

4.3 Collsch1

General description. Reads and stores the Filling Schemes of beam 1 and beam 2.	
Detailed description	Reads one file located at the folder *FillingScheme* of the specified format in chapter Input files containing the collision schedule or Filling Scheme of both beam 1 and beam 2. If the input parameter *emitt_f* read in MAIN program has a value of 8, the input Filling Scheme is expected to be a file of 9 columns, where the beam currents and the emmitances are specified.Writes the numeration of the bunches as well as the filling scheme of beam 1 in file *fort.77*. If *emitt_f* $\neq 8$, initializes the horizontal and vertical emittance of both beams to 1. Otherwise, reads the emittance as input parameters.
Input	*Unit*: Input unit where the filling scheme has been opened.

* * *

4.4 Rdoptc

General description. Reads the Twiss files generated with MAD-X.

Detailed description	Reads the twiss tables in the files *train.optf* for beam 1 and *train.optb* for beam 2 and fills the arrays corresponding to the name of the element *name*, the distance from the start of the sequence *s* (in IP3 or IP4), the horizontal orbit offset in mm *x*, the horizontal β-function *betx*, the horizontal α *alfx*, the horizontal dispersion *dx*, the vertical orbit offset in mm *y*, the vertical β-function *bety*, the vertical α *alfy* and the vertical dispersion *dy*. It also reads the header of the Twiss table using the subroutine *read_opth* and counts the total number of interaction points that will be available. The position of the four interaction points is saved in *ippos*. It also generates the output file *train.list* with the obtained parameters. Finally, the type of particle of each beam is identified as a proton, an electron a positron or an ion. Both beams are not allowed to be formed by different types of particles.
Input	*nbeam*: Number of the beam (1 or 2). *nint*: Number of interactions.
Dependencies	Uses the functions *lastnb, parasit, headon,get_tockens* and *nampos*. Uses the subroutines *read_opth* and *get_cpos*.

* * *

4.5 Mkpits2

General description. Identifies the active interaction points.	
Detailed description	Generates an indexed table with the head-on collisions present in the list *name*, the number of the collision point where the interaction takes place *ipit*, the number of the IP where the collision takes place *pitnam* and the position s of the pit *si*. It also sets to 1 the positions of the interaction points that are active in *iact* and saves the number of the interaction point where the head-on collision is.

Dependencies	Uses the function *headon*.

* * *

4.6 Assoc

General description. Associates the long-range collision points to the head-on collision points (pits).

Detailed description	Selects the parasitic encounters and finds out the interaction point to which the parasitic interaction is associated to by requiring that the distance between the long-range collision point and the head-on collision point (pit) is smaller than 250 m. Saves the total number of long-range interactions associated to each pit i *ncoll* and computes the total number of one side long-range interactions per pit, that are saved in the array *npar* (length eight).
Dependencies	Uses the function *headon* and *parasit*.

* * *

4.7 Rdsurvey

General description. Reads the survey files also generated using MAD-X.

Detailed description	Read the input survey files and stores the survey x, y and coordinates of the four interaction points in *sx*, *sy* and *sz*.
Input	*nbeam*: Number of the beam (1 or 2). *nint*: Number of interactions.
Dependencies	Uses the functions *nampos*, *lastnb*, *get_tockens*, *headon* and *parasit*.

* * *

4.8 Calcsurvey

General description. Computes the separation of the long-range interaction points with respect to the interaction point.	
Detailed description	Computes the angle between interaction point and the long range interaction, as well as the separation of the long range interaction to the interaction point.

* * *

4.9 Rdmaps2

General description. Fill the tables for the transfer maps generated by MAD-X.	
Detailed description	Reads the MAD-X input files *'train.manf'* and *'train.manb'* for the forward and backward beam respectively, in opposite sense (from top to bottom for beam 1 and from bottom to top for beam 2). If sector maps of the files connect interaction points (extra elements haven't been introduced in the maps) saves the position of the element *ss*, the kick vector *sk*, the first order transfer map *sr* connecting two consecutive interaction points and the second order map *st* connecting two consecutive interaction points for both beam 1 and 2. If there are extra elements in the maps, more sector maps than interaction points, the total number of maps is saved. A mask is created marking with a 0 the position of the extra maps that don't correspond to interaction points.
Dependencies	Uses the function *lastnb*.

* * *

4.10 Prcoll

General description. Simulates the collision and registers collision scheme, i.e., which bunches of beam 2 interact with which ones of beam 1 at each interaction point.	
Detailed description	Counts the total number of bunches in both the forward and backward beams, and the total number of collision points. Simulates the collision of bunches of beam 1 with those of beam 2 dividing the ring in half buckets or interaction points, so for every bunch of beam 1, saves all the coordinates of all bunches of beam 2 that interact with that bunch of beam 1. Counts the number of bunches with different forward and backward counts and finds the equivalent classes for both the forward and the backward beams. Computes the total number of interactions and the total number of equivalent classes of both beams. If the input parameter *f_coll* is true (all bunches are taken), the arrays *hitlist* are filled with the injection scheme. Otherwise, gets the *hitlists* using the subroutine *get_hits.* Saves the bucket number where the interactions happen for both the forward and the backward beam. For every bucket in which an interaction point occur for beam 1, we look if there is in *list_f* a bunch that interacts *cp* times. In this case, the bunch of beam 2 that interact with this bunch in the interaction point is seeked. Fills the arrays *maskmp* and *maskmn,* which are the number of the previous or current and next or current collision points.Finds the super pacman bunches and saves the data in files *freq_f.count, coll.count,freq_b.count, equ_f.count, equ_b.count, set_f.list,set_b.count, alt.list* and *reg.list.*
Dependencies	Uses the function *mylist.* Uses the subroutins *getmask2,collide, equ_class, get_hits,prsup* and *prcdmp.*

* * *

4.11 Set_cuem

General description. Sets the beam current and the emittances for all bunches.	
Detailed description	initializes the array of bunches that are present in beam 1 and 2. If the input parameter *beamc_f* =3, the user defined beam currents are used. Depending on the value of *beamc_f* and of the input parameter *emitt_f*, the beam currents and the emittance are computed in different ways. All possible options depending on the input values of the variables *beamc_f* and *emitt_f* are detailed on chapter Input files. The base beam current is read from the header of the input Twiss files in MAIN program and Subroutines. Writes the index of the bunch, the beam currents of both beams and the emittance in file *fort.93*.

* * *

4.12 Orbit02

General description. Finds the closed orbits of all bunches and both beams without beam-beam interactions.

Detailed description	Initializes all six coordinates in phase space of both beams for the first bunch and the interaction point 0 for beam 1 and the interaction point $m = ninter + 1$ for beam 2. Seeks an initial guess of the closed orbit for both beam 1 and beam 2 assuming that there is no dispersion, without taking into consideration the second order maps and without beam-beam interactions. In order to do so, obtains the one turn map for each beam and iterates until the solution for the closed orbit is found solving $\bar{x}_{i,j,k+1} = M_i(\bar{x}_{i,j,k})$, where M_i is the one turn map for beam 1 and 2 and $\bar{x}_{i,j,k}$ stands for the 6 coordinates of beam i, bunch j in turn k. In TRAIN, the one-turn first order maps are saved in the variables *tr*. Since we are not considering beam-beam interactions, the closed orbit is just found for the first bunch of both beams.
Dependencies	Uses the subroutines *track0* and *solver*.

* * *

4.13 Orbw

General description. Writes in the output file *train.orb* the horizontal and vertical normalized distance of the unperturbed orbits.

Detailed description	Writes in the output file *train.orb* the horizontal and vertical normalized distance between bunches in all the interaction points for the first bunch of both beams. The orbits of both beams have been computed without beam-beam interactions.
Dependencies	Uses the functions *sigx* and *sigy*.

* * *

4.14 Eicoll

General description. Finds the eigenvectors of the one-turn map at the observation point for the output bunches.

Detailed description	If the numjber of output bunches is different from 0, computes the one turn map at the position where the output bunches are going to be observed and the orbit of the list of ouput bunches in the output position. In order to do so, it tracks through the sector *outpos* to the end and and from sectors 0 to *outpos -1* and concatenates the maps. Computes the eigenvalues and the eigenvectors of the first order one-turn map.
Dependencies	Uses the subroutines *mxone*, *dzero*, *rdcopy*,*trmap*, *mapcat*,*ddcopy* and *eigen*.

* * *

4.15 Mktunemad2

General description. Analyze the tunes (also in the presence of coupling), computes the minimum tune approach as a measure of the global linear coupling and computes the uncoupled dispersion and chromaticity.

Detailed description	If the input parameter *c_orbit* is True, the tunes are introduced and analyzed. The closed orbit is recomputed. If the input argument of *mktunemad2* is true, *trackb2* is called, so the orbits of all bunches including second order terms and interactions, although not considering the dispersion, are tracked. For all the bunches both the initial first order dispersion and the initial second order dispersion are computed. Computes the horizontal and vertical motion and tracks the dispersion, which is the change of orbit for a given variation of the energy ($\delta p/p$). The horizontal and vertical tunes in the presence of coupling are computed. Also, the beta and alpha functions are recomputed in the presence of coupling at the interaction points 1 and m for beams 1 and 2 respectively. The minimum tune approach is also computed for both beams and written at the output files *fort.closest_tune_app*. The uncoupled chromaticity and second-order dispersion are computed.
Input	*flag*: If true all bunches are tracked without dispersion and with beam-beam interactions and the tune, chromaticity and dispersion are computed for all bunches. Otherwise, bunch 1 is tracked without beam-beam interactions and the tune, chromaticity and dispersion are computed for the first bunch.
Dependencies	Uses the subroutines *trackb*, *track0*, *eigenmad*, *twisspar*, *lnrcoup*,*disp* and *eigen*.

* * *

4.16 Orbitb2

General description. Finds the closed orbits of all bunches and both beams with beam-beam interactions.

Detailed description	Solves the closed orbit with beam-beam interactions. Initialize the phase coordinates for beam 1 and beam 2 with the coordinates of the first bunch of both beams, which orbit without beam beam interactions has been computed in MAIN program. The input variable *f_second* allows to select either to include second order terms or not. Two loops take place. In the inner one, the closed orbit is solved assuming fixed kicks, i.e., solving $\bar{x}_{i,j,k+1} = M'_{i,j}(\bar{x}_{i,j,k})$, where $M'_{i,j} = \prod_l^{N_{BB}} M_{i,l} M^{BB}_{i,j,l}$, $M^{BB}_{i,j,l} : x'_{i,j} \to x'_{i,j} + \Delta x'_{coh}(x_{i,j} - x_{S(i,j,l)})$ is the one turn map once the beam-beam interactions have been introduced following the collision schedule $S(i,j,l)$, that indicates the bunch index of the other beam colliding with beam i and bunch j at the interaction point l. In the outer loop the bunch positions are updated until the convergence of both closed orbits into a stable solution.
Dependencies	Uses the subroutines *trackb* and *solver*.

* * *

4.17 Print2

General description. Generates the output files containing information about the orbit, the dispersion and the tunes and chromaticities.	
Detailed description	Construct and output files containing information about: • Tunes and chromaticities • The offset, slope and dispersion per pit • The separation and crossing angles per crossing point

Inputs	*flag*: If true, outputs the results for the unperturbed beams. Otherwise, the results with beam-beam interactions are outputted. *tunes*: If true, gives the results for tune, chromaticity and dispersion. Otherwise, outputs the files of orbit.
Dependencies	Uses the subroutines *prsclb*, *prlumi*, *pravlumi*, *prsobs*, *prsep* and *prscl0*.

* * *

4.18 Initrack

General description. Initializes strong-strong tracking of the output bunches.	
Detailed description	Initializes Strong-Strong tracking. If *outbcnt* is greater than 0, it opens the bunch output units. Computes the factor for luminosity as $lumifact = \frac{1}{4\sqrt{\sigma_{x,1}\sigma_{y,1}\sigma_{x,2}\sigma_{y,2}}}$. It also sets to 0 the kick maps for both beams (*sk1* and *sk2*) and for the closed orbit for all bunches(*ztr*). Depending on the input parameter *amp_bunch*, the orbit amplitude is computed differently. If *amp_fac* is different from 0, the orbit amplitude is computed as $orb\_amp = \sigma A$, being A the amplitud. If *amp_fac* is 0, it also includes an stochastic value in $vx = \sigma_x \times A \times random$ and $vy = \sigma_x \times A \times random$.
Dependencies	Uses the function *rg32cut*. Uses the subroutine *dzero*.

* * *

4.19 Bottrack

General description. Tracks all output bunches one full turn from the initial position.	
Detailed description	Tracks all bunches one turn from initial position. For all interaction points and all bunches, looks if the bunch is still present in both rings. If the bunch is present in one ring, tracks the new position through the subroutine *gaptrack*. Checks if the bunch is also on the other ring and for every bunch present in both rings, finds the bunch of beam 2 that interact with the considered bunch of beam 1. Selects the required factor depending on if the interaction is head on or not. Sums the orbit amplitudes to the computed orbits. Track the pair of bunches over the given interaction point and compute the orbit distortions. If the considered bunch is an out bunch, it computes the number of bunches that contribute to the luminosity and the average luminosity.
Dependencies	Uses the subroutines *gaptrack*, *bbtrac2*, *wtrack* and *trstat*.

* * *

4.20 Pbunch

General description. Prints global tracking results.	
Detailed description	Prints global tracking results, the normalized average luminosity, that computes as $lumiav/lumicnt$, being $lumiav$ the average luminosity and $lumicnt$ the number of bunches that contribute to the luminosity.

* * *

Chapter 5

Subroutines

In the following, the subroutines that are called within the subroutines of the main program are listed and explained. The dependencies of each subroutine have also been specified, indicating which functions and subroutines call each function. In this case, there is not a sequential order, so the subroutines have not been ordered.

Just the subroutines that are connected to the main program are explained. In the file of TRAIN there are more subroutines that are not called.

* * *

5.1 Read_opth

General description. Reads and saves the variables of the header of the Twiss table, as well as the number and name of columns of the file.

Detailed description	Reads the header of the file stored in *unit* and quits in the line before reading the data of the file, marked with a \$ symbol. Extracts the information of the Twiss table regarding the type of particle the beam is formed with, the name of the sequence, the energy (GeV), the base beam current (A), the horizontal and vertical nominal tunes and chromaticities, the length of the accelerator (m), the horizontal and vertical emittances and the rms energy spread (q-Gaussian) (*SIGE*). Saves the number of data columns in *ncolumns* and the name of the data variables stored in the file (the line starting with *) in c_names.

Input	*unit*: Integer. Input unit where the Twiss table has been opened. *nbeam*: Integer. Number of the beam (1 or 2).
Outputs	*particle*: String. Type of particle forming the beam. *seq_name*: String. Name of the used sequence. *gev_l*: Float. Energy of the beam. *bcurr_l*: Float. Beam current of the beam. *xix*: Float. Horizontal chromaticity. *xiy*: Float. Vertical chromaticity. *qx*: Float. Horizontal tune. *qx*: Float. Vertical tune. *circum*: Float. Accelerator circumference (m). *delta*: Float. Difference between the reference momentum and the design momentum, divided by the design momentum. *epsx0*: Float. Horizontal emittance. *epsy0*: Float. Vertical emittance. *deltap*: Float. The relative energy spread (σ_E/E). *ncolumn*: Float. Number of columns of the Twiss file. *c_names*: String. Name of the columns in the Twiss file.
Dependencies	Uses the functions *get_tockens* and *lastnb*.

* * *

5.2 Get_cpos

General description. Gives the order of appearance of the different variables in the columns of the Twiss file.	
Detailed description	Given the length of an array n containing the labels *'NAME', 'S', 'X', 'BETX', 'ALFX', 'DX', 'Y', 'BETY', 'ALFY'* and *'DY'* finds their position in *names* and saves the order of appearance in an array *pos*.
Inputs	*n*: Integer. Number of columns. *names*: String. Name of the columns.

Outputs	*pos*: Integer array. Contains the position of appearance in the array of names *names* of the strings *'NAME', 'S', 'X', 'BETX', 'ALFX', 'DX', 'Y', 'BETY', 'ALFY'* and *'DY'*.

* * *

5.3 Getmask2

General description. Generates a mask in half slots marking the long-range interaction points with a 1 and the head-on ones with a 2.	
Detailed description	Creates a mask *mask* along all the ring indicating all the possible interaction points (twice the number of buckets, since there is an interaction every half bucket). The origin of the mask is chosen to be the at the leftmost parasitic interaction of IP5 if it is an active pit. Otherwise, the origin is chosen to be at the leftmost parasitic interaction of the first active interaction point the order 5,8,1 or 2 (at least one active pit must exist). The sense in which the mask is placed also is consistent with the previous ordering, if IP5 is active, it will go from IP5 towards IP6. The position of the head-on collisions indicated with a 2 and the position of the long-range interactions with a 1. The total number of collision points is also computed.
Input	*mdslt*: Integer. Half buckets. *npart*: Integer(8). One side parasitic interactions. *circ*: Float. Accelerator circumference. *iact*: Integer(8). Indicates the pits that are active. For instance, $iact = [1,0,0,0,1,0,0,0]$ would indicate that pits 1 and 5 are active (interactions take place). *ip*: Integer(8). Array containing the s position of the interaction points.

Outputs	*mask*: Integer array of length *mdslt*. Half slot mask for the injection scheme. Marks with a 1 the long-range interaction points and with a 2 the head-on ones. Starts at the leftmost parasitic interaction of IP5 if present. If not, it starts at the leftmost parasitic interaction of the IP 5,8,1,2 in order of appearance. At least one active IP is required by TRAIN. *maski*: Integer array of length *mdslt*. Contains the cumulative sum of the number of interaction points (either head-on or long-range) or 0 if there is not collision in the slot. *count*: Integer. Total number of head-on and long-range interactions.
Dependencies	Uses the function *rotate*.

* * *

5.4 Collide

General description. Simulates the collision of all bunches of beam 1 with the ones of beam 2.

Detailed description	Simulates the collision between both beams at the interaction points (long-range and head-on). Taking into account the offset between beams, iterates over the filling scheme of beam 1 and checks if that particular bunch is inside an interaction point (half buckets) using the mask that has already been defined. For each bunch of beam 1 finds out with which bunch of beam 2 it would interact at a certain interaction point and if due to the filling scheme that half bucket is filled or not. If there is an interaction (either head-on or long-range), the outputs of this subroutine are updated, *colcnt_f* and *colcnt_b,* where the number of collisions of each bucket is saved for the forward and backward beam respectively, *list_f* and *list_b,* which save the collision point number where the collision takes place. Finally, in *part_f* and *part_b,* the bunch of the backward and forward beam interact are also saved. If there is a collision for a forward or backward bunch, the arrays *part_f* and *part_b* are sorted according to the order of occurrence, colcnt_f and colcnt_b.
Inputs	*mbuck*: Integer. Total number of buckets *mdslt*: Integer. Half buckets. *mcol*: Integer. Maximum number of collision points. *npart*: Integer(8). One side parasitic interactions. *b2off*: Integer. Offset of beam 2 (in half-buckets) with respect to beam 1 at IP5 (negative towards IP4, positive towards IP6). *a*: Integer(2,mbuck). Filling scheme. *mask*: Integer array of length *mdslt.* Half slot mask for the injection scheme. Marks with a 1 the long-range interaction points and with a 2 the head-on ones. Starts at the leftmost parasitic interaction of IP5 if present. If not, it starts at the leftmost parasitic interaction of the IP 5,8,1,2 in order of appearance. At least one active IP is required by TRAIN.

	maski: Integer array of length *mdslt*. Contains the cumulative sum of the number of interaction points (either head-on or long-range) or 0 if there is not collision in the slot.
Outputs	*colcnt_f*: Integer(mbuck). Number of collisions of each bunch of beam 1 with the bunches of beam 2. *list_f*: Integer(mcol,mbuck). Stores the cumulative number of collisions as a function of the collision number of the given bunch of beam 1 and the number of the bunch of beam 1 that is colliding. *part_f*: Integer(mcol,mbuck). Stores the number of the bunch of beam 2 as a function of the collision number of the given bunch of beam 1 and the number of the bunch of beam 1 with which it is colliding. *colcnt_b*: Integer(mbuck). Number of collisions of each bunch of beam 2 with the bunches of beam 1. *list_b*: Integer(mcol,mbuck). Stores the cumulative number of collisions as a function of the collision number of the given bunch of beam 2 and the number of the bunch of beam 2 that is colliding. *part_b*: Integer(mcol,mbuck). Stores the number of the bunch of beam 1 as a function of the collision number of the given bunch of beam 2 and the number of the bunch of beam 2 with which it is colliding.
Dependencies	Uses the subroutines *selipnum* and *mysort*.

* * *

5.5 Equ_class

General description. Finds equivalent classes.

Detailed description	The set of buckets in which the same number of collisions occur build an equivalent class. The higher and lower number of collisions are found, i.e., the higher and lower equivalent classes are found. Being *i* the number of the bucket with a given number of collisions and *j* the id number of the class, in *lordl(i,j)* the number of the bucket is saved. In *nordl(i)* the number of buckets belonging to the class *i* is saved, and in *cordl(i)* the number of collisions of the corresponding class *i*. It also saves in *nequl* the number of buckets within the class *i* that have a different number of collisions that have collided in one point (i.e., if two points of different classes *i* and *j* have coincided in a concrete point, then *nequl(i)* and *nequl(j)* would be incremented by 1). In *lequl(j,i)*, the id of the class of the buckets that have collided at the same point is stored.
Inputs	*mbuck*: Integer. Total number of buckets *mcol*: Integer. Maximum number of collision points. *a*: Integer(2,mbuck). Filling scheme. *colcnt*: Integer(mbuck). Number of collisions of each bunch of one beam (either 1 or 2) with the bunches of the other beam. *list*: Integer(mcol,mbuck). Stores the cumulative number of collisions as a function of the collision number of the given bunch and the number of the bunch that is colliding of one of both beams.
Outputs	*low*: Integer. Minimum number of collisions *high*: Integer. Larger number of collisions. *equl*: Integer. Total number of equivalent classes. *cequl*: Integer. Number of collisions of each bunch of one beam (either 1 or 2) with the bunches of the other beam. *nequl*: Integer. The number of buckets within the class *i* that have a different number of collisions which have collided in one point. *lequl*: Integer. Id of the class of the buckets that have collided at the same point. *ordl*: Integer. Number of ordered collision lists (i.e. number of one's in the filling scheme).

	cordl: Integer. Number of collisions of a given class. *nordl*: Integer. Number of buckets belonging to a given class. *lordl*: Integer. Number of the bucket as a function of the buckets where collisions occur and the id of the class.

* * *

5.6 Get_hits

General description. Finds the bunch slot mask for all equivalent classes.	
Detailed description	Fills *hitlist* of beam 1 according to the equivalent classes, indicating with a 1 where a hit is produced. Loops over both hitlists *hitlist_f* and *hitlist_b*, so the position in the *hitlist*of beam 1 is set to 1 if the position of the bunch of beam 2 that interact with that bunch of beam 1 is set to 0 and the other way around. loops until all bunches of beam 1 and 2 are assembled.
Inputs	*mbuck*: Integer. Total number of buckets *mcol*: Integer. Maximum number of collision points. *part_f*: Integer(mcol,mbuck). Stores the number of the bunch of beam 2 as a function of the collision number of the given bunch of beam 1 and the number of the bunch of beam 1 with which it is colliding. *part_b*: Integer(mcol,mbuck). Stores the number of the bunch of beam 1 as a function of the collision number of the given bunch of beam 2 and the number of the bunch of beam 2 with which it is colliding. *colcnt_f*: Integer(mbuck). Number of collisions of each bunch of beam 1 with the bunches of beam 2. *colcnt_b*: Integer(mbuck). Number of collisions of each bunch of beam 2 with the bunches of beam 1. *equl*: Integer. Total number of equivalent classes.

	nequl: Integer. The number of buckets within a given class that have a different number of collisions which have collided in one point. *lequl*: Integer. Id of the class of the buckets that have collided at the same point.
Outputs	*hitlist_f*: Bunch slot mask for all equivalent class bunches of beam 1. *hitlist_b*: Bunch slot mask for all equivalent class bunches of beam 2.

* * *

5.7 Prsup

General description. Seeks the PACMAN and super-PACMAN bunches.	
Detailed description	For both the forward and backward beams, counts the number of collisions that experience each bunch in order to infer the number of nominal collisions (all possible collisions with no PACMAN bunches). Seeks the super-PACMAN bunches (missing head on collision in the pits) for each pit. Finds out the number of nominal bunches, the number of super-PACMAN bunches, the super-PACMAN counts per active pit and the super-PACMAN multiplicity counts for both the forward and the backward beams.

* * *

5.8 Prcdmp

General description. Writes the information of the equivalent classes, the collision scheme, the set of forward and backward bunches and the list of regular bunches in files.

Detailed description	Saves the data regarding to the equivalent classes, the collision scheme, the set of forward and backward bunches and the list of regular bunches in the files *freq_f.count*, *coll.count*,*freq_b.count*, *equ_f.count*, *equ_b.count*, *set_f.list*,*set_b.count*, *alt.list*, *reg.list* if the input variables *w_coll*, *w_freq*, *w_detail*, *w_equ*, *w_set* and *w_alt* are true.

* * *

5.9 Mysort

General description. Sorts the first array in ascendant order and the second one in the same order as the first one.	
Detailed description	Sorts the second array l in ascendant order and the third one p in the same order as l. Returns the sorted arrays.
Inputs	n: Integer. Number of elements to sort. l: Integer array. Array to sort in ascendant order. p: Integer array. Array to sort the same order as l.
Outputs	l: Integer array. Sorted array. p: Integer array. Sorted array.

* * *

5.10 Track02

General description. Obtains the one-turn map without beam-beam interactions concatenating the input sector maps.	

Detailed description	Initializes the one-turn map (*tr*) for bunch 1 of both beams and computes the transfer map of the first bunch starting in the first interaction point (0 for beam 1 and m for beam 2). Obtain the transfer map concatenating all the input sector maps, including the ones of extra elements. Saves the orbit coordinates z of turn k and turn $k+1$ at the origin of the transfer map. Finds out the one-turn map for all bunches and both beams without beam-beam interactions.
Inputs	*fsec*: Boolean. Second order maps flag. If True, the second order maps are also used in the tracking. *fdisp*: Boolean. Dispersion flag. If true, the dispersion is also computed.
Dependencies	Uses the subroutines *mxone, dzero, rdcopy, trmap, mapcat, trdisp* and *drcopy*.

* * *

5.11 Solver

General description. solves the linear equation $A \cdot X = B$ using the Gauss-Jordan elimination method.

Detailed description	Given the augmented matrix of the system *augmat*, it solves the linear equation $A \cdot X = B$ using the Gauss-Jordan elimination method.
Inputs	*augmat*: Float(ndim,mdim). Augmented matrix. *ndim*: Integer. Number of rows. *mdim*: Integer. Number of added columns.
Outputs	*irak*: Integer. Number of independent solutions.

* * *

5.12 Disp

General description. Computes the initial values of the 4 component dispersion vector.

Detailed description	Computes the initial values of the 4 component dispersion vector. If it is unable of finding a determined solution, sets the dispersion to 0. Given an input matrix r and a vector of size 4 *aux*, solves $(R-I)B = AUX$ for B using the Gauss-Jordan elimination Method.
Inputs	*r*: Float(6,6). Input matrix. *aux*: Float(6). Auxiliar vector.
Outputs	*d*: Real(6). dispersion vector.
Dependencies	uses the subroutine *solver*.

* * *

5.13 Mxone

General description. Generates a block identity matrix of dimension n in a squared m dimension matrix.

Detailed description	Generates a block identity matrix of dimension n in a squared m dimension matrix.
Inputs	*target*: Float(m,m). Input matrix. *m*: Integer. Dimension of the squared matrix.
Outputs	*n*: Integer. Dimension of the block intensity.

* * *

5.14 Trackb2

General description. Finds the one turn map of all the bunches once the beam-beam interactions have been introduced.

Detailed description	If the input parameter *bno* is negative, it tracks all bunches. Otherwise it just tracks the bunch number *bno*. If the input parameter *fsec* is true, second order terms are also included. If *fdis* is true, dispersion terms are included. Initializes the map for the given bunch and tracks it through the sector 0, concatenates the maps and include dispersion if needed using the subroutine *trdisp*. Tracks the bunch or the bunches through all points connected by sector maps. Finds which bunches of both beams interact and computes the beam-beam map, that concatenates to the sector maps at the interaction points. In order to do so, uses the map mask for selecting the maps associated with an interaction points. Computes the coherent beam-beam factor $ccp = \pm 1 \frac{beam\_current}{ev}$, where the sign can be positive or negative depending on the kind of particles that are interacting, e is the charge of the electron and $v = \frac{c}{circum}$ the accelerator frequency, with c the speed of light and $circum$ the accelerator circumference. If the interaction point coincides with a pit, it multiplies ccp by the head on factor and by the $xifact$ otherwise. The distance between both beams and $\sigma_x = \sqrt{\sigma_x^2(1) + \sigma_x^2(2)}$ and $\sigma_y = \sqrt{\sigma_y^2(1) + \sigma_y^2(2)}$ are also calculated, that are going to be required while computing the elements of the beam-beam map. The orbit coordinates of both beams are saved before and after finding the one-turn map.
Inputs	*fsec*: Boolean. Second order maps flag. If True, the second order maps are also used in the tracking. *fdisp*: Boolean. Dispersion flag. If true, the dispersion is also computed. *bno*: Integer. Bunch number.
Dependencies	Uses the subroutines *mxone, dzero, rdcopy, trmap, mapcat, trdisp,trbb* and *drcopy*.

* * *

5.15 Trbb

General description. Computes the orbit once the coherent beam-beam kick has been applied, the beam-beam one-turn map and the second order beam-beam map.	
Detailed description	Computes the orbit once the coherent beam-beam kick has been applied, the beam-beam one-turn map and the second order beam-beam map. If *fsec* is true, it also computes the second order beam-beam map. The beam-beam first and second order maps *re* and *te* are initialized and the factor for the beam-beam kick $fk = \frac{2r_0 ccp}{\gamma}$ is found. It also computes xs and ys, which are the x and y projections of the distance between bunches of beam 1 and beam 2 in each interaction point. The exact shape of the beam-beam maps can be found explicitly at Subroutines.
Inputs	*fsec*: Boolean. Second order maps flag. If True, the second order maps are also used in the tracking. *ccp*: Float. Coherent beam-beam factor. *sx*: Float. Horizontal beam size. *sy*: Float. Vertical beam size. *xm*: Float. Horizontal bunch-bunch separation. *yx*: Float. Vertical bunch-bunch separation.
Outputs	*orbit*: Float(6). Modified orbit after the coherent kick. *re*: Float(6,6). First order beam-beam map. *te*:Float(6,6,6). Second order beam-beam map.
Dependencies	Uses the subroutines *mxone*, *dzero* and *errf*.

Mathematical development

Two different cases are contemplated depending on if the beams are round or elliptic:

- $\sigma_x = \sigma_y$: for round beams. In this case, it computes the factor $\rho^2 = xs^2 + ys^2$. If $xs = xy = 0$, the elements (2,1) and (4,3) of the first order transfer map are set to $\frac{fk}{2\sigma_x^2}$. Otherwise, it computes the kick as $\Delta x' =$

$\frac{2r_0(\pm1)N}{\gamma}\frac{xs}{\rho^2}\left(1-e^{-\frac{\rho^2}{2\sigma}}\right)$ and the first order map as

$$re = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ \Delta x' & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \Delta x' & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The kick is added to x' and y'. The first and second order orbit effects are computed, resulting in the first order map

$$re = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ \Delta_{x,x} & 1 & \Delta_{x,y} & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \Delta_{x,y} & 0 & \Delta_{y,y} & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

and in the second order map

$$te(2) = \begin{pmatrix} \Delta_{x,x,x} & 0 & \Delta_{x,x,y} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \Delta_{x,x,y} & 0 & \Delta_{x,y,y} & 0 & 0 & 0 \\ & & & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$te(4) = \begin{pmatrix} \Delta_{x,x,y} & 0 & \Delta_{x,y,y} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \Delta_{x,y,y} & 0 & \Delta_{y,y,y} & 0 & 0 & 0 \\ & & & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

where the elements in the maps are

$$\Delta_s = \frac{2r_0(\pm1)N}{\gamma}\frac{s}{\rho^2}\left(1-e^{-\frac{\rho^2}{2\sigma}}\right) \tag{5.1}$$

$$\Delta_{s,s} = \frac{2r_0(\pm1)N}{\gamma}\left(-\left(1-e^{-\frac{\rho^2}{2\sigma^2}}\right)\frac{xs^2-ys^2}{\rho^4}+\frac{s^2}{\rho^2\sigma^2}e^{-\frac{\rho^2}{2\sigma^2}}\right) \tag{5.2}$$

$$\Delta_{x,y} = \frac{2r_0(\pm1)N}{\gamma}\left(-\left(1-e^{-\frac{\rho^2}{2\sigma^2}}\right)\frac{2xsys}{\rho^4}+\frac{xsys}{\rho^2\sigma^2}e^{-\frac{\rho^2}{2\sigma^2}}\right) \tag{5.3}$$

and $s = x, y$.

- $\sigma_x \neq \sigma_y$: Elliptic beams. The cases in which $\sigma_x > \sigma_y$ and $\sigma_x < \sigma_y$ are considered. The procedure explained above is repeated for the general case, so we now consider that $r = \sqrt{2(\sigma_x^2 + \sigma_y^2)}$ and the elements of the first and second order maps

$$\Delta_x = \frac{2r_0(\pm 1)N}{\gamma}\left(cry - e^{-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2}+\frac{y^2}{\sigma_y^2}\right)}cby\right) \tag{5.4}$$

$$\Delta_y = \frac{2r_0(\pm 1)N}{\gamma}\left(crx - e^{-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2}+\frac{y^2}{\sigma_y^2}\right)}cbx\right) \tag{5.5}$$

$$\Delta_{x,x} = \frac{1}{\sigma_x^2-\sigma_y^2}\left(-(x\Delta_x + y\Delta_y) + \frac{2r_0(\pm 1)N}{\gamma}\left(1 - \frac{\sigma_y}{\sigma_x}e^{-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2}-\frac{y^2}{\sigma_y^2}\right)}\right)\right) \tag{5.6}$$

$$\Delta_{x,y} = \frac{1}{\sigma_x^2-\sigma_y^2}\left(-(x\Delta_y - y\Delta_x)\right) \tag{5.7}$$

$$\Delta_{y,y} = \frac{1}{\sigma_x^2-\sigma_y^2}\left(x\Delta_x + y\Delta_y + \frac{2r_0(\pm 1)N}{\gamma}\left(1 - \frac{\sigma_y}{\sigma_x}e^{-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2}-\frac{y^2}{\sigma_y^2}\right)}\right)\right) \tag{5.8}$$

It also computes the first and the second order orbit effects.

$$\Delta_{x,x,x} = \frac{1}{2(\sigma_x^2-\sigma_y^2)}\left(-\Delta_x - (x\Delta_{x,x} + y\Delta_{x,y}) + \frac{2r_0(\pm 1)N}{\gamma}x\frac{\sigma_y}{\sigma_x^3}e^{-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2}-\frac{y^2}{\sigma_y^2}\right)}\right) \tag{5.9}$$

$$\Delta_{x,x,y} = \frac{1}{2(\sigma_x^2-\sigma_y^2)}(-\Delta_y - (x\Delta_{x,y} - y\Delta_{x,x})) \tag{5.10}$$

$$\Delta_{x,y,y} = \frac{1}{2(\sigma_x^2-\sigma_y^2)}(\Delta_x - (x\Delta_{y,y} - y\Delta_{x,y})) \tag{5.11}$$

$$\Delta_{y,y,y} = \frac{1}{2(\sigma_x^2-\sigma_y^2)}\left(\Delta_y + (x\Delta_{x,y} + y\Delta_{y,y}) + \frac{2r_0(\pm 1)N}{\gamma}y\frac{\sigma_x}{\sigma_y^3}e^{-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2}-\frac{y^2}{\sigma_y^2}\right)}\right) \tag{5.12}$$

* * *

5.16 Dzero

General description. Returns a double precision array of zeros	
Detailed description	Returns a double precision array of zeros *d_in* of length *d_count*.

Inputs	*d_in*: Float. Array to initialize. *d_count*: Integer. Length of the array.
Outputs	*d_in*: Float. Array of zeros and length *d_count*.

* * *

5.17 Rdcopy

General description. Converts a real array into a double precision array.	
Detailed description	Given a real array *d_in*, a double precision array *d_out* and the length of the array *d_count*, it copies real to double precision arrays.
Inputs	*d_in*: Real array. Array to be copied. *d_count*: Integer. Number of elements to be copied.
Outputs	*d_out*: Float array. Copy of the array *d_in*.

* * *

5.18 Trmap

General description. Tracks the orbit through a sector.	
Detailed description	Tracks the orbit through a sector. $$z_j^{(2)} = \Delta z_j + \sum_{k=1}^{6} R_{jk} z_k^{(1)} + \sum_{k=1}^{6}\sum_{l=1}^{6} T_{jkl} z_k^{(1)} z_l^{(1)} \quad (5.13)$$ Computes the transport map rt. $$rt_{ij} = re_{ij} + 2\sum_{l=1}^{6} T_{ijl} z_l \quad (5.14)$$
Inputs	*ek*: Float(6). Kick. *re*: Float(6,6). First order sector map. *te*: Float(6,6,6). Second order sector map.

Outputs	*orbit*: Float(6). Orbit through the tracked sector. *rt*: Float(6,6). First order one turn map.

* * *

5.19 Gaptrack

General description. Tracks a given bunch from an interaction point to the another.	
Detailed description	Tracks one bunch over the interaction gap between *inpt1* and *inpt2*. If *dir* is greater than 0, tracks for ring 1 (forward beam). Otherwise, tracks for ring 2 (backward beam). Identifies the start and end point of the tracking and computes the orbit *ztr* for the forward or backward beam and the bunch *bnum*.
Inputs	*bnum*: Integer.Bunch number. *inpt1*: Integer. Interaction point where the tracking will start. *inpt2*: Integer. Interaction point where the traking will stop. *dir*: Integer. Number od the beam (1 or 2).
Dependencies	Uses the subroutine *trmap*.

* * *

5.20 Mapcat

General description. Computes the product of two transfer maps.	
Detailed description	Computes the product of two transfer maps of first order *ra* and *rb*. If *fsec* is true, it also computes the second order product.
Inputs	*fsec*: Boolean. Second order maps flag. If True, the second order maps are also used in the tracking. *rb*: Float(6,6). First order map.

	tb: Float(6,6,6). Second order map. *ra*: Float(6,6). First order map. *ta*: Float(6,6,6).Second order map.
Outputs	*rd*: Float(6,6). First order map product of the two input ones. *td*: Float(6,6,6).Second order map product of the two input ones.
Dependencies	Uses the subroutine *ddcopy*.

* * *

5.21 Ddcopy

General description. Copies double precision to double precision arrays.	
Detailed description	Copies double precision to double precision arrays.
Inputs	*d_in*: Float. Input array. *d_count*: Integer. Length of the array.
Outputs	*d_in*: Float. Array copy of *d_in*.

* * *

5.22 Eigen

General description. Finds the eigenvectors and eigenvalues of the input matrix.	
Detailed description	Given an $n \times n$ matrix *a*, it finds the eigenvectors and eigenvalues of *a*.
Inputs	*a*: Float(n,n). Input array to diagonalize. *n*: Integer. Dimension of the input matrix.
Outputs	*q1*: Float. Eigenvalue of *a*. *q2*: Float. Eigenvalue of *a*.
Dependencies	Uses the subroutines *orthes, ortran* and *hqr2*.

* * *

5.23 Orthes

General description. Transforms an asymmetric real matrix to upper Hessenberg form applying successive orthogonal transformations.	
Detailed description	Converts an asymmetric real matrix *a*, to upper Hessenberg form applying successive orthogonal transformations. Translation of the algorithm procedure orthes in: Handbook series linear algebra, num. math. 12, 349-368 (1968) by R. S. Martin and J. H. Wilkinson.
Inputs	*n*: Integer. Order of the matrix *a*. *ndim*: Integer. Order of the matrix *a*. *ilow, iupp*: Integer. Determine a submatrix, set by balanc. May be set to 1 and n respectively. *a*: Real(ndim,n). Input matrix.
Outputs	*a*: Float(ndim,n). The matrix *a*, converted to upper hessenberg. The lower triangle contains information about the orthogonal transformations. *d*: Float(n). Further information.

* * *

5.24 Ortran

General description. Accumulate the orthogonal similarity transformation used by *orthes* to reduce a general real matrix a to upper hessenberg form.	
Detailed description	Accumulate the orthogonal similarity transformation used by *orthes* to reduce a general real matrix a to upper hessenberg form. Translation of the algol procedure *ortrans* in: Handbook series linear algebra, num. math. 16, 181-204 (1970) by G. Peters and J. H. Wilkinson.
Inputs	*n*: Integer. order of the matrices *a* and *v*. *ndim*: Integer. Order of the matrices *h* and *v*.

	ilow, iupp: Integer. Determine a sub-matrix, set by balanc. May be set to 1 and n respectively. *h*: Real(ndim,n). The matrix resulting from running orthes. *d*: Float(n). Further information about the transformation.
Outputs	*v*: Float(ndim,n). The accumulated transformation. *d*: Float(n). Destroyed.

* * *

5.25 Hqr2

General description.	
Detailed description	Finds eigenvalues and eigenvectors of an unsymmetric real matrix, *a* which has been reduced to Upper Hessenberg form, *h*, by the subroutine *orthes*. The orthogonal transformations must be placed in the array *vecs* by subroutine *ortran*. Translation of the algol procedure hqr2 in: Handbook series linear algebra, num. math. 16, 181 - 204 (1970) by G. Peters and J. H. Wilkinson.
Inputs	*n*: Integer. order of the matrices *a* and *v*. *ndim*: Integer. Order of the hessemberg matrix *h*. *ilow, iupp*: Integer. Determine a sub-matrix, set by balanc. May be set to 1 and n respectively. *h*: Real(ndim,n). The hessemberg matrix resulting from running orthes. *vecs*: Float(ndim,n). a square matrix of order n containing the similarity transformation from a to h.
Outputs	*h*: Real(ndim,n). Modified. *wr*: Float(n). Real parts of eigenvalues of *h* (or *a*). *wi*: Float(n). Imaginary parts of eigenvalues of *h* (or *a*). *vecs*: Float(ndim,n). the unnormalized eigenvectors of *a*. Complex vectors are stored as pairs of reals.

* * *

5.26 Mxmpy

General description. Multiply two matrices.	
Detailed description	Multiply two matrices.
Inputs	*fact1*: Float(m,m). Input matrix. *fact2*: Float(m,m). Input matrix. *m*: Integer. Dimension of the input and output matrix. *n*: Integer. Dimension of the block matrix that it is going to be multiplied.
Outputs	*target*: Float(m,m) Product of both matrices.

* * *

5.27 Bbtrac2

General description.	
Detailed description	Tracks one bunch pair over the interaction point-origin of the orbit. Computes the x and y projection of the distance between the bunch number *bnum1* of beam 1 and the bunch number *bnum2* of beam 2 in the interaction point *intp*. It also computes the bunch size σ and the horizontal and vertical kick felt by the bunches at their unperturbed distance and considering the perturbation. Returns the x and y components of the perturbed orbit of bunches *bnum1* and *bnum2* of beam 1 and 2. It uses the long-range factor *fact* given as an input variable for computing the kick.
Inputs	*bnum1*: Integer. Bunch number of beam 1. *bnum2*: Integer. Bunch number of beam 2. *intp*: Integer. Interaction point. *fact*: Float. Long-range factor.

Dependencies	Uses the functions *sigx* and *sigy*. Uses the subroutine *bbtr2*.

* * *

5.28 Wtrack

General description. Tracks bunches of *outblist*.	
Detailed description	If *outnorm* is larger than 0, the number of the iteration in which *bottrack* is called and the orbit of both the input *bunch1* and *bunch2* are written in the file unit 30+ *index of the out bunch in outblist* and generates files *bunch.* depending on the corresponding IP. It also writes the transfer maps for bunch 1 and bunch 2.
Inputs	*bunch1*: Integer. Bunch number of beam 1. *bunch2*: Integer. Bunch number of beam 2. *lp*: Integer. Number of output bunch.
Dependencies	Uses the functions *sigx* and *sigy*.

* * *

5.29 Bbtr2

General description. Computes the transport map and the kick for beam-beam element.

Detailed description	Computes the transport map and the kick for beam-beam element. Initializes the output kicks *phix* and *phiy* and computes the beam-beam kick factor. Two different cases are contemplated: • $\sigma_x = \sigma_y$: for round beams. It computes the horizontal and vertical incoherent kicks as follow $\Delta s' = \frac{2r_0N}{\gamma}\frac{s}{\rho^2}\left(1-e^{-\frac{-\rho^2}{2\sigma^2}}\right) \quad (5.15)$ with $s = \{x, y\}$. • $\sigma_x \neq \sigma_y$: For elliptic beams, the general case. The kick is computed defining $r = \sqrt{2(\sigma_x^2 - \sigma_y^2)}$, so $\Delta x' = \frac{2r_0N}{\gamma}\frac{\sqrt{\pi}}{r}\left(errf\left(\frac{xs}{r},\frac{ys}{r}\right) - errf\left(\frac{\sigma_y}{\sigma_x}\frac{xs}{r},\frac{\sigma_x}{\sigma_y}\frac{ys}{r}\right)e^{-\frac{1}{2}\left(\frac{xs^2}{\sigma_x^2}+\frac{ys^2}{\sigma_y^2}\right)}\right) \quad (5.16)$
Inputs	*fsec*: Boolean. Second order maps flag. If True, the second order maps are also used in the tracking. *ccp*: Float. Coherent beam-beam factor. *sx*: Float. Horizontal beam size. *sy*: Float. Vertical beam size. *xm*: Float. Horizontal bunch-bunch separation. *yx*: Float. Vertical bunch-bunch separation.
Outputs	*phix*: Float. Horizontal coherent kick. *phiy*: Float. Vertical coherent kick.
Dependencies	Uses the subroutine *errf*.

* * *

5.30 Errf

General description. Computes the two dimensional double precision complex error function.

Detailed description	Modification of wwerf, double precision complex error function, written at CERN by K. Koelbig. Returns the error function *wx* and *wy* of the input variables *x* and *y*.
Inputs	*xx*: Float. Input variable to integrate. *yy*: Float. Input variable to integrate.
Outputs	*wx*: Float. Real part of the complex error function. *wy*: Float. Imaginary part of the complex error function.

* * *

5.31 Trdisp

General description. Tracks dispersion for one bunch through a sector.

Detailed description	Track dispersion for one bunch through a sector. Given the input matrix *re*, *te*, *da* and *dda* computes *db* and *ddb* as $$db_i = \sum_{k=1}^{6} te_{ijk} da_j \qquad (5.17)$$ and $$ddb_i = \sum_{k=1}^{6}\left(\left(\sum_{j=1}^{6} te_{ijk} da_j\right) da_k + re_{ik} dda_k\right) \qquad (5.18)$$ for $i = 1, \ldots, 4$.
Inputs	*re*: Float(6,6). First order sector map. *te*: Float(6,6,6). Second order sector map. *da*: Real(6). Dispersion vector. *dda*: Real(6). Second order dispersion vector.
Outputs	*db*: Real(6). Dispersion vector. *ddb*: Real(6). Second order dispersion vector.
Dependencies	Uses the subroutine *drcopy*.

* * *

5.32 Drcopy

General description. Copies Double to real precision arrays.	
Detailed description	Copies Double to real precision arrays.
Inputs	*d_in*: Float array. Array to be copied. *d_count*: Integer. Number of elements to be copied.
Outputs	*d_out*: Real array. Copy of the array *d_in*.

* * *

5.33 Trstat

General description. Statistical information at observation point.	
Detailed description	Statistical information at observation point. Computes the luminosity as the exponential of the perturbed distance between bunches $\sqrt{x'^2 + y'^2}$ multiplied by the factor of luminosity. If the input parameter *lumi_list* is true, the resulting value is written in *hit.list*. Counts the number of bunches contributing to the luminosity and computes the average luminosity.
Inputs	*bunch1*: Integer. Number of the bunch of beam 1. *bunch1*: Integer. Number of the bunch of beam 2.

* * *

5.34 Prsclb2

General description. Prints value table per bunch and all pits for one beam.

Detailed description	Prints value table per bunch and all pits for one beam. The values are scaled by *scale*. Writes the global results in file *train.list*, writes the orbit as a function of the slot id for the available pits and the extra elements, computes the maximum global peak to peak orbit spread and the rms of the orbit spread, outputted on the files *fort.fort.ps*.
Inputs	*fhead*: String. Identifier of the output file. *header*: String. Header of the global table that is being written in *train.list*. *bd*: Integer. Index of the beam (1 or 2). *z*: Real(6,mbunch,mmaps). Orbit for all bunches and all available elements. *index*: Integer. Coordinate that it is going to be outputted. *scale*: Float. Factor that multiplies the values of orbit.

* * *

5.35 Prlumi

General description. Prints relative luminosities.	
Detailed description	Prints relative luminosities. Computes the beam average position over all bunches for beam 1 and beam 2. Computes the maximum, minimum and the average luminosity and writes it in *train.list*. Generates the file *lumi.* and writes the luminosity per bunch as a function of the slot id.
Dependencies	Uses the functions *sigx* and *sigy*.

* * *

5.36 Pravlumi

General description. Prints relative luminosities after the elimination of the average distance	
Detailed description	Prints relative luminosities after the elimination of the average distance. Generates the files *av_lumi.* with the average luminosities after the shift as a function of the bunch id and writes in *train.list* the global results for the shifted average luminosity.
Dependencies	Uses the functions *sigx* and *sigy*.

* * *

5.37 Prsobs2

General description. Prints value table per bunch at observation points for one beam and the list of output bunches.	
Detailed description	Prints value table per bunch at observation points for one beam. The values are scaled by *scale*. Writes the global results of orbit for all output bunches at *train.list* and the selected orbit coordinate in the output position as a function of the bunch id in the files *.obs.*.
Inputs	*fhead*: String. Identifier of the output file. *header*: String. Header of the global table that is being written in *train.list*. *bd*: Integer. Index of the beam (1 or 2). *z*: Real(6,mbunch,mmaps). Orbit for all bunches and all available elements. *index*: Integer. Coordinate that it is going to be outputted. *scale*: Float. Factor that multiplies the values of orbit.
Dependencies	Uses the function *eqoptmap*.

* * *

5.38 Prsep2

General description. Prints the relative distance of the bunches of both beams as a function of the slot id of beam 1.	
Detailed description	Generates the files *hsep_sig* and *vsep_sig* and *hsep_mu* and *vsep_mu* for the selected pit. Prints the normalized bunch distance and the bunch distance in micrometers as a function of the slot id of beam 1.
Inputs	*unit*: Integer. Input unit where the file results are going to be written. *pitnum*: Integer. Index of the pit. *index*: Integer. Coordinate of the plane that is outputted.
Dependencies	Uses the function *eqvoptmap*.

* * *

5.39 Prscl02

General description. Prints the value table per interaction point for both unperturbed beams.	
Detailed description	Prints the value table per interaction point for both unperturbed beams at the global results file *train.list*. The values are scaled by *scale*.
Inputs	*header*: String. Header of the global table that is being written in *train.list*. *z1*: Real(6,mbunch,mmaps). Orbit for all bunches and all available elements for beam 1. *z2*: Real(6,mbunch,mmaps). Orbit for all bunches and all available elements for beam 2. *index*: Integer. Coordinate that it is going to be outputted. *scale*: Float. Factor that multiplies the values of orbit.
Dependencies	Uses the function *eqvoptmap*.

* * *

5.40 Prtune

General description. Prints tunes and chromaticities.	
Detailed description	Prints tunes and chromaticities. If flag is True, gives the results for the perturbed beams and generates the output files *tune_f.list* and *tune_b.list* with the bunch id, the beam current, the horizontal and vertical tune and the horizontal and vertical chromaticity and write the global results for the perturbed beam in *train.list*. The file *fort.tuneschroma* is also generated, containing the maximum peak to peak tune and chromaticity spread. Otherwise, the results for the unperturbed beams are written in *train.list*.
Inputs	*flag*: Boolean. If True, gives the results for the perturbed beams. Otherwise, the results for the unperturbed beams are written in *train.list*.
Dependencies	Uses the subroutine *matrzero*.

* * *

5.41 Eigenmad

General description. Block diagonalizes the input matrix.	
Detailed description	Given an initial symplectic matrix *inim* of dimension 4×4 finds out the block diagonal matrix following the procedure of Edwards and Teng [2] returns the two 2×2 block matrices e and f.
Inputs	*inim*: Float(n,n). Input symplectic matrix. *n*: Integer. Dimension of the input matrix.
Outputs	*e*: Float(n/2,n/2). Upper block diagonal. *f*: Float(n/2,n/2). Lower block diagonal. *deth*: Float. Dethderminant of the h matrix, i.e, $H = C + \bar{B}$. *r*: Float(2,2). Submatrix defining the transformation. *aux2*: Float. Maximum off diagonal value.
Dependencies	Uses the subroutines *symcheck*, *matrzero*, *matbar* and *mxmpy*.

Mathematical development

Given an initial 4×4 symplectic matrix M, it can be divided into 4 block 2×2 matrices

$$M = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \tag{5.19}$$

The normal form of the input matrix $M_\perp$ (block diagonal) can be found defining a similarity R_M such that transforms M into $M_\perp$

$$M_\perp = g^2 \bar{R}_M M R_M = \begin{pmatrix} E & 0 \\ 0 & F \end{pmatrix} \tag{5.20}$$

with the similarity transformation defined

$$R_M = \begin{pmatrix} \mathbb{1} & \bar{R} \\ -R & \mathbb{1} \end{pmatrix} \tag{5.21}$$

and the transformation factor g satisfying

$$g^2 \bar{R}_M R_M = \mathbb{1} \rightarrow g = |R_M|^{-\frac{1}{2}} \tag{5.22}$$

Following [3], the symplectic conjugate matrix is defined

$$\bar{R} = -S R^T S, \quad S = \begin{pmatrix} S_2 & & \\ & S_2 & \\ & & \ddots \end{pmatrix} \tag{5.23}$$

with $S_2 = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$.

The block similarity R can be found to be

$$R = -\left(\frac{1}{2}(\mathrm{Tr}A - \mathrm{Tr}D) + \frac{1}{2} sign(\mathrm{Tr}A - \mathrm{Tr}D)\sqrt{\Delta}\right)^{-1}(C + \bar{B}) \tag{5.24}$$

with $\Delta = (\mathrm{Tr}A + \mathrm{Tr}D)^2 + 4|C + \bar{B}|, \quad g = (1 + |R|)^{-\frac{1}{2}}$. Once the transformation has been found the block diagonal matrices E and F can be found applying the transformation, or simply

$$E = A - BR \tag{5.25}$$

$$F = D + RB \tag{5.26}$$

Physically, we will be interested in this kind of transformation in order to find the tunes in presence of coupling, see Subroutines, Twisspar.

* * *

5.42 Twisspar

General description. Associates the terms of the input matrix with the Twiss parameters	
Detailed description	Given a 2×2 upper or lower block diagonal matrix of a symplectic one-turn map, associates the terms of the input matrix with the Twiss parameters, so the tunes, α-function and β-function are properly calculated in the presence of coupling. The formalism has been developed following [3].
Inputs	*a*: Float(2,2). Input block diagonal sub-matrix.
Outputs	*mu*: Float. Coupled tune. *beta*: Float. Coupled beta function. *alpha*: Float. Coupled alpha function.

Mathematical development

Once the one-turn map has been converted to normal form, the Twiss parameters in presence of coupling can be extracted using the standard formulas

$$m_{\perp,i} = \begin{pmatrix} \cos\theta_i + \alpha_i \sin\theta_i & \beta_i \sin\theta_i \\ -\gamma_i \sin\theta_i & \cos\theta_i - \alpha_i \sin\theta_i \end{pmatrix} = \begin{pmatrix} m_{i,1,1} & m_{i,1,2} \\ m_{i,2,1} & m_{i,2,2} \end{pmatrix} \tag{5.27}$$

where $m_{\perp,i}$ is the upper or lower 2×2 block diagonal matrix obtained from the block diagonalization of Edward and Teng. With simple math, the value of the α and β functions at the interaction point where the calculation is being done can be obtained. The eigentunes $\theta_i = 2\pi Q_i$ can be directly obtained from Eq. 5.27,

$$\cos\theta_i = \frac{1}{2}\mathrm{Tr}(m_{\perp,i}), \quad \cos\theta_i = \mathrm{sign}(m_{i,1,2})\sqrt{-m_{i,1,2}m_{i,2,1} - \left(\frac{1}{2}\mathrm{Tr}(m_{\perp,i})\right)^2} \tag{5.28}$$

and will allow to compute the tunes of both modes e and f in the presence of coupling.

* * *

5.43 Lnrcoup

General description. Computes the global coupling using the minimum tune approach.	
Detailed description	Computes the global coupling using the minimum tune approach [4]. $$\Delta Q_{min} = \frac{\sqrt{\lvert C+\bar{B}\rvert}}{\pi(\sin(2\pi Q_e)+\sin(2\pi Q_f))} \quad (5.29)$$ with Q_e and Q_f the first and second coupled tunes extracted from the upper block diagonal 2×2 matrix and the lower one respectively.
Inputs	*deth*: Float. Determinant of the h matrix, i.e, $H = C+\bar{B}$. *qx*: Float. First mode coupled tune. *qy*: Float. Second mode coupled tune.
Outputs	*dqmin*: Float. Global linear coupling.

* * *

5.44 Selipnum

General description. Gives the number of one side parasitic interactions of each IP, depending on which pit the IP is closer to.	
Detailed description	Gives the number of one side parasitic interactions of each Interaction Point, depending on which pit the IP is closer to, i.e, if the requested pit is associated to IP5, returns the number of one side parasitic interactions of IP5.
Inputs	*npar*: Integer(8). Number of one side parasitic interactions per IP. *mcol*: Integer. Maximum number of collision points. *mdslt*: Integer. Half buckets. *k*: Integer. Number of the interaction point.
Outputs	*output*: Integer. Number of one side parasitic interactions.

* * *

5.45 Symcheck

General description. Checks if the input matrix *a* is symplectic.	
Detailed description	Checks if the input matrix *a* is symplectic relying on the symplectic condition $$M^T SM = S \tag{5.30}$$ where M is a sector map, T stands for the transpose of the map and $$S = \begin{pmatrix} S_2 & & \\ & S_2 & \\ & & \ddots \end{pmatrix} \tag{5.31}$$ and $S_2 = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$.
Inputs	*a*: Float(n,n). Input squared map. *n*: Integer. Dimension of the input matrix.
Outputs	*maxtsa*: Float. Maximum deviation with respect to a symplectic behavior.
Dependencies	Uses the functions *M44DET* and *M66DET*. Uses the subroutine *mxmpy*.

* * *

5.46 Matrzero

General description. Generates a 3 dimensional map of zeros.	
Detailed description	Generates a 3 dimensional map of zeros.
Inputs	*a_in*: Float(a_count1,a_count2,a_count3). Input matrix. *a_count1*: Integer. First dimension of *a_in*. *a_count2*: Integer. Second dimension of *a_in*.

	a_count3: Integer. Third dimension of *a_in*.
Outputs	*a_in*: Float(a_count1,a_count2,a_count3). Initialized matrix of zeros.

* * *

5.47 Matbar

General description. Computes the symplectic conjugate of a symplectic matrix.

Detailed description	Computes the symplectic conjugate of a symplectic matrix $$\bar{A} = -SA^TS \quad (A \in Sp(2n,R) \leftrightarrow \bar{A} = A^{-1}) \qquad (5.32)$$ with $$S = \begin{pmatrix} S_2 & & \\ & S_2 & \\ & & \ddots \end{pmatrix} \qquad (5.33)$$ and $S_2 = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$.
Inputs	*a*: Float(n,n). Symplectic squared matrix. *n*: Integer. Dimension of a, where n is even.
Outputs	*abar*: Float(n,n). symplectic conjugate of *a*.
Dependencies	Uses the subroutine *mxmpy*.

* * *

5.48 Getmaskmap

General description. Creates a mask indicating where are the elements presents in the input list *name*.

Detailed description	Creates a mask of length *nmap* ($nmap > nname$) signaling the order of appearance of an element of *name* in *mapname*. If the element is just present on *mapname*, *mask* presents a 0.
Inputs	*name*: String(nname). List of names. *mapname*: String(nmap). List of names. *nname*: Integer. Number of elements in *name* *nmap*: Integer. Number of elements in *mapname*
Outputs	*mask*: Integer(nmap). Mask.
Dependencies	uses the function *lastnb*.

* * *

5.49 Upper

General description. Converts all input characters in upper case.	
Detailed description	Converts all input characters in upper case. Written by H. Grote.
Inputs	*sl*: String. String in SL.
Outputs	*sl*: String. String in upper case.

* * *

Chapter 6

Functions

The list of functions that are used in the subroutines of TRAIN are detailed. Since a sequential order does not exist, the functions haven't been sorted.

* * *

6.1 Nxline

General description. Returns the next non-commented line of an input file.	
Detailed description	Reads sequentially each line of the file input while the lines don't start with "#". Each call of nxline returns the next non-commented line of the input file.
Output	*nxline*: String(120). Next line of input file.
Dependencies	Uses the function *lastnb*.

* * *

6.2 Lastnb

General description. Returns the position of the last non-blank character	
Detailed description	Given an input string, returns the position of the last non-blank character.
Input	*string*: String. Character to be used.
Output	*lastnb*: Integer. Length of *string*.

* * *

6.3 Get_tokens

General description. Gets the number of elements on *string* and separate the string into a list of strings in *tokens*.	
Detailed description	Gets the number of elements on *string* and separate the string into a list of strings in *tokens*. In more detail, for each character in *string*, looks for the same character in *skip*. If the character of *string* doesn't appear in *skip* and it is the first character of *string* or if it appears after a character that is in *skip*, a unit is added to get_token. The characters of *string* that don't appear in *skip* are added to *tokens*
Input	*length*: Integer. Maximum length of *string*. *string*: String. Input string. *skip*: String to skip.
Output	*tokens*: String. List of separated elements. *get_tokens*: Integer. Number of elements in *tokens*.

* * *

6.4 Headon

General description. Returns the number of the IP.	
Detailed description	If the input variable *name* is equal to the string *'MKIP'* plus an integer number, it returns the number corresponding to the interaction point (IP). Otherwise, it returns 0.
Input	*name*: String. Marker of the element.
Output	*headon*: Integer. Number of the interaction point.
Dependencies	Uses the function *lastnb*.

* * *

6.5 Parasit

General description. Identifies the long-range interactions.	
Detailed description	The input variables *name* and *nbeam* refer to a MADX file marker and to the number of the beam (either if it refers to beam 1 or beam 2). If *name* starts by *'MKIP'* and its length is superior to 5 , returns 1 in the case of beam 2 and the number of parasitic encounters in case of beam 1. It checks if the marker *name* is in the list *para_names*. If it is already on the list, returns the current number of parasitic encounters, otherwise, it appends the marker and adds one to the number of parasitic encounters.
Input	*name*: String. marker to be checked. *nbeam*: Integer. Index of the beam (1 or 2).
Output	*parasit*: Integer. Number of the parasitic interaction.
Dependencies	Uses the functions *lastnb* and *nampos*.

* * *

6.6 Nampos

General description. Returns the index of the list where a certain input element is.	
Detailed description	Given the name to be looked for *sname*, the name list *slist* and the number of names in the list *nlist*, returns the position of the list in which *sname* is and 0 if *sname* is not found. If *sname* appears more than once, the index of the last appearance is given (for instance, in the case *sname* = 'a', *slist*= 'a,b,c,a' and *nlist* = 4, 4 would be returned).
Input	*sname*: String. Name to be looked up. *slist*: String array. List of names. *nlist*: Integer. Number of elements in *slist*
Output	*nampos*: Position of *sname* in *slist*.

* * *

6.7 Mylist

General description. Finds a given number in an ascending sorted list with binary search.	
Detailed description	Finds number *kelem* in ascending sorted list *klist* with binary search, given the length of the table *nlist*. It returns 0 if the number is not in the table and the position in the table otherwise.
Input	*kelem*: Integer. Number to be looked up. *nlist*: String array. Length of the table. *klist*: Integer array. table of values.
Output	*mylist*: Position of *kelem* in *klist*.

* * *

6.8 Pit_number

General description. Indicates if the input pit number is an Interaction Point.	
Detailed description	Logical function. If the input parameter *n* corresponds to the number of the interaction point where there is an interaction, it returns true. It returns false otherwise.
Input	*n*: Integer. Interaction point number.
Output	*pit_number*: Boolean. True if the input number corresponds to a pit.

* * *

6.9 In_list

General description. Boolean function that returns True if any value in the input two dimensional array is repeated with the same first index.	
Detailed description	Logical function. Returns *true* if the value in *list* with second index *n* is repeated with the same first index. Returns *false* otherwise.
Input	*mcol*: Integer. One of the dimensions of *list*. *n*: Integer. Second index of *list*. *cnt*: Integer. Limit for the first index. *list*: Integer(mcol,mcol+1). List of values where it is seek if a number is repeated. *neq*: Integer. Number of elements in *leq*. *leq*: Integer array. List of second index of *list*.
Output	*in_list*: Boolean. True if any value in *list* is repeated with the same first index.

* * *

6.10 Bc_fun

General description. Uses a function in the definition of the beam currents.	
Detailed description	If the input option *b_curr* is two it uses these function in order to introduce a behavior in the beam current given by a function. It is important to notice that the function is currently commented, so no value is returned using this function. The definition of the function is left to the user.
Input	*x*: Float. Input variable of the function.
Output	*bc_fun*: Float. Bunch current given by a function defined by the user.

* * *

6.11 Sigx

General description. Computes the horizontal beam size.	
Detailed description	Computes the horizontal beam size $\sigma_x = \sqrt{\epsilon_x \beta_x + \left(\frac{\sigma_E}{E}{}_x d_x\right)^2}$, with ϵ_x and β_x the horizontal bunch length and the beta star, $\frac{\sigma_E}{E}$ the relative energy spread and d_x the horizontal dispersion extracted from the Twiss input table.
Input	*bunchnum*: Integer. Number of the bunch. *ipnt*: Integer. Interaction point number. *beamnum*: Integer. Number of the beam (1 or 2).
Output	*sigx*: Float. Horizontal beam size.

* * *

6.12 Sigy

General description. Computes the vertical beam size.	
Detailed description	Computes the vertical beam size $\sigma_y = \sqrt{\epsilon_y \beta_y + \left(\frac{\sigma_E}{E}{}_y d_y\right)^2}$, with ϵ_y and β_y the horizontal bunch length and the beta star, $\frac{\sigma_E}{E}$ the relative energy spread and d_y the horizontal dispersion extracted from the Twiss input table.
Input	*bunchnum*: Integer. Number of the bunch. *ipnt*: Integer. Interaction point number. *beamnum*: Integer. Number of the beam (1 or 2).
Output	*sigy*: Float. Vertical beam size.

* * *

6.13 Outbunch

General description. Returns the index of the list of output bunches.

Detailed description	If *outbcnt* is greater than 0, returns the number of the index of *outblist* where the out *bunch* is. Otherwise returns 0.
Input	*bunch*: Integer. Bunch number.
Output	*outbunch*: Integer. Index of *bunch* in *outblist* or 0.

* * *

6.14 Rg32cut

General description. Returns a real random value smaller than *cut*.	
Detailed description	Returns a real random value smaller than *cut*. The random seed is not initialized.
Input	*cut*: Real. Maximum value that the random number generator can provide.
Output	*rg32cut*: Real. Random value smaller than *cut* that is returned.

* * *

6.15 Rotate

General description. Chooses as reference the leftmost parasitic interaction of the first interaction point appearing in the maps files.	
Detailed description	Chooses as reference the leftmost parasitic interaction of the first interaction point appearing in the maps files. This way we can match the interaction number associated to the optical functions and the maps without the need of always introducing IP5.
Inputs	*ipos*: Integer(8). Array containing the position of the IP in half buckets.
Outputs	*rotate*: Integer. Slot (half bucket) of the leftmost parasitic interaction appearing in the map files counting from IP3.

* * *

6.16 Eqvoptmap

General description. Finds the equivalence between the index of the optic tables and of the map ones when there are extra elements in the maps	
Detailed description	Finds the equivalence between the index of the optic tables and of the map ones when there are extra elements in the maps (the number of maps and the number of interaction points with optical functions are different).
Inputs	*nopt*: Integer. Index of element in Twisss tables.
Outputs	*eqvoptmap*: Integer.Index of element in map tables.

* * *

6.17 M44DET

General description. Computes the determinant of a 4×4 matrix.	
Detailed description	Computes the determinant of a 4×4 matrix.
Inputs	*A*: Float(4,4). Input matrix.
Outputs	*M44DET*: Float. Determinant of A.

* * *

6.18 M66DET

General description. Computes the determinant of a 6×6 matrix.	
Detailed description	Computes the determinant of a 6×6 matrix.
Inputs	*A*: Float(6,6). Input matrix.
Outputs	*M66DET*: Float. Determinant of A.

* * *

6.19 Signdet

General description. Returns the sign of the input value.	
Detailed description	Returns 1 if the input value *det* is positive and -1 otherwise.
Inputs	*det*: Float. Input value.
Outputs	*signdet*: Integer. Sign of *det*.

* * *

6.20 Nstart

General description. Returns the number of irregular bunches.	
Detailed description	Returns the number of irregular bunches within the filling schemes, i.e., those bunches that are present in the filling scheme of B1 and of B2 but in a different bucket.
Outputs	*Nstart*: Integer. Number of irregular bunches.

* * *

CHAPTER 7

ASSOCIATED SCRIPTS AND PROGRAMS

7.1 Bash scripts of A. Gorzawski

In this section the bash scripts developed by A. Gorzawski are detailed.

General description.	
updateMadFiles.sh	Checks if the input file *LHC_$version$.mad* is in the directory *MAD_PART*, changes to the *MAD_PART* directory and executes the input file with the MAD-X version generated by M. Hostettler (corrected bug in sectormap). The input file *LHC_$version$.mad* calls the files *commonBeamBeamPart.mad* and *collisionPath*, which generate the *twiss*, *map* and *survey* files *train.optf*, *train.optb*, *train.manf*, *train.manb*, *train.surf* and *train.surb*. Goes to the main directory and remove the old files with the same name, and copy the newly generated files to the current folder. Copies the files *train.mapf* and *train.mapb* into *opt_train_$version$.manf* and *opt_train_$version$.manb* and the files *train.mapf* and *train.mapb* into *opt_train_$version$.manf* and *opt_train_$version$.manb*. Proper symbolic links are made to the current chosen version through the execution of *./updateLinkToOpticsFile.sh $version$* bash script.

updateLinkToOpticsFile.sh	Checks that the file *opt_train_$version$.optf* is in the current directory, remove the files *train.mapf*, *train.optf*, *train.mapb* and *train.optb*. Erase the lines starting by *MKIP* of the input file and creates these symbolic links to *opt_train_$version$.optf*, *opt_train_$version$.optb*, *opt_train_$version$.manf* and *opt_train_$version$.manb*
runTrainForFillingScheme.sh	If the option *-h* is selected, the help information is displayed. Creates the directory *RESULTS* and generates the *setup.input* file. Runs *./runTrainWithInputFile.sh* with the name of the results directory.
runTrainWithInputFile.sh	If the option *-h* is selected, the help information is displayed. List all the files starting with *train.* and execute *./ltrain_new* with the input file *setup.input*. Moves the results to the created folder in *RESULTS*. If the input option 'plot' is selected, executes the file *plotVerHorOffset.py* with python and the input files in the $results$ directory.
plotVerHorOffsets.py	Plots the horizontal and vertical offset in IP1 and IP5.

* * *

7.2 Filling Schemes generator

The python script *generateFillingScheme.py* generates a filling scheme on the format of 5 columns that TRAIN accepts. It can be used executing

```
$ python generateFillingScheme.py {Input.csv}{Output.in}
```

where $Input.csv$ refers to a csv file containing the information of the injection chain, for example as in [1][1], and $Output.in$ is the name of the filling scheme in the format accepted by TRAIN.

* * *

[1]https://espace.cern.ch/HiLumi/WP2/Shared\%20Documents/Filling\%20Schemes\%20HL-LHC/25ns_2760b_2748_2494_2572_288bpi_13inj.csv

Bibliography

[1] S. Antipov, F. Antoniou, R. Appleby, G. Arduini, J. Barranco, P. Baudrenghien, N. Biancacci, C. Bracco, R. Bruce, X. Buffat, R. Calaga, L.R. Carver, M. Crouch, R. De Maria, S. Fartoukh, D. Gamba, M. Giovannozzi, P. Goncalves Jorge, W. Höfle, G. Iadarola, N. Karastathis, A. Lasheen, K. Li, T. Mastoridis, L. Medina, A. Mereghetti, E. Métral, D. Mirarchi, B. Muratori, S. Papadopoulou, Y. Papaphilippou, D. Pellegrini, T. Pieloni, S. Redaelli, G. Rumolo, B. Salvant, E. Shaposhnikova, M. Solfaroli-Camillocci, C. Tambasco, R. Tomás, D. Valuch, *Update of the HL-LHC operational scenarios for proton operation*, CERN, Geneva, Switzerland, CERN-ATS-Note-2018, 2018.

[2] D. A. Edwards and L.C.Teng, *Parametrization of linear coupled motion in periodic systems*, National Accelerator Laboratory, Batavia, Illinois, 1973.

[3] D. Sagan and D.Rubin, *Linear analysis of coupled lattices*, Willson Laboratory, Cornell University, Ithaca, New York, 1999.

[4] Wolfram Fischer, *Robust linear coupling correction with N-turn maps*, Brookhaven National Laboratory, Upton, New York, 2003.

[5] L. Deniau, H. Grote, G. Roy and F. Schmidt, *The MAD-X Program. User's reference Manual*, CERN, Geneva, Switzerland,`http://mad.web.cern.ch/mad/`, 2017.