Bachelor's Degree in
Computational Mathematics

End-of-Degree Project

Post-Processing Routines for Long-Term Beam Stability in Accelerators

---

Author:
Xavier Valls Pla

Supervisors:
Riccardo De Maria (CERN),
José Antonio López Ortí (UJI)

Reading date: 27th of November 2015
Academic course 2014/2015

I certify that this work has been carried out by the student Xavier Valls Pla, under my supervision, and is presented to constitute an End-of-Degree Project for the Bachelor's Degree in Computational Mathematics.

Signed: José Antonio López Ortí.

Summary

Accelerator complexes require the collaboration of thousands of scientists and engineers to be able to build wonders like the Large Hadron Collider (LHC) at CERN and study all the information obtained with its operation.

One of the key aspects for the operation of accelerators is the study of charged-particle beam optics and dynamics. For instance, the simulation of single-particle trajectories is used to predict and optimize the long-term stability of particle beams inside an accelerator, which can take several hours or about 400 billion revolutions for the LHC.

SixTrack is one of the codes used for simulating single-particle dynamics for the LHC and other accelerators. It has been in development for the past 25 years at CERN together with an ecosystem of pre- and post-processing tools (SixDesk) to prepare, launch and process the results of the simulations.

This project contributes to SixTrack clarifying the theoretical foundations of the physics behind the post-processing, providing a reference for its code and rewriting part of SixDesk's post-processing routines with modern libraries and re-optimized algorithms together with a critical analysis of the decisions taken in the design process.

The results consist in an improvement in term of analysis speed, a simplification of the code structure, a thorough documentation of the Fortran post-processing subroutines, the implementation of new features and an improvement in the exposition of the mathematical and physical methods behind the post-processing routines.

Keywords

SixTrack, beam dynamics, accelerator physics, particle tracking, Python.

Contents

List of Tables

List of Figures

Chapter 1

Introduction and Motivation

Accelerators have become a key tool for the study of experimental particle physics. Today, thousand of physicists around the world depend on the results generated by the Large Hadron Collider.

Having such a big amount of people relying on the results of one of the most complex works of engineering ever created demands an extraordinary precision in the computations and tuning of the path the beam will follow. For that purpose, simulators like SixTrack were created.

This chapter introduces CERN as an accelerator complex (Section 1.1), describes what is hoped to achieve by tracking particles (Section 1.2) and gives an overview (Section 1.3) of SixTrack as a particle-tracking simulator, its SixDesk run environment and the distributed voluntary-based computing platform LHC@Home.

Then, Section 1.4 will outline the problem to attack and the motivation behind this project will be explained in Section 1.5, followed by a description of the method applied to approach this project in Section 1.6. Finally, a brief description of the structure of this dissertation is given in Section 1.7.

1.1 CERN

A particle accelerator is a machine built out of a large number of magnets and electromagnetic devices. This devices are the ones in charge of guiding the beam through the path described by the accelerator.

CERN is a complex of accelerators created as an intergovernmental organization in 1954 in order to spark the study of particle physics making use of the several linear and circular accelerators built with the collaboration of the 21 state members and the associate state members.

Not all accelerators are equals or have the same purpose. For example, at CERN, linear accelerators (LINAC2, LINAC3) are mainly used to inject the beam into the circular ones, and some of this circular accelerators require a previous pass through other smaller circular accelerators just to be sure that the beam is injected at the right speed.

There are two beams at a time circulating on this chain of accelerators, each beam consisting of a large number of bunches of the order of 10^{11} protons. For the Large Hadron Collider (LHC) the chain starts extracting protons from hydrogen gas and injecting them into the LINAC2. LINAC2 will accelerate the protons to 50 MeV and then inject them to the PS Booster, which will bring them up to 1.4 GeV.Then both beams are injected into the PS (Proton Synchrotron) and accelerated to 25 GeV to the last link of the chain (and the second biggest accelerator at CERN) before being injected to the LHC, the SPS (Super Proton Synchrotron), where they will reach the required injection energy for the LHC, 460 GeV. Finally, in the LHC the beams will be able to reach up to 7 TeV of Energy.

A schematic drawing of the CERN accelerator complex, including the LHC chain, can be observed in Figure 1.1.

At CERN, particle physics are studied in several ways, being the study of the optics and beam dynamics one of the most central parts of it. This is done, for instance, tracking particles in the beam and observing them at the same point of the synchrotron for each turn.

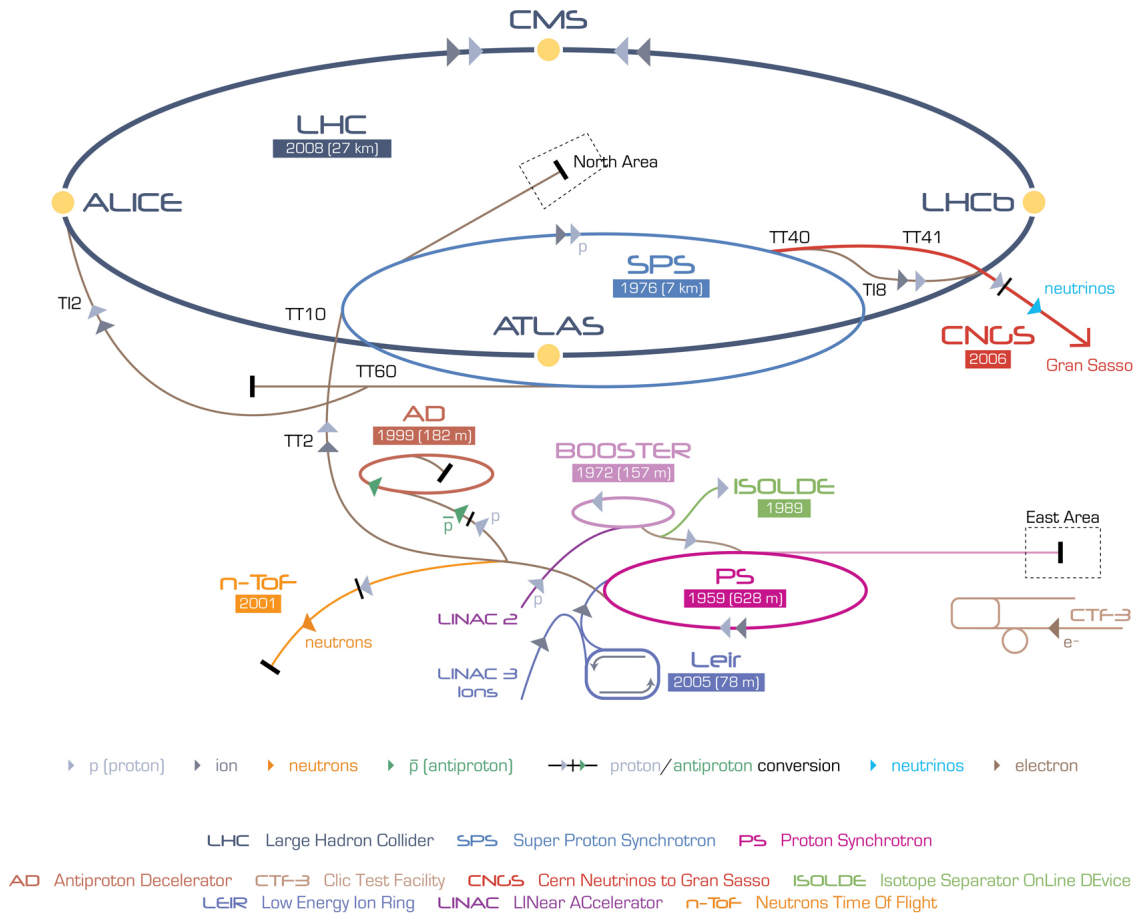


Figure 1.1: The CERN accelerator complex, including the LHC chain. The yellow dots in the LHC represent each one of the four big experiments.

1.2 Beam Dynamics and Particle Tracking

Most of the effects and beam dynamics in an accelerator can be observed by tracking a single particle instead of a whole bunch of them inside an electromagnetic field that represents a combination of the diverse elements that conform the accelerator: dipoles, quadrupoles, drift spaces...

We are able to define the equations of motion for each one of this devices and simplifying them into a one-turn symplectic matrix, making good use of the properties of Hamiltonians to describe the electromagnetic field of the circular accelerator.

Particle tracking is useful, for example, for studying the dynamic aperture or the long-term stability of the beam, and because of this, particle tracking codes aimed at different studies, configurations and accelerators started to appear. One of the long-lived ones is SixTrack.

1.3 SixTrack and the LHC@Home Platform

Accelerators are incredibly complex machines that require a huge amount of energy and time to make them run, and the slightest deviation on the calculus of the optics or the energy needed, or even an error in its setup (not achieving the void, a loose screw, millimeters of difference between the platforms it stands on..) can mean months of delay for its operation.

At the same time, the energy required and produced, and the limitations of what the materials can stand, suppose that the operation of the LHC can only be available during a limited amount of time. This requires physicists to make a selection from all the tests they would ideally like to run given infinite time and, knowing that they will have a fixed time window to do their tests, to tune the beam as precisely as they can to obtain their desired results.

In this context, simulation codes have become an essential tool. SixTrack, the simulator this project is about, is a six-dimensional particle tracking code aimed at analyzing single-particle effects in circular accelerators.

As said before, accelerators are too complex that even simulators like Sixtrack depend on the flexibility of the parameters used to tune the simulations and studies

to run on them. This flexibility makes this tools really powerful but, at the same time, too complex to use. For the sake of simplicity, a run environment for SixTrack called SixDesk [11] was developed, hiding most of the complications. The user then would only need to tune a small set of the parameters in a specific file that will be parsed and feed into SixTrack for running a study.

But as technology advanced other bottlenecks appeared. The construction of new, bigger and more complex accelerators and the continuous growth of computer-aided research fueled a scientific-advance never seen before. Soon the existing processing power of traditional computers was not enough for the biggest simulations, and the demand for more computational power continued to soar.

In this situation, the search for new ways to acquire that greatly needed computational power lead to the implementation of the LHC@Home platform [7]: a volunteer computing system for massive numerical simulations of beam dynamics and high energy physics events, where volunteers from all around the world, currently more than 125.000 registered users, help running beam dynamics simulation by contributing spare processing capacity on their home and laptop computers. SixTrack was the first project to make use of distributed computing this way at CERN, achieving a ten to hundredfold increase in computing capacity by implementing and making use of the BOINC [1] libraries, which will split the work to do from the binaries and sent each one of the pieces to the volunteer computers for its execution when they are idle.

1.4 Problem

SixTrack and SixDesk's code is a complex mixture of Fortran 77, C and various shell scripts and utilities. SixTrack's core code itself has more than 70.000 lines of Fortran code and a complex structure of source files managed by a preprocessor.

The evolution of computational power and the rise of projects like the LHC@Home platform have empowered SixTrack's users to a point where the maximum number of entries under a folder limitation of the distributed file system used at CERN, the Andrew File System (AFS) [8], has been reached quite frequently. To overcome this limitations, a Python database-centered port of SixDesk has been taken into development during the past year.

The underlying physics behind Sixtrack have been traditionally not documented

and, while there has been some remarkable improvement on this lately [4], its operation required some background using similar tools and advanced knowledge of how the physics models are usually implemented in this kind of programs.

This project is mainly centered on the different SixTrack post-processing procedures, and will involve documenting the Fortran subroutines, porting the post-processing scripts to Python and integrating them in the new SixDesk database port.

1.5 Motivation

This project aims at making SixTrack's development easier, fast and more understandable by improving its physics documentation and providing an understandable description of the analysis and computations made on its post-processing subroutines.

The Python database-centered implementation of SixTrack will provide a faster way to manage, obtain and display the results of the post-processing runs, providing an interface for the results obtained instead of looking at the files. It also attempts to stop the proliferation of SixTrack's private branches, making the code more modular, using a friendlier language, improving execution times and, most of all, eliminating its reliance on a complex structure of interdependent Bash and Fortran scripts.

1.6 Method

The project will start documenting the basics of the beam dynamics parametrization in 6 dimensions. A thorough walk through the post-processing subroutines will follow, extracting from it all the theory, formulas and the process they follow.

Then, the `run_post` script and all its interdependent Bash and Fortran code for long runs will be adapted and ported to the new SixDesk's database-centered implementation, guaranteeing the replication of the Fortran-generated files and ensuring numerical reproducibility.

Finally, SixDesk's port will be extended with an on-demand interface to obtain

the plots generated by `run_post` script.

1.7 Project's Structure

A brief overview of the chapters in this project:

- Chapter 2 introduces the relevant theoretical framework for this dissertation and lays the ground for the implementation. It will introduce topics like Hamiltonian mechanics, symplectic integration, dynamic stabilities and the contribution made to the SixTrack physics manual, the theoretical 6-D parameterization.
- Chapter 3 gives an introduction to SixTrack and particle tracking describing its purpose, putting a spotlight on the Post-processing computations. Then, it walks the reader through its post-processing subroutine.
- Chapter 4 describes the work done implementing the proposed improvements to SixDesk's database, describing the original and final structures and displaying the results.
- Chapter 5 exposes the conclusions of this project and reflects on its impact in Sixtrack's future development.

Chapter 2

Theory Foundations

This Chapter familiarizes the reader with the theory foundations used to program SixTrack and its post-processing subroutines.

Section 2.1 describes the reference coordinate system employed to parameterize the particle motion. Section 2.2 describes the three alternate approaches for classical mechanics used in beam dynamics and applies them to describe the motion of a particle in an accelerator.

Finally, Section 2.3 dives into the more advanced topic of Beam Optics, introducing the components of an accelerator and walking the reader through the process followed to parameterize the motion in six dimensions and the description of the phase-space and the symplectic condition as the key concepts behind it.

2.1 Coordinate System

While describing the motion of a particle inside an accelerator, it's convenient to carry out a change of coordinates to a Frenet-Serret coordinate system like the one shown in Figure 2.1.

The coordinate system moves with the particle along a reference trajectory defined by an ideal particle. This trajectory, the **design orbit**, represents the ideal closed orbit a reference particle with constant energy follows in a uniform and con-

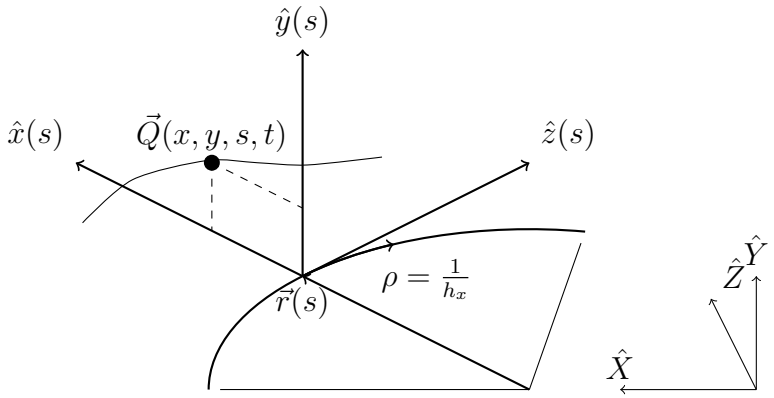


Figure 2.1: Moving reference frame $(\hat{x}, \hat{y}, \hat{z})$ parameterized by $s(t)$. The trajectory of a particle Q can be described by the coordinates (x, y, s, t).

stant magnetic field. That means an ideal particle would circulate in the machine forever.

Assuming the design orbit exists, it is referenced in the Figure 2.1 by $\vec{r}_0(s)$, where the path length s measures the distance to a chosen origin along the design orbit.This path is described by $\vec{Q}(x, y, s, t)$ where x and y are transverse coordinates specified to the design orbit.

In this ideal orbit, the bending radius $\rho(s)$ and its inverse $h(s)$ remain constant, and the inverse of the bending radius will be denoted as h_x and h_y depending on which plane the bending is performed.

To describe the trajectory of the particle relative to the design orbit, three unit vectors are used: the tangent vector $\vec{e}_s$, the unit normal vector $\vec{e}_N$ and the unit binormal vector $\vec{e}_B$. However, is convenient to define two new unit vectors combination of the last two:

$$\begin{aligned} \vec{e}_x(s) &= \begin{cases} +\vec{e}_N(s), & \text{orbit in horizontal plane} \\ -\vec{e}_B(s), & \text{orbit in vertical plane} \end{cases} \\ \vec{e}_y(s) &= \begin{cases} +\vec{e}_N(s), & \text{orbit in horizontal plane} \\ +\vec{e}_B(s), & \text{orbit in vertical plane} \end{cases} \end{aligned} \tag{2.1}$$

that leads to $\vec{e}_x(s) \times \vec{e}_y(s) = \vec{e}_s(s)$, what means that $\{\vec{e}_x(s), \vec{e}_y(s), \vec{e}_s(s)\}$ is a right-handed orthonormal system with $\vec{e}_x(s)$ always in the horizontal plane and $\vec{e}_y(s)$ always on the vertical one.

Then, the equation of the movement of the tracked particle can become:

$$\vec{Q}(x, y, s, t) = \vec{r}_0(s(t)) + x \cdot \vec{e}_x(s(t)) + y \cdot \vec{e}_y(s(t)) \tag{2.2}$$

2.2 Hamiltonian Mechanics

In accelerator physics, three alternate approaches to classical mechanics are used: the Newtonian, Lagrangian and Hamiltonian mechanics.They differ on the way of defining a dynamical system in its canonical form: Using the force, the Lagrangian or the Hamiltonian, respectively.

The conventional, classic Newtonian mechanics can be summarized with the equation

$$\sum_i F_i = \frac{d(mv)}{dt}$$

which relates the sum of the forces of a system with the temporal evolution of the mechanical momentum of said system.

For the Lagrangian mechanics approach, the Lagrangian L of a system is defined as

$$L \equiv T - V$$

where T is the kinetic energy of the system, and V is the potential energy.

Given the Lagrangian as the function $L(q_j, \dot{q_j}, t)$, the equations of motion of a system can be expressed as

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_j}\right) - \frac{\partial L}{\partial\ q_j} = 0, \qquad (j = 1, ..., n) \tag{2.3}$$

where q_i are the components of q. This formulation is known as the Euler-Langrange equations of motion.

The third formulation, the Hamiltonian mechanics, is the one used during the rest of the project.

From the same Lagrangian $L(q_j, \dot{q_j}, t)$, the conjugate momenta p_i of the system can be obtained by

$$p_i \equiv \frac{\partial L(q_j, \dot{q_j}, t)}{\partial \dot{q_i}}, \qquad i = 1, ..., n \tag{2.4}$$

The Hamiltonian of a system is defined in terms of a set of coordinates q_i and their corresponding momenta p_i. This coordinate pairs (q_i, p_i) are known as canonical variables, and the Hamiltonian is written

$$H = \sum_i \dot{q}_i p_i - l(q, \dot{q}, t) \tag{2.5}$$

If the forces acting on the system are conservative and the position in space does not explicitly depend on time, the Hamiltonian follows the equation

$$H = T + V \tag{2.6}$$

In this case, the Hamiltonian is an expression for the total energy in the system.

Given the Hamiltonian, the equations of motion for a dynamical system can be expressed as

$$\begin{aligned} \frac{dx}{dt} &= +\frac{\partial H}{\partial p_x}, & \frac{dy}{dt} &= +\frac{\partial H}{\partial p_y}, & \frac{dz}{dt} &= +\frac{\partial H}{\partial p_z}, \\ \frac{dp_x}{dt} &= -\frac{\partial H}{\partial x}, & \frac{dp_y}{dt} &= -\frac{\partial H}{\partial y}, & \frac{dp_z}{dt} &= -\frac{\partial H}{\partial z} \end{aligned} \tag{2.7}$$

Canonical transformation
In Hamiltonian mechanics, a canonical transformation is a mapping from one set of canonical coordinates to another preserving the form of Hamilton's equation.

This transformation is usually performed to simplify the problem at hand (For example, map from cartesian coordinates to polar coordinates for phase-space analysis). In this case, the transformation $(p_i, q_i) \rightarrow (Q_i, P_i)$ performed is written

$$Q_i = Q_i(q, p, t) P_i = P_i(q, p, t), \qquad i = 1, ..., N \tag{2.8}$$

where p and q are the old sets of canonical coordinates.

This mapping leads to express the new canonical coordinates (Q_i, P_i) with the new Hamiltonian K as

$$\begin{aligned} \frac{dQ_i}{dt} &= +\frac{\partial K}{\partial P_i}, \\ \frac{dP_i}{dt} &= -\frac{\partial K}{\partial Q_i} \end{aligned} \tag{2.9}$$

2.2.1 Particle Motion in an Accelerator

The motion of a charged relativistic particle in the electromagnetic field **E** and **B** is governed by the Lorentz Force:

$$\mathbf{F} + q(\mathbf{E} + \mathbf{v} \times \mathbf{B}), \tag{2.10}$$

being the electric field **E** and the magnetic field **B** related to the electromagnetic scalar potential ϕ and the electromagnetic vector potential **A** as

$$\begin{aligned} \mathbf{E} &= -\nabla\phi - \frac{\partial \mathbf{A}}{\partial t}, \\ \mathbf{B} &= \nabla \times \mathbf{A} \end{aligned} \tag{2.11}$$

The Lagrangian is expressed as follows [9]:

$$L = -m_0 c^2 \sqrt{1 - \frac{|\mathbf{v}|^2}{c^2}} - q\phi + q\mathbf{v} \cdot \mathbf{A} \tag{2.12}$$

and deriving from the equation 2.5 we obtain the Hamiltonian

$$\begin{aligned} &H \equiv H(x, p_x, y, p_y, \sigma, p_\sigma; s), \\ &H = p_\sigma - (1 + h_x x)\left(\sqrt{(1+\delta)^2 - (p_x - a_x)^2 - (p_y - a_y)^2} + a_s\right) \end{aligned} \tag{2.13}$$

where h_x describes the horizontal inverse bending radius, $\delta \equiv \delta(p_\sigma)$ is the momentum deviation respect to the reference particle and a_i represents the components of the electromagnetic vector potential normalized :

$$a_x = \frac{q}{P_0} A_x, \qquad a_y = \frac{q}{P_0} A_y, \qquad a_s = \frac{q}{P_0} A_s \tag{2.14}$$

The canonical momentum (p_x, p_y) is given by

$$\begin{aligned} p_x &= \frac{1}{P_0}\left(\frac{mv_x}{\sqrt{1-\frac{|v|^2}{c^2}}} + qA_x\right), \\ p_y &= \frac{1}{P_0}\left(\frac{mv_y}{\sqrt{1-\frac{|v|^2}{c^2}}} + qA_y\right) \end{aligned} \tag{2.15}$$

The longitudinal coordinates (σ, p_σ) are defined by:

$$\begin{aligned} \sigma &= s - \beta_0 ct, \\ p_\sigma &= \frac{1}{\beta_0}\frac{E-E_0}{P_0 c} \end{aligned} \tag{2.16}$$

With this canonical variables and its Hamiltonian, the following Hamilton's equations can be defined:

$$\begin{aligned} \frac{dx}{ds} &= +\frac{\partial H}{\partial p_x}, & \frac{dy}{ds} &= +\frac{\partial H}{\partial p_y}, & \frac{d\sigma}{ds} &= +\frac{\partial H}{\partial p_\sigma}, \\ \frac{dp_x}{ds} &= -\frac{\partial H}{\partial x}, & \frac{dp_y}{ds} &= -\frac{\partial H}{\partial y}, & \frac{dp_\sigma}{ds} &= -\frac{\partial H}{\partial \sigma} \end{aligned} \tag{2.17}$$

2.3 Beam Optics

The motion of the particle inside an accelerator can be described in terms of the set of coordinates $z = \{x, x', y, y', \sigma, \delta\}$, where $\sigma = s - v_0 \times t$ is the path length and $\delta = \frac{\Delta p - p_0}{p_0}$ is the relative momentum deviation from the design orbit's momentum p_0.

2.3.1 The Symplectic Integrator

A geometrical integrator maps or transform a given set of coordinates to a new one by means of canonical transformations. A symplectic integrator is a geometrical numerical integrator of differential equations, and the symplectic condition is stated as follows:

$$\mathbf{M}^T\mathbf{S}\mathbf{M} = \mathbf{S}, \tag{2.18}$$

where

$$\mathbf{M} = \begin{bmatrix} \frac{\partial X_1}{\partial x_1} & \frac{\partial X_1}{\partial p_1} & \cdots & \frac{\partial X_1}{\partial x_N} & \frac{\partial X_1}{\partial p_N} \\ \frac{\partial X_2}{\partial x_1} & \frac{\partial X_2}{\partial p_1} & \cdots & \frac{\partial X_2}{\partial x_N} & \frac{\partial X_2}{\partial p_N} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{\partial X_N}{\partial x_1} & \frac{\partial X_N}{\partial p_1} & \cdots & \frac{\partial X_N}{\partial x_N} & \frac{\partial X_N}{\partial p_N} \end{bmatrix} \tag{2.19}$$

is the Jacobian matrix of the transformation from the set of coordinates $(x_1, p_1, x_2, p_2, ..., x_N, p_N)$ to the new one $(X_1, P_1, X_2, P_2, ..., X_N, P_N)$ and $\mathbf{S}$ is the symplectic matrix

$$\mathbf{S} = \begin{bmatrix} 0 & 1 & & & \\ -1 & 0 & & & \\ & & \ddots & & \\ & & & 0 & 1 \\ & & & -1 & 0 \end{bmatrix}, \tag{2.20}$$

its form depending on which order the coordinates are defined in.

An important property from this is that for any number of Jacobian matrices $M_1, M_2, ..., M_N$ that satisfy the symplectic condition in Equation 2.18, the product

$$\prod_{i=1}^{N} M_i = M_1 \cdot M_2 \cdots M_n = M_{total} \tag{2.21}$$

also satisfies the symplectic condition.

2.3.2 The Lattice of an Accelerator: Components

An accelerator is a complex machine composed of thousand of smaller devices, mainly magnets. All the magnets fall within the following categories:

- **Dipole**: Magnet used to bend the beam so it follows the path described by the accelerator. In circular accelerators, the bending angle of all the dipoles has to come to 2π.
- **Quadrupole:** Magnet used to focus the beam around the accelerator. Although it focuses the beam in one plane, it the focuses it in the other plane. That's why they are usually bundled in sets combining Focusing and defocusing quadrupoles with drift spaces, meaning by focusing a vertical focusing quadrupole and by defocusing the horizontal focusing one.

- **Sextupoles:** Used for chromaticity compensation.
- **Higher order magnets:** Octupoles, decapoles, dodecapoles...they are used to correct or introduce nonlinear behaviors in the beam.

while other components include:

- **Drift space:** Field-free region of the accelerator, usually located between elements. A particle moving inside a drift space doesn't experience any change in its momentum.
- **Radio Frequency-cavity:** Accelerates and focuses the beam longitudinally to preserve the bunch structure of the beam.

The equations of motion for each one of this devices can be obtained as transfer maps in the form of symplectic matrices that satisfy the symplectic condition stated in the Equation 2.18.

The **one-turn map** of a circular accelerator is the set of functions that relate the initial coordinates of the particle to the final coordinates after one turn. This one turn map for a particular position in the ring can be obtained applying the properties of symplectic matrices as shown in the Equation 2.21.

2.3.3 Transverse Dynamics and Phase-space

In theory, the dipoles in a circular accelerator define an ideal orbit for a particle with the reference momentum p_0. This orbit follows a perfect path through the center of each element and closes itself after one complete turn along the circumference.

An orbit that closes in upon itself is referenced as a **closed orbit**. In practice this dipoles around the ring have errors and, alongside other effects, will distort the real closed orbit from the ideal design orbit and will require the use of quadrupoles for focusing and defocusing.

In a circular accelerator, the focusing due to the quadrupoles is periodic along the path, and therefore the motion of a particle in the transverse plane can be studied

with the linearized Hill's equation

$$z'' \pm k(s)z = 0 \qquad z = x, y \qquad z' = \frac{dz}{ds} \simeq \frac{p_z}{1+\delta}, \tag{2.22}$$

where k(s) is a periodic focusing coefficient determined by the properties of the lattice. This equation can be derived by the solving the Hamilton equation for:

$$H = \frac{1}{2}\frac{(p_x^2 + p_y^2)}{1+\delta} + k(s)\frac{(x^2 - y^2)}{2} \tag{2.23}$$

which is a linear approximation (Equation 2.13) for a pure quadrupole and $k = \frac{\partial Bx}{\partial y} = -\frac{\partial By}{\partial x}$.

The solutions for Hill's equation take the form of

$$z = \sqrt{\beta_z(s)\epsilon_z}\cos(\phi_z(s) + \phi_{z_0}), \qquad z = x, y \tag{2.24}$$

where ϕ is the phase angle and ϵ is a constant of motions called Courant-Snyder invariant (or amplitude) which average over all particles in beam relates to the so-called **beam emittance**, and represents the area a particle occupies in the phase-space.

Under the influence of conservative forces the particle density in the phase space stays constant (Liouville's theorem). The phase ellipse in Figure 2.2 is described by:

$$\gamma(s)z^2 + 2\alpha(s)zz' + \beta(s)z'^2 = \epsilon, \quad z = x, y \tag{2.25}$$

where α, β and γ are called the Courant-Snyder lattice functions. β is the beta-function of the accelerator that describes the variation of the oscillation envelope around the ring, γ describes the envelope of oscillations in x' and y' and both are related by the alpha function

$$\alpha_z = -\frac{1}{2}\frac{d}{ds}\beta_z(s) = \sqrt{\gamma_z(s)\beta_z(s) - 1}, \quad z = x, y \tag{2.26}$$

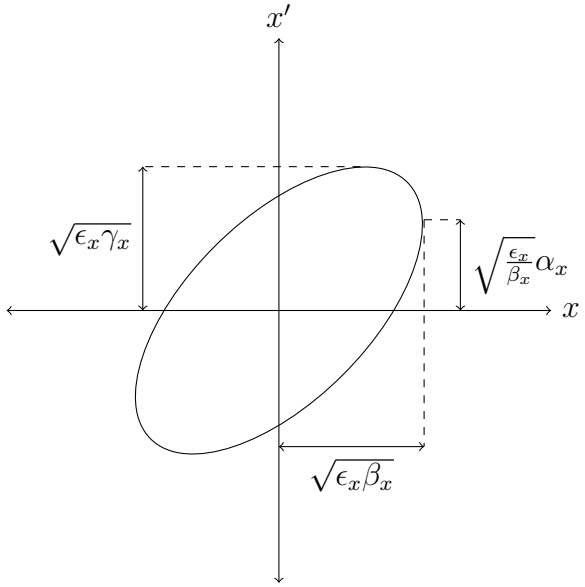


Figure 2.2: Phase space ellipse in the transverse x, x' plane.

2.3.4 Beam Optics Parameterization

The one-turn map M leads to stable motion whenever the eigenvalues $\lambda^{\pm}_{I,II,III} = \exp(\pm i 2\pi Q_{I,II,III})$ are complex and conjugate and the tunes $Q_{I,II,III}$ are real.

The one-turn map can be then decomposed in

$$\mathbf{M} = TRT^{-1} \tag{2.27}$$

where R is

$$R = \begin{pmatrix} \cos(2\pi Q_I) & \sin(2\pi Q_I) & 0 & 0 & 0 & 0 \\ -\sin(2\pi Q_I) & \cos(2\pi Q_I) & 0 & 0 & 0 & 0 \\ 0 & 0 & \cos(2\pi Q_{II}) & \sin(2\pi Q_{II}) & 0 & 0 \\ 0 & 0 & -\sin(2\pi Q_{II}) & \cos(2\pi Q_{II}) & 0 & 0 \\ 0 & 0 & 0 & 0 & \cos(2\pi Q_{III}) & \sin(2\pi Q_{III}) \\ 0 & 0 & 0 & 0 & -\sin(2\pi Q_{III}) & \cos(2\pi Q_{III}) \end{pmatrix}$$

the rotation matrix and

$$T = \begin{pmatrix} x_1 & x'_1 & y_1 & y'_1 & \sigma_1 & \delta_1 \\ x_2 & x'_2 & y_2 & y'_2 & \sigma_2 & \delta_2 \\ x_3 & x'_3 & y_3 & y'_3 & \sigma_3 & \delta_3 \\ x_4 & x'_4 & y_4 & y'_4 & \sigma_4 & \delta_4 \\ x_5 & x'_5 & y_5 & y'_5 & \sigma_5 & \delta_5 \\ x_6 & x'_6 & y_6 & y'_6 & \sigma_6 & \delta_6 \end{pmatrix}$$

has the eigenvectors $\vec{z}_i = (x_i, x'_i, y_i, y'_i, \sigma_i, \delta_i)$ for $i = 1, \ldots, 6$ as rows.

The eigenvectors can be normalized in such a way

$$TST^T = \begin{pmatrix} x_1x'_1 - x_1x'_1 + ... = 0 & x_1x'_2 - x_2x'_1 + ... = 1 & ... \\ x_2x'_1 - x_1x'_2 + ... = -1 & x_2x'_2 - x_2x'_2 + ... = 0 & ... \\ ... & & \end{pmatrix} = S$$

where S is the symplectic form in Eq. 2.20.

The optical properties of a lattice are conveniently described by a set of lattice functions.

For a general six-dimensional case, including coupling, three distinct oscillation modes I, II, III give rise to three sets of lattice functions [16].

Matrix T can be used to extract the coupled lattice functions by the following steps.

A point in the 6-dimensional phase space $\vec{z} = (x, x', y, y', \sigma, \delta)$ can be described by 6 independent vectors $\vec{z}_i = (x_i, x'_i, y_i, y'_i, \sigma_i, \delta_i)$ for $i = 1, \ldots, 6$

$$\begin{aligned} x_1 &= \sqrt{\beta_{xI}(s)}\cos\Phi_{xI}(s) & x_2 &= \sqrt{\beta_{xI}(s)}\sin\Phi_{xI}(s) \\ x'_1 &= \sqrt{\gamma_{xI}(s)}\cos\tilde{\Phi}_{xI}(s) & x'_2 &= \sqrt{\gamma_{xI}(s)}\sin\tilde{\Phi}_{xI}(s) \\ x_3 &= \sqrt{\beta_{xII}(s)}\cos\Phi_{xII}(s) & x_4 &= \sqrt{\beta_{xII}(s)}\sin\Phi_{xII}(s) \\ x'_3 &= \sqrt{\gamma_{xII}(s)}\cos\tilde{\Phi}_{xII}(s) & x'_4 &= \sqrt{\gamma_{xII}(s)}\sin\tilde{\Phi}_{xII}(s) \\ x_5 &= \sqrt{\beta_{xIII}(s)}\cos\Phi_{xIII}(s) & x_6 &= \sqrt{\beta_{xIII}(s)}\sin\Phi_{xIII}(s) \\ x'_5 &= \sqrt{\gamma_{xIII}(s)}\cos\tilde{\Phi}_{xIII}(s) & x'_6 &= \sqrt{\gamma_{xIII}(s)}\sin\tilde{\Phi}_{xIII}(s), \end{aligned}$$

and similarly for the y_i, y'_i, σ_i and δ_i coordinates, using the lattice functions for the vertical and longitudinal planes and

$$\tilde{\Phi}_{yI} = \Phi_{yI} - \arctan(1/\alpha_{yI}) \tag{2.28}$$

$$\vec{z} = \begin{pmatrix} \sqrt{\epsilon_I}\sqrt{\beta_{xI}}\cos(\Phi_{xI}+\phi_I) + \sqrt{\epsilon_{II}}\sqrt{\beta_{xII}}\cos(\Phi_{xII}+\phi_{II}) + \sqrt{\epsilon_{III}}\sqrt{\beta_{xIII}}\cos(\Phi_{xIII}+\phi_{III}) \\ \sqrt{\epsilon_I}\sqrt{\gamma_{xI}}\cos(\tilde{\Phi}_{xI}+\phi_I) + \sqrt{\epsilon_{II}}\sqrt{\gamma_{xII}}\cos(\tilde{\Phi}_{xII}+\phi_{II}) + \sqrt{\epsilon_{III}}\sqrt{\gamma_{xIII}}\cos(\tilde{\Phi}_{xIII}+\phi_{III}) \\ \sqrt{\epsilon_I}\sqrt{\beta_{yI}}\cos(\Phi_{yI}+\phi_I) + \sqrt{\epsilon_{II}}\sqrt{\beta_{yII}}\cos(\Phi_{yII}+\phi_{II}) + \sqrt{\epsilon_{III}}\sqrt{\beta_{yIII}}\cos(\Phi_{yIII}+\phi_{III}) \\ \sqrt{\epsilon_I}\sqrt{\gamma_{yI}}\cos(\tilde{\Phi}_{yI}+\phi_I) + \sqrt{\epsilon_{II}}\sqrt{\gamma_{yII}}\cos(\tilde{\Phi}_{yII}+\phi_{II}) + \sqrt{\epsilon_{III}}\sqrt{\gamma_{yIII}}\cos(\tilde{\Phi}_{yIII}+\phi_{III}) \\ \sqrt{\epsilon_I}\sqrt{\beta_{\sigma I}}\cos(\Phi_{\sigma I}+\phi_I) + \sqrt{\epsilon_{II}}\sqrt{\beta_{\sigma II}}\cos(\Phi_{\sigma II}+\phi_{II}) + \sqrt{\epsilon_{III}}\sqrt{\beta_{\sigma III}}\cos(\Phi_{\sigma III}+\phi_{III}) \\ \sqrt{\epsilon_I}\sqrt{\beta_{\delta I}}\cos(\Phi_{\delta I}+\phi_I) + \sqrt{\epsilon_{II}}\sqrt{\beta_{\delta II}}\cos(\Phi_{\delta II}+\phi_{II}) + \sqrt{\epsilon_{III}}\sqrt{\beta_{\delta III}}\cos(\Phi_{\delta III}+\phi_{III}) \end{pmatrix},$$

where ϕ_I, ϕ_{II} and ϕ_{III} are initial phases for the I , II and III modes, and $\sqrt{\epsilon_I}$, $\sqrt{\epsilon_{II}}$, and $\sqrt{\epsilon_{III}}$ are the amplitudes of mode I, II and III, respectively.

Under this definition the effect of the one-turn map can be also expressed as:

$$\begin{aligned} \vec{z} = &\sqrt{\epsilon_I}\left(\vec{z_1}\cos(\phi_I + 2\pi Q_I) + \vec{z_2}\sin(\phi_I + 2\pi Q_I)\right) && (2.29)\\ &+ \sqrt{\epsilon_{II}}\left(\vec{z_3}\cos(\phi_{II} + 2\pi Q_{II}) + \vec{z_4}\sin(\phi_{II} + 2\pi Q_{II})\right) && (2.30)\\ &+ \sqrt{\epsilon_{III}}\left(\vec{z_5}\cos(\phi_{III} + 2\pi Q_{III}) + \vec{z_6}\sin(\phi_{III} + 2\pi Q_{III})\right) && (2.31) \end{aligned}$$

The relation between T matrix and lattice functions can be found by:

$$\begin{pmatrix} \beta_{xI} = x_1^2 + x_2^2 & \gamma_{xI} = x'^2_1 + x'^2_2 & \alpha_{xI} = -(x_1x'_1 + x_2x'_2) & \Phi_{xI} = \arctan(x_2/x_1) \\ \beta_{xII} = x_3^2 + x_4^2 & \gamma_{xII} = x'^2_3 + x'^2_4 & \alpha_{xII} = -(x_3x'_3 + x_4x'_4) & \Phi_{xII} = \arctan(x_4/x_3) \\ \beta_{xIII} = x_5^2 + x_6^2 & \gamma_{xIII} = x'^2_5 + x'^2_6 & \alpha_{xIII} = -(x_3x'_5 + x_6x'_6) & \Phi_{xIII} = \arctan(x_6/x_5) \\ \beta_{yI} = y_1^2 + y_2^2 & \gamma_{yI} = y'^2_1 + y'^2_2 & \alpha_{yI} = -(y_1y'_1 + y_2y'_2) & \Phi_{yI} = \arctan(y_2/y_1) \\ \beta_{yII} = y_3^2 + y_4^2 & \gamma_{yII} = y'^2_3 + y'^2_4 & \alpha_{yII} = -(y_3y'_3 + y_4y'_4) & \Phi_{yII} = \arctan(y_4/y_3) \\ \beta_{yIII} = y_5^2 + y_6^2 & \gamma_{yIII} = y'^2_5 + y'^2_6 & \alpha_{yIII} = -(y_3y'_5 + y_6y'_6) & \Phi_{yIII} = \arctan(y_6/y_5) \\ \beta_{sI} = s_1^2 + s_2^2 & \gamma_{sI} = \delta_1^2 + \delta_2^2 & \alpha_{\sigma I} = -(\sigma_1\delta_1 + \sigma_2\delta_2) & \Phi_{\sigma I} = \arctan(\sigma_2/\sigma_1) \\ \beta_{\sigma II} = \sigma_3^2 + \sigma_4^2 & \gamma_{\sigma II} = \delta_3^2 + \delta_4^2 & \alpha_{\sigma II} = -(\sigma_3\delta_3 + \sigma_4\delta_4) & \Phi_{\sigma II} = \arctan(\sigma_4/\sigma_3) \\ \beta_{\sigma III} = \sigma_5^2 + \sigma_6^2 & \gamma_{\sigma III} = \delta_5^2 + \delta_6^2 & \alpha_{\sigma III} = -(\sigma_3\delta_5 + \sigma_6\delta_6) & \Phi_{\sigma III} = \arctan(\sigma_6/\sigma_5) \end{pmatrix} \tag{2.32}$$

where $a_i = \frac{\partial a}{\partial z_i}$, that is, $x_1 = \frac{\partial x}{\partial x}$, $x_2 = \frac{\partial x}{\partial x'}$, $x_3 = \frac{\partial x}{\partial y}$, ... and

To obtain the lattice functions in another point of the circular accelerator it is enough with transporting the generating vectors through the lattice.

2.3.5 Non-Linear motion

If the motion is linear and the one-turn map has stable eigenvalues, the motion of a single particle is bounded for an infinity large number of turns and any phase-space point can be associated with an linear invariant. However, the motion is perturbed in case of the existence of a non-linear force, leading to chaotic motion with particle trajectories that are not anymore bounded inside an ellipse of the phase space.

In case of weak-nonlinearities, the motion is still stable for a long time in a relatively large area of the phase-space whose projection in the x,y plane is usually comparable with the vacuum pipe cross-section of an accelerator. In fact, if it is too small, not so many particles can circulate in the machine and most of them will be lost quickly. The region is typically not very large, since non-linear elements are added on purpose to compensate energy errors and provide a spread of the frequency of the natural modes.

For weak non-linearities it still makes sense to calculate linear invariant like amplitudes, tunes, lattice functions that this time will evolve turn-by-turn.

In addition it is also useful to track particles with very close coordinates to study the sensitivity of the motion under a small variation of the initial conditions.

Instead of looking directly at the trajectory coordinates, the coefficient of the linear parameterization (like ϵ, ϕ) offers a more stable and decoupled set of the coordinates, whose unique source of variation comes from the non-linear motion because in case of following an ideal linear motion they are invariant.

SixTrack computes both particle trajectories and linear lattice functions whose combined analysis forms the foundation of the post-processing routines.

Chapter 3

SixTrack Post-Processing

SixTrack [17] is a single-particle six-dimensional symplectic tracking code optimized for long-term tracking in high-energy rings. Based in RACETRACK [19] and published for the first time in 1990 by Frank Schmidt, it has been used for several kinds of studies for the LHC like dynamic aperture, tune optimization or collimation studies.

SixTrack has been in heavy development for the past 25 years [5], and it will continue evolving in the near future with improvements like our database-based post-processing port, new physics models, new particle parameters (mass, charge state...) and the development of GPU libraries to delegate the computationally heavy parts on them.

This chapter introduces SixTrack as a particle-tracking code and its post-processing subroutine, describing the problems tackled with it, its relationship with the tracking routines and how does everything fit together with SixDesk's post-processing runs. Section 3.1 explains the motivation behind particle tracking, while Section 3.3 points specifically to SixTrack's post-processing subroutines and provides a documented walk through their code and Section 3.4 gives an overview of the whole simulation and study process as it is run by SixDesk.

3.1 Particle Tracking

Particle tracking has become a key tool for the simulation and study of the beam behavior inside an accelerator. There are some interesting studies to be done with the help of particle tracking [10]. We use it for the study of, among others, the possibility of calculating and predicting the dynamic aperture (effective aperture of the particle motion), its dependence on the nonlinearities, tunes, finding closed orbit distortions or long-term beam stability.

During the past thirty years, simulation codes like SixTrack, MAD-X, RACE-TRACK or PTC have been in development to apply particle tracking techniques from different approaches. This codes run simulations of the movement of the particle inside the high-energy ring for thousands of turns, observing the characteristics of the particle (amplitude, phase advance, coordinates,...) in the position s_0 for each one of this turns.

Tracking codes have traditionally had the problems of rounding precision and limited CPU-time. While the processing time has been improving over time and had the help of new techniques like distributed computing (CPSS project [13], LHC@Home project [7]), the problem of rounding-related differences between platforms appeared for each one of the compilers available, until was finally solved in SixTrack with the inclusion of the `crlibm` library [3], which corrected rounding error difference in a portable way [12], the enforcement of strict ordering of the operation and specific compiler flags, making the result numerically reproducible.

A typical SixTrack run for a DA study in the LHC involves the simulation of 5 different angles in the phase-space and 60 different representation of the magnetic errors (seeds), in each of those cases tracking 30 pairs of particles for 100^5 turns. A study of that size can take several days, as each one of the cases needs around 2 hours of computing time to be finished.

3.2 SixTrack

SixTrack as a single-particle tracking code is being used to simulate the motion of charged particles inside the LHC. SixTrack is written in Fortran 77 and takes more than 70000 lines of code in a structure that requires the use of a pre-processor to simplify the code and to substitute special expressions on the source files applying

the Differential Algebra extension for Fortran (DAFOR) [2].

This structure can be observed in the Figure 4.5.

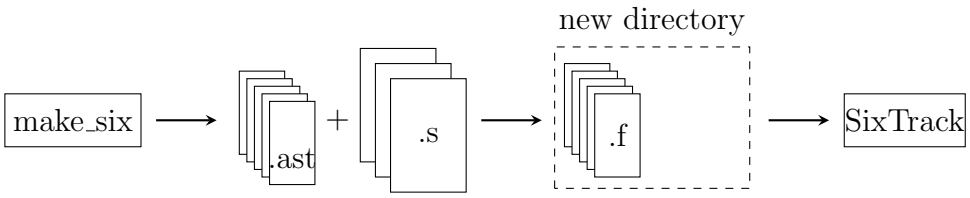


Figure 3.1: SixTrack build process. The `make_six` file generates an `.ast` file for each Fortran file to be produced. The `.ast` files produce the Fortran files from the source code in the three `.s` files, `sixtrack.s`, `lielib.s` and `dabnew.s`. The Fortran files are linked and compiled to produce the SixTrack executable file.

`make_six` has to be invoked with certain flags to choose which `.ast` mask files are going to be used for selecting the blocks of code from the `.s` source files that will appear on the final Fortran files.

While running the `make_six` script, five Fortran files are generated:

- **track.f** File that contains the tracking maps for the lattice: dipoles, quadrupoles, octupoles, RF cavities, higher-order magnets, crab cavities...although most of the subroutines are adapted for both 4-D and the 6D case, for some specific cases it's only possible to use the 6D approach.
- **sixve.f** Main Fortran file. It defines the main program and most of the subroutines. The `postpr` post-processing subroutine is contained on this file.
- **sixvefox.f** Contains subroutines for the calculation of the closed orbit and the lattice optics mapping using a differential algebra approach.
- **lielib.f** E. Forest subroutines for normal form calculations [6].
- **dabnews.f** DAFOR subroutines called by `sixvefox.f` during the close orbit calculation.

This files will be compiled into a Fortran executable that will take as an input the description of the accelerator in the `fort.2` file and the settings and description

of the elements from the `fort.3` file. Then, it will initialize all parameters and variables, compute the six-dimensional close orbit as well as the Courant-Snyder invariants and perform the tracking.

Finally, the tracking data is analyzed and post-processed as specified in the `fort.3` file.

3.3 SixTrack's Post-Processing: A Walkthrough

SixTrack's post-processing subroutine is the one in charge of and generating the fort.10 file, among others.

Sixtrack post-processes the data in the routine called `postpr(nfile)`. This routine reads binary files (`fort.90`, `fort.89`, ...) produced by the turn-by-turn tracking subroutine. This files contain data collected for a certain number of turns specified by the field `nwr(3)`.

The results of this post-processing are stored in the `fort.10` file, each row representing a measure and each column a variable. A detailed description of each one of the columns of the final `fort.10` can be found in Table 3.1 and Table 3.2, while a reference for the beam optics involved can be found in Table 3.3.

What follows is a walk through the different sections of the subroutine and its flow. Each one of this sections will be introduced by a descriptive title and will contain a description of the most relevant lines of code and variables it contains.

Keep in mind that this walkthrough is not a line-by-line description of the code and it is supposed to be followed while reading the original Sixtrack' s Fortran source code. It should be read sequentially and while most of the sections will be explained in detail, some of them will just have a description of what is happening in that portion of the code.

All the formulas expressed in this code have been obtained looking at the code and double-checking them with the reference bibliography.

C	Variable	Description	Formula
1	nnuml	Maximum turn number	
2	nlost	Stability Flag (0=stable, 1=lost)	
3	qwc(1)	Horizontal Tune	Q_x
4	qwc(2)	Vertical Tune	Q_y
5	bet0(1)	Horizontal βfunction	$\beta_{xI} = {T_{1,1}}^2 + {T_{1,2}}^2$
6	bet0(2)	Vertical βfunction	$\beta_{yI} = {T_{3,3}}^2 + {T_{3,4}}^2$
7		Horizontal amplitude 1st particle	$\sqrt{\beta_{xI} * \epsilon_I} + \sqrt{\beta_{xII} * \epsilon_{II}}$
8		Vertical amplitude 1st particle	$\sqrt{\beta_{yI} * \epsilon_I} + \sqrt{\beta_{yII} * \epsilon_{II}}$
9		Relative momentum deviation $\frac{\Delta p}{p_0}$	$\frac{\Delta p}{p_0}$ - δ_0
10	biav(i)	Final distance in phase space	
11	slope(i)	Max. slope of distance in phase space	
12		Horizontal detuning	$\overline{\psi}_x - Q_x$
13	sdpx	Spread of horizontal detuning	
14		Vertical detuning	$\overline{\psi}_y - Q_y$
15	sdpz	Spread of vertical detuning	
16	im1s	Horizontal factor to nearest resonance	
17	jm1s	Vertical factor to nearest resonance	
18	im1s+jm1s	Order of nearest resonance	
19	sevx	Horizontal smear	$sevx_{i-1} + (\epsilon_{vx_i} - \overline{\epsilon_{vx_i}})^2$
20	sevz	Vertical smear	$sevz_{i-1} + (\epsilon_{vx_i} - \overline{\epsilon_{vz_i}})^2$
21	sevt	Transverse smear	$sevt_{i-1} + (\epsilon_{vx_i} + \epsilon_{vz_i} - \overline{\epsilon_{vt_i}})^2$
22	ia/ifipa	Survived turns 1st particle	
23	ia/ilapa	Survived turns 2nd particle	
24	dizu0	Starting seed for random generator	
25	tph6	Synchrotron tune	
26		Horizontal amplitude 2nd particle	$\sqrt{\beta_{xI} * \epsilon_{vx2}} + \sqrt{\beta_{xII} * \epsilon_{vy2}}$
27		Vertical amplitude 2nd particle	$\sqrt{\beta_{yI} * \epsilon_{vy2}} + \sqrt{\beta_{yII} * \epsilon_{vx2}}$
28		Minimum horizontal amplitude	$\sqrt{\beta_{yI} * \lvert \min \epsilon_x \rvert}$
29		Mean horizontal amplitude	$\sqrt{\beta_{xI} * \overline{\epsilon_x}}$
30		Maximum horizontal amplitude	$\sqrt{\beta_{xI} * \max \epsilon_x}$

Table 3.1: Description of the entries in the fort.10 results file, with the corresponding variable the Fortran source code when present.

C	Variable	Description	Formula
31		Minimum vertical amplitude	$\sqrt{\beta_{yI} * \lvert \min \epsilon_y \rvert}$
32		Mean vertical amplitude	$\sqrt{\beta_{yI} * \overline{\epsilon_y}}$
33		Maximum vertical amplitude	$\sqrt{\beta_{yI} * \max \epsilon_z}$
34		Minimum horizontal amplitude ld	$\sqrt{\beta_{yI} * \lvert \min \epsilon_{vx} \rvert}$
35		Mean horizontal amplitude ld	$\sqrt{\beta_{xI} * \overline{\epsilon_{vx}}}$
36		Maximum horizontal amplitude ld	$\sqrt{\beta_{xI} * \max \epsilon_{vx}}$
37		Minimum vertical amplitude ld	$\sqrt{\beta_{yI} * \lvert \min \epsilon_{vy} \rvert}$
38		Mean vertical amplitude ld	$\sqrt{\beta_{yI} * \overline{\epsilon_{vz}}}$
39		Maximum vertical amplitude ld	$\sqrt{\beta_{yI} * \max \epsilon_{vy}}$
40		Minimum horizontal amplitude nld	$\sqrt{(\beta_{xI} * \lvert \min \epsilon_{vt} \rvert) * \frac{\overline{\epsilon_x}}{\overline{\epsilon_x}+\overline{\epsilon_y}}}$
41		Mean horizontal amplitude nld	$\sqrt{(\beta_{xI} * \overline{\epsilon_{vt}}) * \frac{\overline{\epsilon_x}}{\overline{\epsilon_x}+\overline{\epsilon_y}}}$
42		Maximum horizontal amplitude nld	$\sqrt{(\beta_{xI} * \max \epsilon_{vt}) * \frac{\overline{\epsilon_x}}{\overline{\epsilon_x}+\overline{\epsilon_y}}}$
43		Minimum vertical amplitude nld	$\sqrt{(\beta_{yI} * \lvert \min \epsilon_{vt} \rvert) * \frac{\overline{\epsilon_y}}{\overline{\epsilon_x}+\overline{\epsilon_y}}}$
44		Mean vertical amplitude nld	$\sqrt{(\beta_{yI} * \overline{\epsilon_{vt}}) * \frac{\overline{\epsilon_y}}{\overline{\epsilon_x}+\overline{\epsilon_y}}}$
45		Maximum vertical amplitude ld	$\sqrt{(\beta_{yI} * \max \epsilon_{vt}) * \frac{\overline{\epsilon_y}}{\overline{\epsilon_x}+\overline{\epsilon_y}}}$
46	emi	Emittance Mode I	$\epsilon_I = \left(x \sum_{i=0}^{6} \frac{\partial z_i}{\partial x}\right)^2 + \left(\widetilde{x'} \sum_{i=0}^{6} \frac{\partial z_i}{\partial x'}\right)^2$
47	emii	Emittance Mode II	$\epsilon_{II} = \left(y \sum_{i=0}^{6} \frac{\partial z_i}{\partial y}\right)^2 + \left(\widetilde{y'}^2 \sum_{i=0}^{6} \frac{\partial z_i}{\partial y'}\right)^2$
48	bet0x2	Secondary horizontal βfunction	$\beta_{xII} = {T_{1,3}}^2 + {T_{1,4}}^2$
49	bet0z2	Secondary vertical βfunction	$\beta_{yII} = {T_{3,1}}^2 + {T_{3,2}}^2$
50	chrom(1)	Q'x	Q'_x
51	chrom(2)	Q'y	Q'_y
52	ttot	SixTrack Version	
53	clo(1)	Closed Orbit x	$z_{01} = x_0$
54	clop(1),	Closed Orbit x'	$z_{02} = {x_0}'$
55	clo(2)	Closed Orbit y	$z_{03} = y_0$
56	clop(2)	Closed Orbit y'	$z_{04} = {y_0}'$
57	clo(3)	Closed Orbit σ	$z_{05} = \sigma_0$
58	clop(3)	Closed Orbit δ	$z_{06} = \delta_0$
59	dnms	The number of the Random Set	
60	trttime	Tracking CPU time in seconds	

Table 3.2: Description of the column entries in the fort.10 results file, with the corresponding variable the Fortran source code when present. ld stands for linear decoupled, nld for nonlinear decoupled.

Symbol	Description	Formula
N	Number of particles	
z	Particle characteristics	$z = \{x, x', y, y', \sigma, \delta\}$
x, y	x and y coordinates	
p_x, p_y	x and y momenta	
σ	Path length	$\sigma = s - v_0 \times t$
δ	Relative momentum deviation	$\delta = \frac{\Delta P}{p_0}$
M	One-turn tracking Matrix	$T_{ij} = \frac{\partial z_i}{\partial z_j}$
z_0	Closed Orbit parameters	$z_0 = \{x_0, {x_0}', y_0, {y_0}', \sigma_0, \delta_0\}$
$\tilde{z}$	Charact. respect to the closed orbit	$\tilde{z} = \{\tilde{x}, \tilde{x}', \tilde{y}, \tilde{y}', \tilde{\sigma}, \tilde{\delta}\}$, $\tilde{z}_i = z_i - z_0$
$\widetilde{x}', \widetilde{y}'$	Canonical momentums $\widetilde{x}', \widetilde{y}'$	$\{\widetilde{x}', \widetilde{y}'\} = \{x', y'\} * ((1 + \delta) + \delta_0)$
Q_x, Q_y	Horizontal and vertical tunes	
Q'_x, Q'_y	Horizontal and vertical chromaticities	
$\psi, \overline{\psi}$	Phase advance, Mean phase advance	
ϵ_x	Horizontal emittance	$\epsilon_x = c^2 + \frac{\beta_{xI} * \widetilde{x}' + \alpha_{xI} * \tilde{x}}{\beta_{xI}}$
ϵ_y	Vertical emittance	$\epsilon_y = c^2 + \frac{\beta_{yI} * \widetilde{y}' + \alpha_{yI} * \tilde{y}}{\beta_{yI}}$
$\overline{\epsilon}$	Averaged emittance	
$\epsilon_I, \epsilon_{II}$	Mode I and Mode II emittances	$\epsilon_I = \epsilon_{vx}$, $\epsilon_{II} = \epsilon_{vy}$
ϵ_{vx}	Linear decoupled horizontal emittance	$\epsilon_{vx} = \left(\tilde{x} \sum_{i=0}^{6} T_{i,1}\right)^2 + \left(x' \sum_{i=0}^{6} T_{i,2}\right)^2$
ϵ_{vy},	Linear decoupled vertical emittance	$\epsilon_{vy} = \left(\tilde{y} \sum_{i=0}^{6} T_{i,3}\right)^2 + \left(\widetilde{y}' \sum_{i=0}^{6} T_{i,4}\right)^2$
ϵ_{vt}	Emittance sum...	$\epsilon_{vt} = \epsilon_x + \epsilon_y$
β_{xI}	Primary horizontal βfunction	$\beta_{xI} = {T_{1,1}}^2 + {T_{1,2}}^2$
β_{xII}	Secondary horizontal βfunction	$\beta_{xII} = {T_{1,3}}^2 + {T_{1,4}}^2$
β_{yI}	Primary vertical βfunction	$\beta_{yI} = {T_{3,3}}^2 + {T_{3,4}}^2$
β_{yII}	Secondary vertical βfunction	$\beta_{yII} = {T_{3,1}}^2 + {T_{3,2}}^2$

Table 3.3: Relations between optics quantities

Format of the Binary Data

For reference, the first thing to describe will be the binary data taken from the input files, specifying the units in which the variables are expressed:

Name	Description
`ia`	Turn number
`ifipa`	Particle number
`b`	Angular distance in phase space ($<= 1$)
`c`	x(mm)
`d`	x'(mrad)
`e`	y(mm)
`f`	y'(mrad)
`g`	Path-length ($\sigma = s - v_0 \times t$) (mm)
`h`	Relative momentum deviation $\delta = \frac{p}{p_0}$
`p`	Energy (MeV)

Initialization and Reading the Header

First of all the code imports all the common blocks it needs and sets up the variables, including the start time ones:

Name	Description
`cdate`	date
`ctime`	time

Then it reads the following initial coordinates and parameters from the file unit specified as argument:

Name	Description
ifipa	first particle in the file
ilapa	last particle in the file
itopa	total number of particles
icode	Dimensions of phase space (4D, 5D, 6D).
numl	Projected number of turns
qwc(1)	Horizontal Tune
qwc(2)	Vertical Tune
qwc(3)	Longitudinal Tune
ta(6,6)	6x6 transfer matrix e.g. ta(i,j) = T_i, j in SI unit
clo(i)	Closed Orbit vector (space)
clop(i)	Closed Orbit vector (momentum)
di0(i)	Dispersion vector (space)
dip0(i)	Dispersion vector (momentum)

At this point, the 5th and 6th columns of the `fort.10` file (β_{xI}, β_{yI}) can be computed right away and the values of the closed orbit coordinates can be written to file:

Variable	Content	Description
sumda(5)	ta(1,1)**2+ta(1,2)**2	Horizontal β-function
sumda(6)	ta(3,3)**2+ta(3,4)**2	Vertical β-function
sumda(53)	clo(1)	closed orbit x —x_0
sumda(54)	clop(1)	closed orbit x' —x'_0
sumda(55)	clo(2)	closed orbit y —y_0
sumda(56)	clop(2)	closed orbit y' —y'_0
sumda(57)	clo(3)	closed orbit σ —σ_0
sumda(58)	clop(3)	closed orbit δ —δ_0

This section of the code starts transforming the physical coordinates in normalized coordinates by using the T matrix (Equation 2.27).

The following identities are defined:

ta(i,1)	rcw1(i)	T_{i1}
ta(i,3)	rcw2(i)	T_{i3}
ta(i,2)	ycw1(i)	T_{i2}
ta(i,4)	ycw2(i)	T_{i4}

The matrix T can be calculated by either the routine `matrix` that calls `umlauf` using the 4D formalism or the routine `qmodda(3,qwc)` using the 6D formalism (`qwc` are the three tunes), which computes the one-turn matrix M from the lattice elements defined in `fort.2` input file.

Once the T matrix is obtained, the initial lattice functions (see Equation 2.32) at the initial point can be defined:

Variable	**Description**
`bet0(1)`	Horizontal β-function
`bet0x2`	Secondary horizontal β-function
`bet0x3`	Tertiary horizontal β-function
`gam0x1`	Horizontal γ-function
`gam0x2`	Secondary horizontal γ-function
`gam0x3`	Tertiary horizontal γ-function
`alf0(1)`	Horizontal α-function
`alf0x2`	Secondary horizontal α-function
`alf0x3`	Tertiary horizontal α-function
`bet0(2)`	Vertical β-function
`bet0z2`	Secondary vertical β-function
`bet0z3`	Tertiary vertical β-function
`gam0z1`	Vertical γ-function
`gam0z2`	Secondary vertical γ-function
`gam0z3`	Tertiary vertical γ-function
`alf0(2)`	Vertical α-function
`alf0z2`	Secondary vertical α-function
`alf0z3`	Tertiary vertical α-function
`bet0(3)`	Longitudinal β-function
`bet0s2`	Secondary longitudinal β-function
`bet0s3`	Tertiary β-function
`gam0s1`	Longitudinal γ-function
`gam0s2`	Secondary longitudinal γ-function
`gam0s3`	Tertiary longitudinal γ-function
`alf0(3)`	Longitudinal α-function
`alf0s2`	Secondary longitudinal α-function
`alf0s3`	Tertiary longitudinal α-function
`bet04(1)`	Horizontal β-function
`bet04(2)`	Vertical β-function
`alf04(1)`	Horizontal α-function
`alf04(2)`	Vertical α-function

and computed:

Code expression	**Relating formula**
`bet0(1)=ta(1,1)**2+ta(1,2)**2`	$\beta_{xI} = {T_{1,1}}^2 + {T_{1,2}}^2$
`bet0x2 =ta(1,3)**2+ta(1,4)**2`	$\beta_{xII} = {T_{1,3}}^2 + {T_{1,4}}^2$
`bet0x3 =ta(1,5)**2+ta16**2`	$\beta_{xIII} = {T_{1,5}}^2 + {T_{1,6}}^2$
`gam0x1 =ta(2,1)**2+ta(2,2)**2`	$\gamma_{xI} = {T_{2,1}}^2 + {T_{2,2}}^2$
`gam0x2 =ta(2,3)**2+ta(2,4)**2`	$\gamma_{xII} = {T_{2,3}}^2 + {T_{2,4}}^2$
`gam0x3 =ta(2,5)**2+ta26**2`	$\gamma_{xIII} = {T_{2,5}}^2 + {T_{2,6}}^2$
`alf0(1)=-(ta(1,1)*ta(2,1)+ta(1,2)*ta(2,2))`	$\alpha_{xI} = -(T_{1,1}T_{2,1} + T_{1,2}T_{2,2})$
`alf0x2 =-(ta(1,3)*ta(2,3)+ta(1,4)*ta(2,4))`	$\alpha_{xII} = -(T_{1,3}T_{2,3} + T_{1,4}T_{2,4})$
`alf0x3 =-(ta(1,5)*ta(2,5)+ta16*ta26)`	$\alpha_{xIII} = -(T_{1,5}T_{2,5} + T_{1,6}T_{2,6})$
`bet0(2)=ta(3,3)**2+ta(3,4)**2`	$\beta_{yI} = {T_{3,3}}^2 + {T_{3,4}}^2$
`bet0z2 =ta(3,1)**2+ta(3,2)**2`	$\beta_{yII} = {T_{3,1}}^2 + {T_{3,2}}^2$
`bet0z3 =ta(3,5)**2+ta36**2`	$\beta_{yIII} = {T_{3,5}}^2 + {T_{3,6}}^2$
`gam0z1 =ta(4,3)**2+ta(4,4)**2`	$\gamma_{yI} = {T_{4,3}}^2 + {T_{4,4}}^2$
`gam0z2 =ta(4,1)**2+ta(4,2)**2`	$\gamma_{yII} = {T_{4,1}}^2 + {T_{4,2}}^2$
`gam0z3 =ta(4,5)**2+ta46**2`	$\gamma_{yIII} = {T_{4,5}}^2 + {T_{4,6}}^2$
`alf0(2)=-(ta(3,3)*ta(4,3)+ta(3,4)*ta(4,4))`	$\alpha_{yI} = -(T_{3,3}T_{4,3} + T_{3,4}T_{4,4})$
`alf0z2 =-(ta(3,1)*ta(4,1)+ta(3,2)*ta(4,2))`	$\alpha_{yII} = -(T_{3,1}T_{3,1} + T_{3,2}T_{4,2})$
`alf0z3 =-(ta(3,5)*ta(4,5)+ta36*ta46)`	$\alpha_{yIII} = -(T_{3,5}T_{4,5} + T_{3,6}T_{4,6})$
`bet0(3)=ta(5,5)**2+ta56**2`	$\beta_{sI} = {T_{5,5}}^2 + {T_{5,6}}^2$
`bet0s2 =ta(5,1)**2+ta(5,2)**2`	$\beta_{sII} = {T_{5,1}}^2 + {T_{5,2}}^2$
`bet0s3 =ta(5,3)**2+ta(5,4)**2`	$\beta_{sIII} = {T_{5,3}}^2 + {T_{5,4}}^2$
`gam0s1 =ta65**2+ta(6,6)**2`	$\gamma_{sI} = {T_{6,5}}^2 + {T_{6,6}}^2$
`gam0s2 =ta61**2+ta62**2`	$\gamma_{sII} = {T_{6,1}}^2 + {T_{6,2}}^2$
`gam0s3 =ta63**2+ta64**2`	$\gamma_{sIII} = {T_{6,3}}^2 + {T_{6,4}}^2$
`alf0(3)=-(ta(5,5)*ta65+ta56*ta(6,6))`	$\alpha_{sI} = -(T_{5,5}T_{6,5} + T_{5,6}T_{6,6})$
`alf0s2 =-(ta(5,1)*ta61+ta(5,2)*ta62)`	$\alpha_{sII} = -(T_{5,1}T_{5,2} + T_{6,1}T_{6,2})$
`alf0s3 =-(ta(5,3)*ta63+ta(5,4)*ta64)`	$\alpha_{sIII} = -(T_{5,3}T_{6,3} + T_{5,4}T_{6,4})$
`bet04(1)=bet0(1)`	$\beta_{xI} = {T_{1,1}}^2 + {T_{1,2}}^2$
`bet04(2)=bet0(2)`	$\beta_{yI} = {T_{3,3}}^2 + {T_{3,4}}^2$
`alf04(1)=alf0(1)`	$\alpha_{xI} = T_{1,1}T_{2,1} + T_{1,2}T_{2,2}$
`alf04(2)=alf0(2)`	$\alpha_{yI} = T_{3,3}T_{4,3} + T_{3,4}T_{4,4}$

Setting up of the parameters

In this section is where the titles of the plots are determined and where the variables are set to zero. The most interesting part in this section of the code is the inversion

of the matrix of the generating vectors, where the vector

Variable	Description	Formula
t(i,j)=ta(j,i)	Transposed of the one-turn transfer matrix	$T^T_{j,i}$

is computed and its inversion is achieved by placing a call to the subroutine dinv(n,a,idim,ir,ifail), where in this case n is 6, a is the transposed (also inverse as it is symplectic) of the one turn transfer matrix, idim is 6, ir gets a dummy value and ifail is the error retrieving variable:

parameter	value	Description
n	6	
a	t	Transposed/inverse of the one turn transfer matrix
idim	6	
ir	dummy	
ifail	error	Error retrieving variable

Finally, the 6D flag is set if the variable

Variable	Description	Formula
tasum	Summation	$\sum_{i=1}^{6}(T_{5,i}+T_{6,i})+\sum_{i=1}^{4}(T_{i,5}+T_{i,6})-4$

is not zero.

Find minimum value of the distance in the phase space

Here it calls the soubroutine distance(x,clo,di0,t,dam) to calculate the minimum value of the distance in the phasespace for two twin particles.

Param	Value	Description
x	x	concatenated array of coordinates for both twin particles
clo	cloau	Closed orbit vector (space)
di0	di0au	Dispersion vector (space)
t	t	Transposed (also inverse) of the one turn transfer matrix
dam	b	Angular distance in phase space (<= 1)

Get first data point as a reference

This section is where the actual post processing starts. Sequentially it:

- Reads the first binary record of the file (first data point) and prints it as the initial coordinates.

- Detects the onset of chaotic motion and thereby the long-term dynamic aperture by evaluating the Lyapunov exponent. Calls the subroutine `distance` as described above.

- Saves the number of turn `ia` as the first turn (`ia0`), initializes variables and computes the deviations respect to the closed orbit of the particle ($\tilde{z}_i$) and the twin particle ($\tilde{z'}_i$) :

Line	**Formula**
`ia0 = ia`	
`xxr(1) = c`	
`xxi(1) = zero`	
`zzr(1) = e`	
`zzi(1) = zero`	
`c = c-clo(1)`	$\tilde{x}$
`d = d-clop(1)`	$\tilde{x'}$
`e = e-clo(2)`	$\tilde{y}$
`f = f-clop(2)`	$\tilde{y'}$
`g = g-clo(3)`	$\tilde{\sigma}$
`h = h-clop(3)`	$\tilde{\delta}$
`c1 = c1-clo(1)`	$\tilde{x_2}$
`d1 = d1-clop(1)`	$\tilde{x'_2}$
`e1 = e1-clo(2)`	$\tilde{y_2}$
`f1 = f1-clop(2)`	$\tilde{y'_2}$
`g1 = g1-clo(3)`	$\tilde{\sigma_2}$
`h1 = h1-clop(3)`	$\tilde{\delta_2}$

- Performs the calculation of the emittances in the x and y phase:

Variable	Description
xp0	intermediate variable
zp0	intermediate variable
emx	Horizontal emittance
emz	Vertical emittance
emt	Emittance sum
emax	Initialization of the maximum horizontal emittance variable
emix	Initialization of the minimum horizontal emittance variable
emxa	Initialization of the averaged horizontal emittance variable
emaz	Initialization of the maximum vertical emittance variable
emiz	Initialization of the minimum vertical emittance variable
emza	Initialization of the averaged vertical emittance variable
emat	Initialization of the maximum composed emittance variable
emit	Initialization of the minimum composed emittance variable
emta	Initialization of the averaged composed emittance variable
emx0	Initialization of the initial horizontal emittance variable
emz0	Initialization of the initial vertical emittance variable

...following the formulas:

Expression	Formula
xp0=bet0(1)*d+alf0(1)*c	$xp_0 = \beta_{xI} * \widetilde{x'} + \alpha_{xI} * \tilde{x}$
zp0=bet0(2)*f+alf0(2)*e	$yp_0 = \beta_{yI} * \widetilde{y'} + \alpha_{yI} * \tilde{y}$
emx = (c**2+xp0**2)/bet0(1)	$\epsilon_x = \frac{\tilde{x}^2+(\beta_{xI}*\widetilde{x'}+\alpha_{xI}*\tilde{x})^2}{\beta_{xI}}$
emz =(e**2+zp0**2)/bet0(2)	$\epsilon_y = \frac{\tilde{y}^2+(\beta_{yI}*y'+\alpha_{yI}*\tilde{y})^2}{\beta_{yI}}$
emt = emx+emz	$\epsilon_{vt} = \epsilon_x + \epsilon_y$
emax = emx	$\max \epsilon_x = \epsilon_x$
emix = emx	$\min \epsilon_x = \epsilon_x$
emxa = emx	$\overline{\epsilon_x} = \epsilon_x$
emaz = emz	$\max \epsilon_y = \epsilon_y$
emiz = emz	$\min \epsilon_y = \epsilon_y$
emza = emz	$\overline{\epsilon_y} = \epsilon_y$
emat = emt	$\max \epsilon_t = \epsilon_t$
emit = emt	$\min \epsilon_t = \epsilon_t$
emta = emt	$\overline{\epsilon_t} = \epsilon_t$
emx0 = emx	$\epsilon_{x0} = \epsilon_x$
emz0 = emz	$\epsilon_{y0} = \epsilon_y$

- Initializes the Courant-Snyder array with the coordinates from the first particle and makes the conversion to canonical variables:

Variable	Description
`xyzv`	vector initialized to $\tilde{z}$
`txyz(iq)`	Courant-Snyder array

Code	Formula
`xyzv`	$xyzv(i) = \tilde{z}_i$
`txyz(iq) = txyz(iq)+t(jq,iq)*xyzv(jq)`	$txyz(i) = \tilde{z}_i \sum_{j=1}^{6} T_{j,i}$

- Prints the angular distance in the phase-space.
- Computes the horizontal and vertical emittances with linear coupling.

Variable	Description
`evx`	Horizontal linear coupled emittance
`evx`	Vertical linear coupled emittance

Code	Formula
`evx = txyz(1)**2+txyz(2)**2)`	$\epsilon_{vx} = \left(\tilde{x} \sum_{j=1}^{6} T_{j,1}\right)^2 + \left(\tilde{x'} \sum_{j=1}^{6} T_{j,2}\right)^2$
`evx = txyz(3)**2+txyz(4)**2`	$\epsilon_{vy} = \left(\tilde{y} \sum_{j=1}^{6} T_{j,3}\right)^2 + \left(\tilde{y'} \sum_{j=1}^{6} T_{j,4}\right)^2$

- Does the conversion to canonical variable as it did above, this time for the twin particle.
- If the 6-D approach is the one being used, computes the third linear coupled emittance.

`emiii = txyz(5)**2*cma2**2+txyz(6)**2*cma1**2`
$\epsilon_{III} = \left(\tilde{\sigma} \sum_{j=1}^{6} T_{j,5}\right)^2 + \left(\tilde{\delta} \sum_{j=1}^{6} T_{j,6}\right)^2$

- Multiplies the Courant-Snyder array by `rbeta`.
- Initializes the variables used to represent the mínimum and máximum values of the coordinates, energy and angular distance in the phase space.

Variable	Description
pmin(2)	Minimum value for the angular distance in phase space parameter
pmax(2)	Maximum value for the angular distance in phase space parameter
pmin(3)	Minimum value for the x parameter
pmax(3)	Maximum value for the x parameter
pmin(4)	Minimum value for the x' parameter
pmax(4)	Maximum value for the x' parameter
pmin(5)	Minimum value for the y parameter
pmax(5)	Maximum value for the y parameter
pmin(6)	Minimum value for the y' parameter
pmax(6)	Maximum value for the y' parameter
pmin(9)	Minimum value for the σ parameter
pmax(9)	Maximum value for the σ parameter
pmin(10)	Minimum value for the δ parameter
pmax(10)	Maximum value for the δ parameter
pmin(16)	Minimum value for the energy parameter
pmax(16)	Maximum value for the energy parameter

Code	Formula
pmin(2) = b	$p_{min}(2) = \theta_{initial}$
pmax(2) = b	$p_{max}(2) = \theta_{initial}$
pmin(3) = c0	$p_{min}(3) = x_{initial}$
pmax(3) = c0	$p_{max}(3) = x_{initial}$
pmin(4) = d0	$p_{min}(4) = x'_{initial}$
pmax(4) = d0	$p_{max}(4) = x'_{initial}$
pmin(5) = e0	$p_{min}(5) = y_{initial}$
pmax(5) = e0	$p_{max}(5) = y_{initial}$
pmin(6) = f0	$p_{min}(6) = y'_{initial}$
pmax(6) = f0	$p_{max}(6) = y'_{initial}$
pmin(9) = g0	$p_{min}(9) = \sigma_{initial}$
pmax(9) = g0	$p_{max}(9) = \sigma_{initial}$
pmin(10) = h0	$p_{min}(10) = \delta_{initial}$
pmax(10) = h0	$p_{max}(10) = \delta_{initial}$
pmin(16) = p	$p_{min}(16) = E_{initial}$
pmax(16) = p	$p_{max}(16) = E_{initial}$

- Initializes the maximum/minimum linear decoupled emittances and the angle variables.

Code	Description	Formula
`emi = evx`	Initialize ϵ_I	$\epsilon_I = \epsilon_{vx}$
`emii = evz`	Initialize ϵ_{II}	$\epsilon_{II} = \epsilon_{vy}$
`angi = zero`	Initialize θ_I	$\theta_I = 0$
`angii = zero`	Initialize θ_{II}	$\theta_{II} = 0$
`angiii = zero`	Initialize θ_{III}	$\theta_{III} = 0$
`evt = evx+evz`	Computes ϵ_{vt}	$\epsilon_{vt} = \epsilon_{vx} + \epsilon_{vy}$
`evxma = evx`	Initialize the maximum of ϵ_{vx}	$\max \epsilon_{vx} = \epsilon_{vx}$
`evzma = evz`	Initialize the maximum of ϵ_{vy}	$\max \epsilon_{vy} = \epsilon_{vy}$
`evtma = evt`	Initialize the maximum of ϵ_{vt}	$\max \epsilon_{vt} = \epsilon_{vt}$
`evxmi = evx`	Initialize the minimum of ϵ_{vx}	$\min \epsilon_{vx} = \epsilon_{vx}$
`evzmi = evz`	Initialize the minimum of ϵ_{vy}	$\min \epsilon_{vy} = \epsilon_{vy}$
`evtmi = evt`	Initialize the minimum of ϵ_{vt}	$\min \epsilon_{vt} = \epsilon_{vt}$

`If(abs(txyz(5)).gt.pieni.or.abs(txyz(6)).gt.pieni):`
angiii = atan2_rn(txyz(6)*cma1,txyz(5)*cma2)

$$if\ any\ of\ \{abs(\sigma \sum_{i=1}^{l} T_{j,5}), abs(\delta \sum_{i=1}^{l} T_{j,6})\} > 0\ then:$$
$$\theta_{III} = call\ \texttt{atan2\_rn}(\sigma \sum_{i=1}^{l} T_{j,5}\, \delta \sum_{i=1}^{l} T_{j,6})$$

- Performs a coordinate-angle conversion with the subroutine `caconv(a,b,c)`, which computes the arc tangent (round near) between b and c and stores it in a. Then it adds the new angle to the initial one:

line `call caconv(dpx,d0,c0)`
`call caconv(dpz,f0,e0)`
`dpxp=tpi+dpx`
`dpzp=tpi+dpz`

- Computes the two invariants that define the elliptical boundary with the subroutine `cinv`.
`cinvar(dpx,dphix,dpz,dpzp,nuex,emz,zinv,invz)` adds $ninv * \epsilon_{my}$ to `zinv` and `ninv` to `invz`.
For the other invariant, `cinvar(dpz,dphiz,dpx,dpxp,nuez,emx,xinv,invx)` adds $ninv * \epsilon_{mx}$ to `xinv` and `ninv` to `invx`.

Get Data Points

Once the first data points are obtained and processed, the subroutine proceeds to get all the other data points and transports them to the circular phase space. Most of the code will be replicated from the previous section of the code:

- Retrieves the parameters of the new particle and increments the counter of binary records in one.
- Detects the onset of chaotic motion and thereby the long-term dynamic aperture by evaluating the Lyapunov exponent. Calls the subroutine `distance` as described above.
- Works on the calculation of the emitances in the x and y phase spaces:

Code	Description
`emt`	Emittance sum
`emxa`	Averaged horizontal emittance variable
`emza`	Averaged vertical emittance variable
`emta`	Averaged composed emittance variable
`emax`	Maximum horizontal emittance variable
`emix`	Minimum horizontal emittance variable
`emaz`	Maximum vertical emittance variable
`emiz`	Minimum vertical emittance variable
`emat`	Maximum composed emittance variable
`emit`	Minimum composed emittance variable

Code	Formula
`emt = emx+emz`	$\epsilon_{vt} = \epsilon_x + \epsilon_y$
`emxa = emx`	$\overline{\epsilon_x} = \overline{\epsilon_x} + \epsilon_x$
`emza = emz`	$\overline{\epsilon_y} = \overline{\epsilon_y} + \epsilon_y$
`emta = emt`	$\overline{\epsilon_t} = \overline{\epsilon_t} + \epsilon_t$
`emax = emx`	$\max \epsilon_x = \max(\epsilon_x, \max \epsilon_x)$
`emix = emx`	$\min \epsilon_x = \min(\epsilon_x, \min \epsilon_x)$
`emaz = emz`	$\max \epsilon_y = \max(\epsilon_y, \max \epsilon_y)$
`emiz = emz`	$\min \epsilon_y = \min(\epsilon_y, \min \epsilon_y)$
`emat = emt`	$\max \epsilon_t = \max(\epsilon_t, \max \epsilon_t)$
`emit = emt`	$\min \epsilon_t = \min(\epsilon_t, \min \epsilon_t)$

- Initializes the Courant-Snyder array and converts it to canonical variables [1].

[1] Dissected in previous subsections

- Computes the horizontal and vertical emittances with linear coupling[1].

- Updates the variables for the mínimum and máximum values of the coordinates, energy and angular distance in the phase space:

Code	**Formula**
`pmin(2) = min(pmin(2),b)`	$p_{min}(2) = \min(p_{min}(2), \theta)$
`pmax(2) = max(pmax(2),b)`	$p_{max}(2) = \max(p_{max}(2), \theta)$
`pmin(3) = min(pmin(3),c0)`	$p_{min}(3) = \min(p_{min}(3), x)$
`pmax(3) = max(pmax(3),c0)`	$p_{max}(3) = \max(p_{max}(3), x)$
`pmin(4) = min(pmin(4),d0)`	$p_{min}(4) = \min(p_{min}(4), x')$
`pmax(4) = max(pmax(4),d0)`	$p_{max}(4) = \max(p_{max}(4), x')$
`pmin(5) = min(pmin(5),e0)`	$p_{min}(5) = \min(p_{min}(5), y)$
`pmax(5) = max(pmax(5),e0)`	$p_{max}(5) = \max(p_{max}(5), y)$
`pmin(6) = min(pmin(6),f0)`	$p_{min}(6) = \min(p_{min}(6), y')$
`pmax(6) = max(pmax(6),f0)`	$p_{max}(6) = \max(p_{max}(6), y')$
`pmin(9) = min(pmin(9),g0)`	$p_{min}(9) = \min(p_{min}(9), \sigma)$
`pmax(9) = max(pmax(9),g0)`	$p_{max}(9) = \max(p_{max}(9), \sigma)$
`pmin(10) = min(pmin(10),h0)`	$p_{min}(10) = \min(p_{min}(10), \delta)$
`pmax(10) = max(pmax(10),h0)`	$p_{max}(10) = \max(p_{max}(10), \delta)$
`pmin(16) = min(pmin(16),p)`	$p_{min}(16) = \min(p_{min}(16), E)$
`pmax(16) = max(pmax(16),p)`	$p_{max}(16) = \max(p_{max}(16), E)$

- Maintains a log of the distances of the phase-space and saves the difference in the number of turns per data entry

Code	**Description**
`ia=ia-nstart`	difference in turns since the beginning of the analysis
`ia=idnt=ia-ia0`	difference in turns since the beginning of the analysis

- Performs the computation of the emittances.

Code	Description	Formula
`evt1 = evt1+evz1`	Computes ϵ_{vt1}	$\epsilon_{vt1} = \epsilon_{vx1} + \epsilon_{vy1}$
`evxma = evx`	Saves the maximum of ϵ_{vx}	$\max \epsilon_{vx} = \max(\max \epsilon_{vx}, \epsilon_{vx1})$
`evzma = evz`	Saves the maximum of ϵ_{vy}	$\max \epsilon_{vy} = \max(\max \epsilon_{vy}, \epsilon_{vy1})$
`evtma = evt`	Saves the maximum of ϵ_{vt}	$\max \epsilon_{vt} = \max(\max \epsilon_{vt}, \epsilon_{vt1})$
`evxmi = evx`	Saves the minimum of ϵ_{vx}	$\min \epsilon_{vx} = \min(\min \epsilon_{vx}, \epsilon_{vx1})$
`evzmi = evz`	Saves the minimum of ϵ_{vy}	$\min \epsilon_{vy} = \min(\min \epsilon_{vy}, \epsilon_{vy1})$
`evtmi = evt`	Saves the minimum of ϵ_{vt}	$\min \epsilon_{vt} = \min(\min \epsilon_{vt}, \epsilon_{vt1})$
`evx = evx+evx1`	Computes ϵ_{vx}	$\epsilon_{vx} = \epsilon_{vx} + \epsilon_{vx1}$
`evz = evz+evz1`	Computes ϵ_{vy}	$\epsilon_{vy} = \epsilon_{vy} + \epsilon_{vy1}$
`evt = evt+evt1`	Computes ϵ_{vt}	$\epsilon_{vt} = \epsilon_{vt} + \epsilon_{vt1}$

- Performs the adding of the phase advances using the subroutine `cphase`, whose arguments in order would be (replace x with y and 6/s for the other dimensions):

Argument	Description
`1`	Row of the `phase` matrix to use
`dphx`	Variable to write on
`sx`	$s_x = x * x'_{initial} - x_{initial} * x'$
`cx`	$s_y = x_{initial} * x + x'_{initial} * x'$
`qx0`	Horizontal tune `qx0`
`ivox`	switch for the Q_x-value close to a half-integer
`iwarx`	set to one if sx and cx $< 10^3 8$ (below minimum precision)
`iapx`	iapx+1 if sx or cx $> 10^3 8$

- Data is averaged in samples of IAV turns.

- The coordinate-angle conversion is performed by the subroutine `caconv(a,b,c)`, which performs the arc-tangent (round near) between b and c and stores it in a. Then it adds the new angle to the initial one[1].

- Computes with the subroutine `cinv` the two invariants that define the elliptical boundary[1].

- Resets the coordinates by taking the actual particle as a reference for the next one: $z_{initial} = z$.

Analyzing Data

Here the program analyzes the data, gets the necessary variables(phase advance, emittances) and ports them back to the circular phase space.

- First, it fits the distance in the phase space calling the two fitting functions [14].

Call	Description
`lfitwd(x,y,w,l,key,a,b,e)`	Fits a straight line y=a*x+b to `l` points with s**2 estimator e.
`lfitd(x,y,w,l,key,a,b,e)`	Fits a straight line y=a*x+b to `l` points with error e.

where weights are in `w` and `l` is the number of points. If `key=0`, points `with y=0` are ignored .

The call

`lfitwd(x,y,w,l,key,a,b,e)`

applies the operations:

$$a = \frac{\sum_{j=1}^{l} {w_j}^2 x_j y_j - \frac{\sum_{j=1}^{l} {w_j}^2 x_j * \sum_{j=1}^{l} {w_j}^2 y_j}{\sum_{j=1}^{l} {w_j}^2}}{\sum_{j=1}^{l} {w_j}^2 {x_j}^2 - \frac{(\sum_{j=1}^{l} {w_j}^2 x_j)^2}{\sum_{j=1}^{l} {w_j}^2}}$$

$$b = \frac{\sum_{j=1}^{l} {w_j}^2 y_j - a \sum_{j=1}^{l} {w_j}^2 x_j}{\sum_{j=1}^{l} {w_j}^2}$$

$$e = \frac{\sum_{j=1}^{l} {w_j}^2 {y_j}^2 - \frac{\sum_{j=1}^{l} {w_j}^2 y_j}{\sum_{j=1}^{l} {w_j}^2} - a \left(\sum_{j=1}^{l} {w_j}^2 x_j y_j - \frac{\sum_{j=1}^{l} {w_j}^2 x_j * \sum_{j=1}^{l} {w_j}^2 y_j}{\sum_{j=1}^{l} {w_j}^2} \right)}{n}$$

while calling

`lfitd(x,y,w,l,key,a,b,e)`

answers to

$$a = \frac{\sum_{i=1}^{l}(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{l}(x_i - \overline{x})^2}$$
$$b = \overline{y} - a\overline{x}$$
$$e = \frac{\sum_{i=1}^{l}(y_i - \overline{y}) - a\left(\sum_{i=1}^{l}(x_i - \overline{x})(y_i - \overline{y})\right)}{n - 2}$$

- After the distance fitting it computes the averaged phase advances:

Code	**Formula**
`tphx=dphx/dble(iapx)`	$\overline{\psi_x}$
`tphz=dphz/dble(iapz)`	$\overline{\psi_y}$
`tph6=dph6/dble(iap6)`	$\overline{\psi_s}$

  where `tphx`, `tphz` and `tph6` are respectively the averaged phase advance on the x coordinate, the averaged phase advance on the y coordinate and the averaged phase advance on the longitudinal coordinate.

- For obtaining the standard deviation of the phase advances, it loops from `i=1` to `iapx`, `iapz` and `iaps`, respectively computing (example for the x case):

```
sdpx=sdpx+(phase(1,i)-tphx)**2.
```

  and after the loop :

```
sdpx=sqrt(sdpx)/dble(iapx).
```

  or, expressed as an equation (for the three coordinates):

$$\sqrt{\frac{\sum_{i=1}^{iapx}(\phi_{x,i} - \overline{\psi}_x)^2}{iapx}}$$
$$\sqrt{\frac{\sum_{i=1}^{iapy}(\phi_{y,i} - \overline{\psi}_y)^2}{iapy}}$$
$$\sqrt{\frac{\sum_{i=1}^{iaps}(\phi_{s,i} - \overline{\psi}_s)^2}{iaps}}$$

- Then it looks for the averaged emittances. Being n the number of particles retrieved and knowing that all the average variables have been just summations until now:

Code	Description	Formula
`emxa=emxa/di11`	Averages the ϵ_x emittances	$\overline{\epsilon_x} = \frac{\overline{\epsilon_x}}{n}$
`emza=emza/di11`	Averages the ϵ_y emittances	$\overline{\epsilon_y} = \frac{\overline{\epsilon_y}}{n}$
`emta=emta/di11`	Averages the ϵ_t emittances	$\overline{\epsilon_t} = \frac{\overline{\epsilon_t}}{n}$
`evxm=evx/di11`	Averages the ϵ_{vx} emittances	$\overline{\epsilon_{vx}} = \frac{\overline{\epsilon_{vx}}}{n}$
`evzm=evz/di11`	Averages the ϵ_{vy} emittances	$\overline{\epsilon_{vy}} = \frac{\overline{\epsilon_{vy}}}{n}$
`evtm=evt/di11`	Averages the ϵ_{vt} emittances	$\overline{\epsilon_{vt}} = \frac{\overline{\epsilon_{vt}}}{n}$

- Performs the smear and 4-D smear calculations. Starts a new loop retrieving the particles parameters since the beginning.
- Obtains the mean emittances[2].
- Initializes the Courant-Snyder array with the coordinates from the first particle[2].
- Ports the Courant-Snyder array to Canonical Variables[2].
- Computes the horizontal and vertical emittances with linear coupling[2].
- Multiplies the Courant-Snyder array by `rbeta`[2].
- Obtains the transverse emittance and the smear for all three dimensions:

Code	Description	Formula
`evt=evx+evz`	Computes ϵ_{vt}	$\epsilon_{vt} = \epsilon_{vx} + \epsilon_{vy}$
`sevx=sevx+(evx-evxm)**2`	Horizontal smear	$sevx_{i-1} + (\epsilon_{vx_i} - \overline{\epsilon_{vx_i}})^2$
`sevz=sevz+(evz-evzm)**2`	Vertical smear	$sevz_{i-1} + (\epsilon_{vx_i} - \overline{\epsilon_{vz_i}})^2$
`sevt=sevt+(evt-evtm)**2`	Transverse smear	$sevt_{i-1} + (\epsilon_{vt_i} - \overline{\epsilon_{vtm_i}})^2$

- Calls the subroutine `sinpro(a,b,c,d,e)` to get the smear in percentage. This function expresses d and e in terms of percentages of a.

Call
`call sinpro(emxa,di11,emxs,emax,emix)`
`call sinpro(emza,di11,emzs,emaz,emiz)`
`call sinpro(emta,di11,emts,emat,emit)`
`call sinpro(evxm,di11,sevx,evxma,evxmi)`
`call sinpro(evzm,di11,sevz,evzma,evzmi)`
`call sinpro(evtm,di11,sevt,evtma,evtmi)`

[2] Formulas and variables involved dissected in previous subsections

Printing

Then it proceeds to build a summary of the post-processing to write it in the `fort.10` file (See Tables 3.1 and 3.2).

- Gets the difference in the number of turns per data entry (`tidnt`) and saves columns 2,10,11,22 and 23 for all the rows in the `fort.10` output file.

Variable	Content	Alternative content
`sumda(22)`	`dble(nnumxv(ifipa))`	`dble(ia)`
`sumda(23)`	`dble(nnumxv(ilapa))`	`dble(ia)`
`sumda(2)`	`dble(nlost)`	`dble(nlost)`
`sumda(10)`	`biav(i2-1)`	`biav(i2-1)`
`sumda(11)`	`slope(i2-1)`	`slopem`

Variable	Description
`sumda(22)`	Number of survived turns of the first particle
`sumda(23)`	Number of survived turns of the twin particle
`sumda(2)`	Stability Flag (0=stable, 1=lost)Number of lost particles
`sumda(10)`	Final distance in phase space
`sumda(11)`	Maximum slope of distance in phase space

- Performs the calculation of the averaged phase-advances and saves the following columns to the variables to write on the `fort.10` file:

Variable	Content	Description
`sumda(12)`	`tphx-qwc(1)`	Horizontal detuning
`sumda(13)`	`sdpx`	Spread of horizontal detuning
`sumda(14)`	`tphz-qwc(2)`	Vertical detuning
`sumda(15)`	`sdpz`	Spread of vertical detuning
`sumda(25)`	`tph6`	Synchrotron tune

Variable	Content	Formula
sumda(12)	tphx-qwc(1)	$\overline{\psi}_x - Q_x$
sumda(13)	sdpx	$\sigma_{\overline{\psi}_x - Q_x} = \sqrt{\frac{\sum_{i=1}^{iapx}(\phi_{x,i} - \overline{\psi}_x)^2}{iapx}}$
sumda(14)	tphz-qwc(2)	$\overline{\psi}_y - Q_y$
sumda(15)	sdpz	$\sigma_{\overline{\psi}_y - Q_y} = \sqrt{\frac{\sum_{i=1}^{iapy}(\phi_{y,i} - \overline{\psi}_y)^2}{iapy}}$
sumda(25)	tph6	

- Obtains the distance of the *Q-values* (averaged phase-advances) to the resonances and saves the columns:

Variable	Content	Description
sumda(16)	dble(im1s)	Horizontal factor to nearest resonance
sumda(17)	dble(jm1s)	Vertical factor to nearest resonance
sumda(18)	sumda(16)+sumda(17)	Order of nearest resonance

- Computes the *Q-values* by an FFT routine. Calls the soubroutine

  `fft(xxr,xxi,ifp,ife)`,

  where `xxr` is the sequence of x positions to analyze and decompose in the complex array given by `xxr, xxi`. `ifp` and `ife` are the start and the end of the interval where the analysis is performed. The same is done for the vertical positions. Then it stores the maximum and minimum values of the tune interval for the Horizontal and vertical tune:

Variable	Description
pmax(21)	Maximum values of the horizontal tune interval
pmin(21)	Minimum value of the horizontal tune interval
pmax(23)	Maximum value of the vertical tune interval
pmin(23)	Minimum value of the vertical tune interval

  This values depend on the variable `ifh`:

$$ifh = 0 : 0 \leq Q \leq 1$$
$$ifh = 0 : 0 \leq Q \leq 0.5$$
$$ifh = 00.5 \leq Q \leq 1$$

  Finally, the maximum and minimum values of the norm for each number of the arrays is computed in each of the extremes of the interval given by `ifh`.

Variable	Description
xxmax	Maximum norm of the numbers in the xx complex array
zzmax	Maximum norm of the numbers in the xx complex array
xxmin	Minimum norm of the numbers in the xx complex array
zzmin	Minimum norm of the numbers in the xx complex array

Variable	Content	Formula
xxmax	max(xxmax,sqrt(xxr(i)**2+xxi(i)**2))	$\max\sqrt{xx^2+xxi^2}$
zzmax	max(zzmax,sqrt(zzr(i)**2+zzi(i)**2))	$\max\sqrt{zz^2+zzi^2}$
xxmin	min(xxmin,sqrt(xxr(i)**2+xxi(i)**2))	$\min\sqrt{xx^2+xxi^2}$
zzmin	min(zzmin,sqrt(zzr(i)**2+zzi(i)**2))	$\min\sqrt{zz^2+zzi^2}$

- Computes the distance of the $Q-values$ (this time the ones generated by the FFT routine) to the resonances.

- Prints the 4-D invariants with linear coupling: writes on the screen the values of emi, emii, emiii,angi,angii, angiii, evxm, sevx, evxma, evxmi, evzm,sevz, evzma, evzmi, evtm, sevt, evtma and evtmi.

- Saves the emittances and the smear:

Variable	Description
`sumda(46)`	Emittance Mode I
`sumda(47)`	Emittance Mode II
`sumda(48)`	Secondary horizontal β-function
`sumda(49)`	Secondary vertical β-function
`sumda(7)`	Horizontal amplitude 1st particle
`sumda(8)`	Vertical amplitude 1st particle
`sumda(26)`	Horizontal amplitude 2nd particle
`sumda(27)`	Vertical amplitude 2nd particle
`sumda(19)`	Horizontal smear
`sumda(20)`	Vertical smear
`sumda(21)`	Transverse smear
`sumda(59)`	The number of the Random Set
`sumda(24)`	Starting seed for random generator
`sumda(28)`	Minimum horizontal amplitude
`sumda(29)`	Mean horizontal amplitude
`sumda(30)`	Maximum horizontal amplitude
`sumda(31)`	Minimum vertical amplitude
`sumda(32)`	Mean vertical amplitude
`sumda(33)`	Maximum vertical amplitude
`sumda(34)`	Minimum horizontal amplitude (linear decoupled)
`sumda(35)`	Mean horizontal amplitude (linear decoupled)
`sumda(36)`	Maximum horizontal amplitude (linear decoupled)
`sumda(37)`	Minimum vertical amplitude (linear decoupled)
`sumda(38)`	Mean vertical amplitude (linear decoupled)
`sumda(39)`	Maximum vertical amplitude (linear decoupled)
`sumda(40)`	Minimum horizontal amplitude (nonlinear decoupled)
`sumda(41)`	Mean horizontal amplitude (nonlinear decoupled)
`sumda(42)`	Maximum horizontal amplitude (nonlinear decoupled)
`sumda(43)`	Minimum vertical amplitude (nonlinear decoupled)
`sumda(44)`	Mean vertical amplitude (nonlinear decoupled)
`sumda(45)`	Maximum vertical amplitude (nonlinear decoupled)

Variable	Content
sumda(46)	emi
sumda(47)	emii
sumda(48)	bet0x2
sumda(49)	bet0z2
sumda(7)	sqrt(bet0(1)*emi)+sqrt(bet0x2*emii)
sumda(8)	sqrt(bet0(2)*emii)+sqrt(bet0z2*emi)
sumda(26)	sqrt(bet0(1)*evx2)+sqrt(bet0x2*evz2)
sumda(27)	sqrt(bet0(2)*evz2)+sqrt(bet0z2*evx2)
sumda(19)	sevx
sumda(20)	sevz
sumda(21)	sevt
sumda(59)	dnms
sumda(24)	dizu0
sumda(28)	sqrt(bet0(1)*abs(emix))
sumda(29)	sqrt(bet0(1)*emxa)
sumda(30)	sqrt(bet0(1)*emax)
sumda(31)	sqrt(bet0(2)*abs(emiz))
sumda(32)	sqrt(bet0(2)*emza)
sumda(33)	sqrt(bet0(2)*emaz)
sumda(34)	sqrt(bet0(1)*abs(evxmi))
sumda(35)	sqrt(bet0(1)*evxm)
sumda(36)	sqrt(bet0(1)*evxma)
sumda(37)	sqrt(bet0(2)*abs(evzmi))
sumda(38)	sqrt(bet0(2)*evzm)
sumda(39)	sqrt(bet0(2)*evzma)
sumda(40)	sqrt((bet0(1)*abs(evtmi))*ratemx)
sumda(41)	sqrt((bet0(1)*evtm)*ratemx)
sumda(42)	sqrt((bet0(1)*evtma)*ratemx)
sumda(43)	sqrt((bet0(2)*abs(evtmi))*ratemz)
sumda(44)	sqrt((bet0(2)*evtm)*ratemz)
sumda(45)	sqrt((bet0(2)*evtma)*ratemz)

Variable Formula	
`sumda(46)`	$\epsilon_I = \left(x \sum_{i=0}^{6} \frac{\partial z_i}{\partial x}\right)^2 + \left(\widetilde{x'} \sum_{i=0}^{6} \frac{\partial z_i}{\partial \widetilde{x'}}\right)^2$
`sumda(47)`	$\epsilon_{II} = \left(y \sum_{i=0}^{6} \frac{\partial z_i}{\partial y}\right)^2 + \left(\widetilde{y'}^2 \sum_{i=0}^{6} \frac{\partial z_i}{\partial \widetilde{y'}}\right)^2$
`sumda(48)`	$\beta_{xII} = {T_{1,3}}^2 + {T_{1,4}}^2$
`sumda(49)`	$\beta_{yII} = {T_{3,1}}^2 + {T_{3,2}}^2$
`sumda(7)`	$\sqrt{\beta_{xI} * \epsilon_I} + \sqrt{\beta_{xII} * \epsilon_{II}}$
`sumda(8)`	$\sqrt{\beta_{yI} * \epsilon_I} + \sqrt{\beta_{yII} * \epsilon_{II}}$
`sumda(26)`	$\sqrt{\beta_{xI} * \epsilon_{vx2}} + \sqrt{\beta_{xII} * \epsilon_{vy2}}$
`sumda(27)`	$\sqrt{\beta_{yI} * \epsilon_{vy2}} + \sqrt{\beta_{yII} * \epsilon_{vx2}}$
`sumda(19)`	$sevx_i = sevx_{i-1} + (\epsilon_{vx_i} - \overline{\epsilon_{vx_i}})^2$
`sumda(20)`	$sevz_i = sevz_{i-1} + (\epsilon_{vx_i} - \overline{\epsilon_{vz_i}})^2$
`sumda(21)`	$sevt_i = sevt_{i-1} + (\epsilon_{vx_i} + \epsilon_{vz_i} - \overline{\epsilon_{vt_i}})^2$
`sumda(59)`	
`sumda(24)`	
`sumda(28)`	$\sqrt{\beta_{yI} * abs(\min \epsilon_x)}$
`sumda(29)`	$\sqrt{\beta_{xI} * \overline{\epsilon_x}}$
`sumda(30)`	$\sqrt{\beta_{xI} * \max \epsilon_x}$
`sumda(31)`	$\sqrt{\beta_{yI} * abs(\min \epsilon_y)}$
`sumda(32)`	$\sqrt{\beta_{yI} * \overline{\epsilon_y}}$
`sumda(33)`	$\sqrt{\beta_{yI} * \max \epsilon_z}$
`sumda(34)`	$\sqrt{\beta_{yI} * abs(\min \epsilon_{vx})}$
`sumda(35)`	$\sqrt{\beta_{xI} * \overline{\epsilon_{vx}}}$
`sumda(36)`	$\sqrt{\beta_{xI} * \max \epsilon_{vx}}$
`sumda(37)`	$\sqrt{\beta_{yI} * abs(\min \epsilon_{vy})}$
`sumda(38)`	$\sqrt{\beta_{yI} * \overline{\epsilon_{vz}}}$
`sumda(39)`	$\sqrt{\beta_{yI} * \max \epsilon_{vy}}$
`sumda(40)`	$\sqrt{(\beta_{xI} * abs(\min \epsilon_{vt})) * \frac{\overline{\epsilon_x}}{\overline{\epsilon_x} + \overline{\epsilon_y}}}$
`sumda(41)`	$\sqrt{(\beta_{xI} * \overline{\epsilon_{vt}}) * \frac{\overline{\epsilon_x}}{\overline{\epsilon_x} + \overline{\epsilon_y}}}$
`sumda(42)`	$\sqrt{(\beta_{xI} * \max \epsilon_{vt}) * \frac{\overline{\epsilon_x}}{\overline{\epsilon_x} + \overline{\epsilon_y}}}$
`sumda(43)`	$\sqrt{(\beta_{yI} * abs(\min \epsilon_{vt})) * \frac{\overline{\epsilon_y}}{\overline{\epsilon_x} + \overline{\epsilon_y}}}$
`sumda(44)`	$\sqrt{(\beta_{yI} * \overline{\epsilon_{vt}}) * \frac{\overline{\epsilon_y}}{\overline{\epsilon_x} + \overline{\epsilon_y}}}$
`sumda(45)`	$\sqrt{(\beta_{yI} * \max \epsilon_{vt}) * \frac{\overline{\epsilon_y}}{\overline{\epsilon_x} + \overline{\epsilon_y}}}$

- Stores the chromaticity values in the correct units in the variables to write in the `fort.10` file.

Variable	Content	Description	Formula
`sumda(50)`	`chromc(1)*c1e3`	$Q'x$	Q'_x
`sumda(51)`	`chromc(2)*c1e3`	$Q'y$	Q'_y

- Writes the data for the summary of the post-processing, the `sumda` vector, in the `fort.10` file.
- Performs the calculation of the invariances of the 4-D transverse motion[3].

Plotting

Finally, it generates several kinds of plots and writes the summary built in the previous section in the `fort.10` file.

- Sets up the variables for plotting
- Calls several functions for the generation of histogram and other kind of plots (1/2).
- Works out the detection of the onset of chaotic motion and thereby the long-term dynamic aperture by evaluating the Lyapunov exponent.
- Converts the results to canonical variables.
- Calls several functions for the generation of histogram and other kind of plots (2/2).
- Writes the data for the summary of the post-processing on the `fort.10` file.
- Rewinds the used files.
- Obtains the time count.

3.4 From SixTrack to SixDesk's Post-Processing

SixDesk, the running environment for SixTrack, will take the results of the analysis made with SixTrack and will apply its own post-processing.

[3]Look above for the calculations of the invariants for a thoroughly understanding of this section

By default, SixDesk's scripts automatically tar all of the Sixtrack run output files and store them in CASTOR [15]. Between this tars can be found the file `fort.6`, which describes all operations made and the possible failures that may have occurred during the SixTrack run; or the binary files `fort.90`, `fort.89`,..., `fort.59`, used for further analysis and properly described in SixTrack's manual [18].

As for the main output file, `fort.10`, running the script `run_join10` gathers the results of the completed jobs and combines the output files. After this, the combined `fort.10` files can be post-processed to find chaotic boundaries and particle losses, as will be described in detail in the following chapter.

SixDesk also provides a couple of options for plotting, defined by the `sixdeskenv` variables `iplot` and `kvar` specific to `run_post`. If `iplot=1` a plot is produced for each seed and the results can be found in the plot directory. Setting iplot to 1 may produce a huge amount of data, even if it is compressed with gzip. The variable `kvar` should be set to 1 (the default) to obtain the dynamic aperture as a function of angles for a long study, and the dynamic aperture over all seeds and angles is plotted for each angle even if `iplot=0`. Notice that this is an extremely time consuming procedure. For special post-processing of the fort.10.gz files, the set of `fort.15` files are produced and saved when `iplot = 1`.

Chapter 4

A Post-Processing Module for SixDeskDB

SixDesk has been a working environment for Sixtrack for the last 25 years. During this time it has proved to be a great and poweful tool to tune SixTrack for the study of accelerator physics and even more with the appearance of the LHC@Home platform.

However, the rise of computer power has led to reaching computational limitations and new advances in physics push the need for adding new studies and physic models to SixDesk's current analyses. This led to the implementation of SixDeskDB: a database-centered Python and SQL port of SixDesk.

This Chapter will be focused in the main post-processing studies implementation port from SixDesk to this new implementation.

Section 4.1 introduces SixDesk's new database-centered implementation, while Section 4.2 summarizes SixDesk's post-processing run and its structure.

Then, the process of design and implementation of both the post-processing scripts for SixDeskDB is described in Section 4.3.

Finally, in Section 4.4 the results and plots generated are discussed.

4.1 SixDeskDB: SixDesk's Database Port

During 2014, and specially during summer 2014 with the help of Google's Summer of Code[1] provided students, a database-centered Python port for SixDesk was developed.

Using this database approach has several benefits:

- **Reduces Complexity**: Replaces a design consisting in several interdependent Bash, AWK, GNUPlot and Fortran scripts, with a Python implementation using SQL for queries.
- **Solves AFS limitations**: AFS can hold, at maximum, around two thousand entries in a single directory, but longer SixTrack simulations demand more entries for all the files generated. Making use of a database as a file or data server removes this limitations.
- **No need to know Fortran**: Implemented with Python and SQL, which are languages less complex and more flexible and widely used than Fortran.
- **Improves execution times**: Removing all operations that require to write on disk a huge amount of files improves dramatically the execution time.
- **Introduces Object-Oriented programming**: and its benefits (and trade-offs) applied to SixTrack.
- **Adds Flexibility**: The generation of all files and plots in a rigid file-system tree is no longer required and can be accessed on request, which helps manipulating them and allows more flexibility at the time of extending or changing the code.
- **Allows the use of modern, popular, state-of-the-art scientific libraries**: The use of the popular fast-growing Numpy and Scipy libraries provides and without as much impact as expected in the performance while substituting Fortran with an interpreted language like Python.

On the other hand, it adds a new language needed to work with SixTrack and it will require a future adaptation for its use with the LHC@Home platform. An

[1] Google Summer of Code's project: SixDesk library for managing massive SixTrack simulations http://www.google-melange.com/gsoc/project/details/google/gsoc2014/monisjaved/5649050225344512

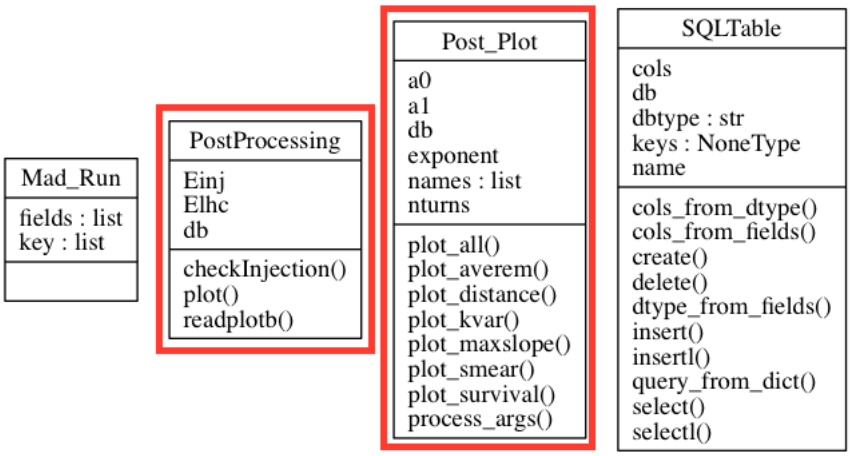


Figure 4.1: Classes that conform the new SixDeskDB (1/4). The classes and functions this project has been involved in are marked in red.

overview of the classes that conform the program is shown from Figure 4.1 to Figure 4.4.

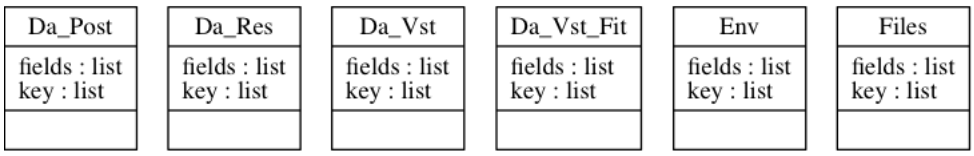


Figure 4.2: Classes that conform the new SixDeskDB (2/4). The classes and functions this project has been involved in are marked in red.

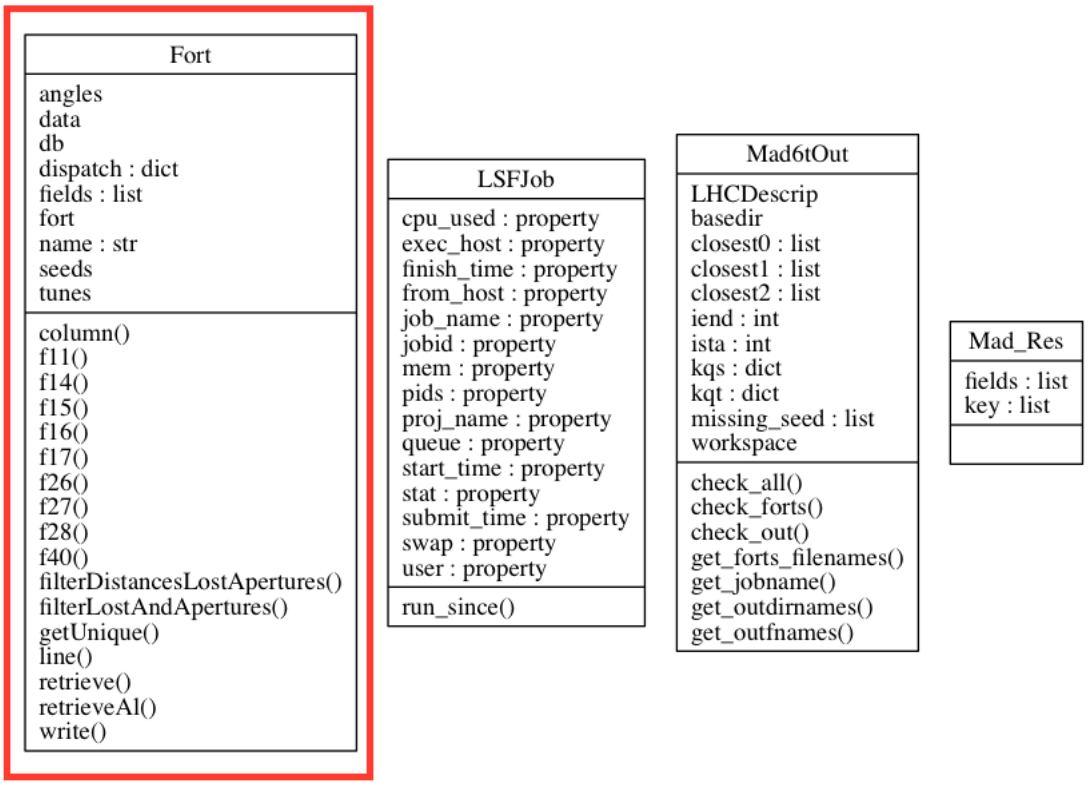


Figure 4.3: Classes that conform the new SixDeskDB (3/4). The classes and functions this project has been involved in are marked in red.

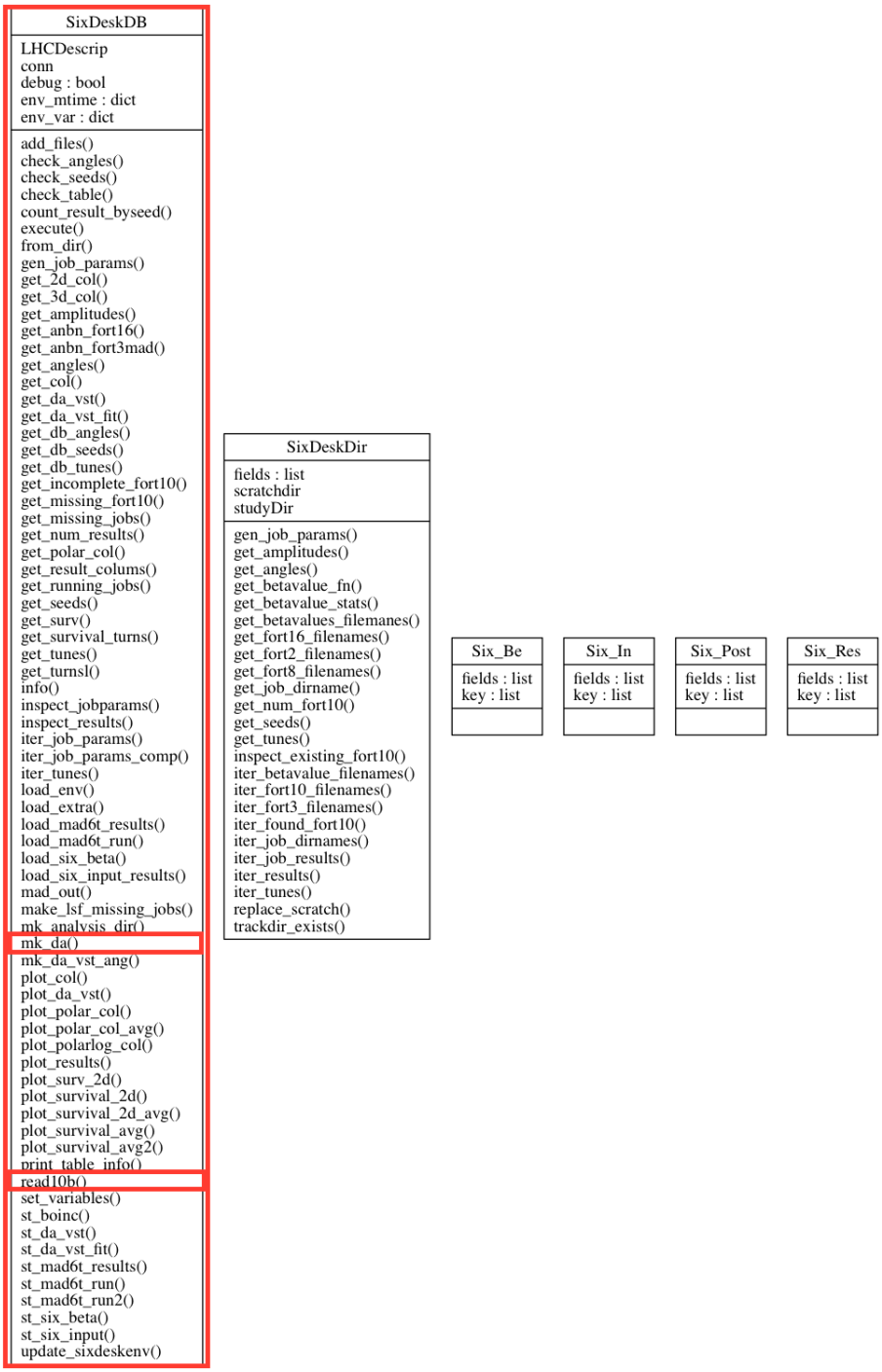


Figure 4.4: Classes that conform the new SixDeskDB (4/4). The classes and functions this project has been involved in are marked in red.

4.2 The Post-Processing Run

SixDesk post-processing run makes use of several interdependent Bash, AWK, GNU-Plot and Fortran scripts alongside a series of configuration files and SixTrack's tracking output files.

The structure would be the shown in Figure 4.5. The main script to run is `run_post`, after running `run_join10` and `run_awk`, which will prepare on request the files taken as an input for their manipulation.

The important functions to be called during the post-processing runs are `read10b` and `readplotb`, where the core of the post-processing calculations are made. This are the Fortran functions to include in SixDeskDB, integrating into them the management and looping done now between several Bash scripts.

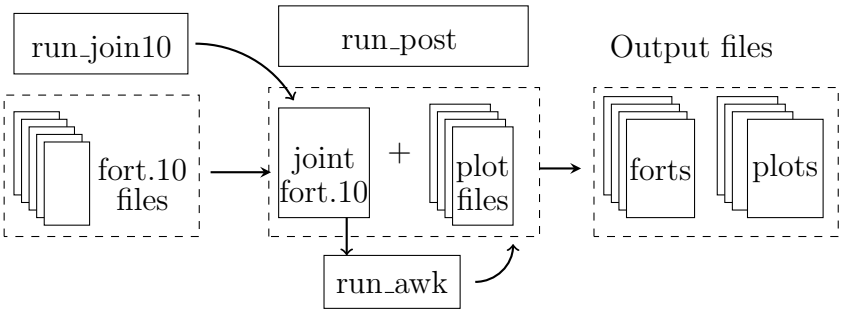


Figure 4.5: SixDesk `run_post` process with plots. First, `run_join10` is run to combine all the `fort.10` files into one. Then `run_awk` will take the joint `fort.10` file as an imput and parse it to generate the files needed for the plot. Finally `run_post` will run with the generated files to produce SixDesk's output files and the actual plots.

After the post-processing run, the results are obtained in a file tree structure as shown in Figure 4.6. Some of the binary output files with further information of the process will be stored automatically in CASTOR [15].

Notice how the mentioned typical SixTrack run for a DA study in the LHC involving the simulation for 5 different angles in the phase-space and 60 seeds would exceed then AFS limitations fast and it will take a long time to compute and write on disk. Hence the need for a new implementation.

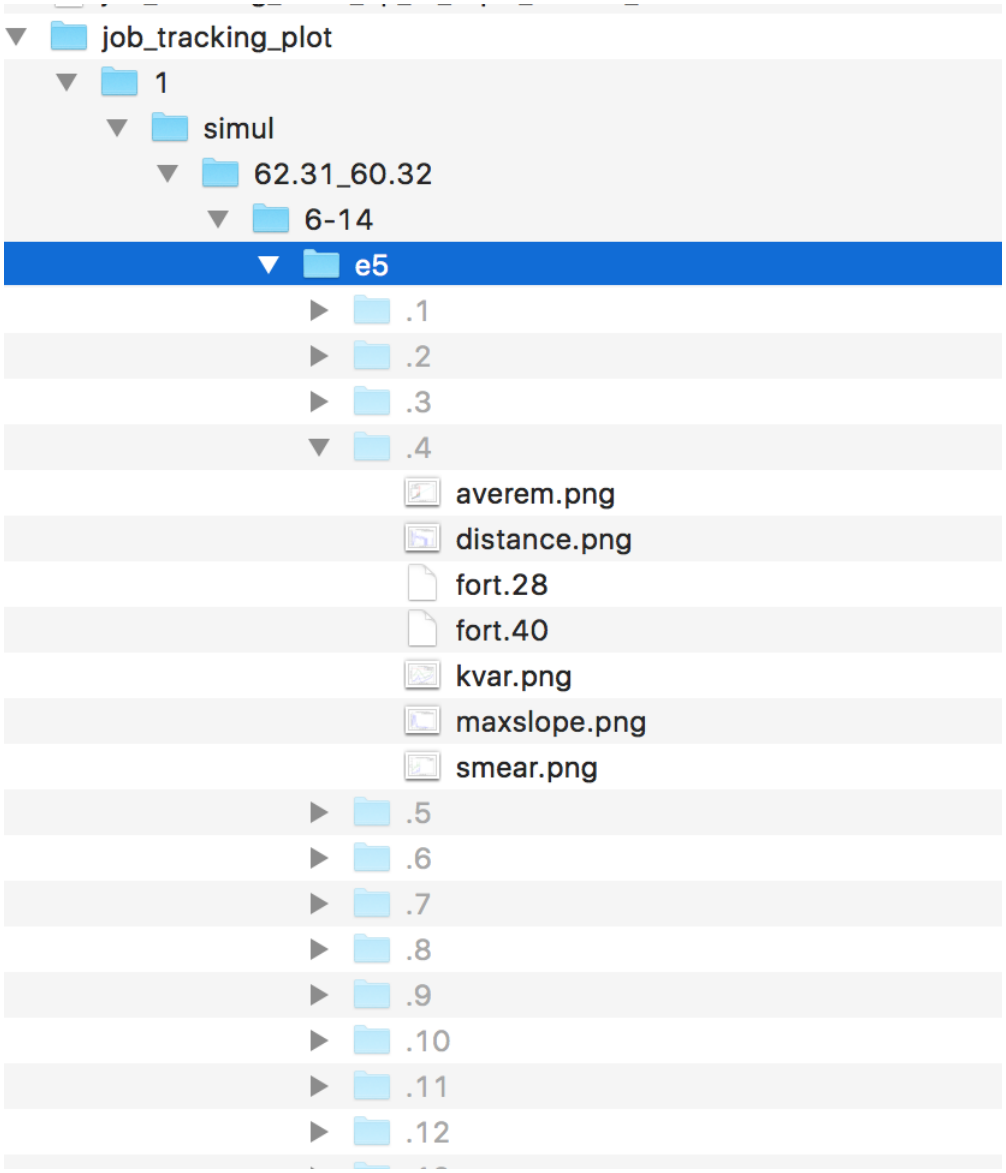


Figure 4.6: SixDesk's output file tree for a long run. The output files are generated in a path following the stucture `/<seed>/simul/<tune range>/<amplitude range>/<order of the number of turns>/<angle>/outputFiles`.

4.3 Design and Implementation

The code developed aims to fully reproduce the physics part of the post-processing run of SixDesk, which corresponds to the code in `read10b` and `readplotb` Fortran files and its launching/looping Bash scripts. That means, for example, performing dynamic aperture analysis or finding particle losses and chaotic boundaries. For this, the first decision taken is to move this analysis out of the main class, in a class of its own and rethinking the software architecture.

It must be taken into observation that this is still an unfinished project. Everything described here answers to the state of the project at this moment. Once finished, this new class will implement SixDesk's post-processing in a modularized way, including the plotting capabilities, both for the short and the long run.

A pseudo-dependencies diagram of the relevant SixDeskDB classes for this project is exposed in Figure 4.7.

There's some interesting things to see in how this classes work and their interrelation:

- **SixDeskDB:** SixDesk's python implementation main class. It does almost everything SixDesk can do, and although at this stage its role is not to do all the operations but to be in charge of the reading of the input files, OS and file management and to serve as an interface for the user to run each one of the different modules and analysis SixDesk's provide. This project works with the `mk_da` function (it stands for make dynamical aperture analysis), that will invoke the post-processing run. The`read10b` function of this class remains to be ported to the PostProcessing class.
- **Fort:** A new, auxiliary class that serves as an interface to retrieve the different Fort files from the database, hiding the SQL queries from the physics code (and even from the `Fort` class, as they are defined in the `queries.py` file) and providing a better and more intuitive way to explore the data in the files: each one of the fields of the data for each `fort` file is named for ease of access, filtering and manipulation. This way of working with the database has been fully integrated in the PostProcessing and Post_Plot classes and it is making its way to the SixDeskDB class.

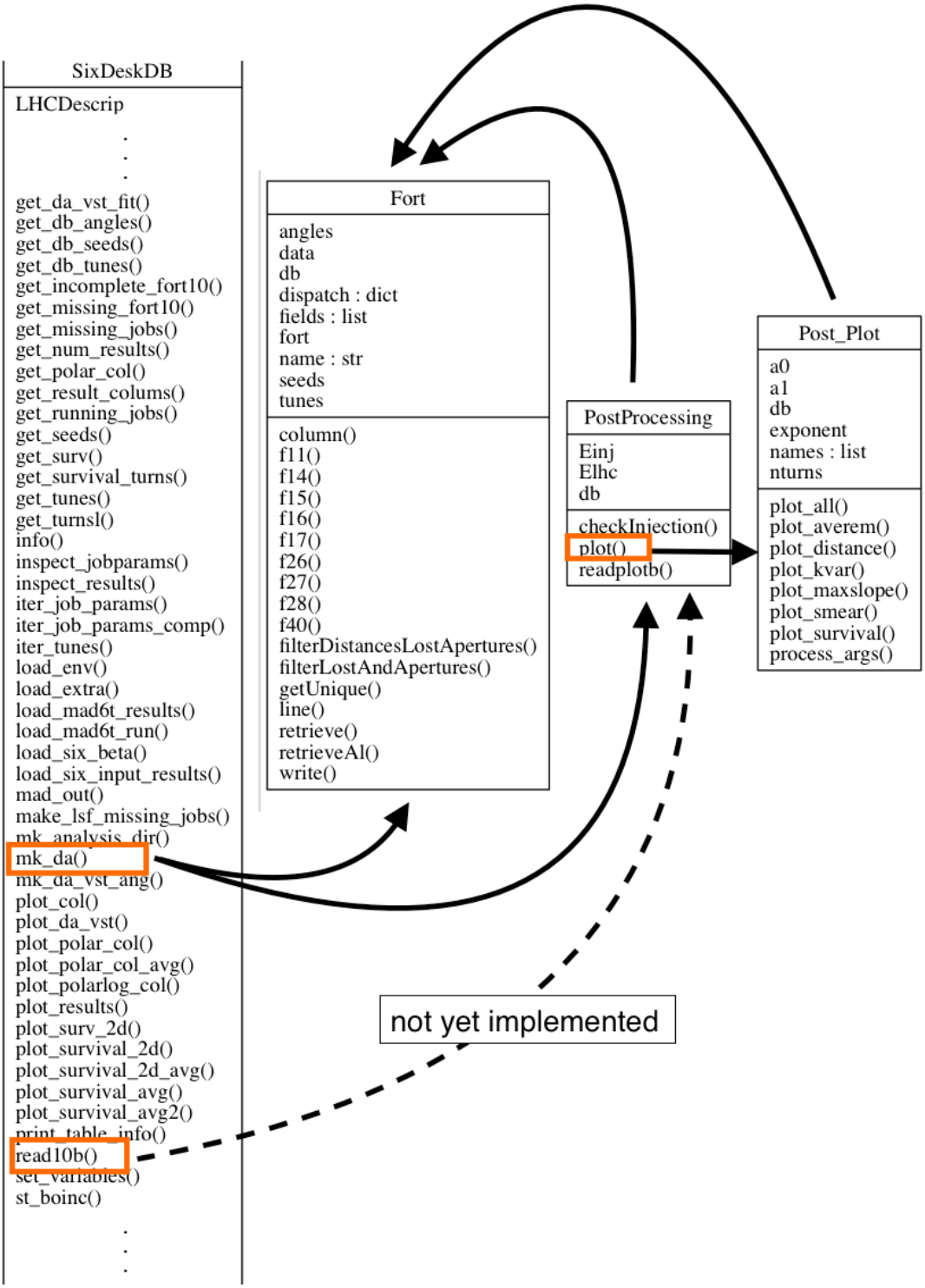


Figure 4.7: Dependencies between classes and functions in this project's implementation of the post-processing run.

- **PostProcessing:** Class in charge of the physics computations and the generation of the output/results `fort` files. Currently it implements SixDesk's `readplotb` function.
- **Post_Plot:** Plotting class. Defines the mechanisms needed to generate the plots (just for long runs at the moment) and answers to `PostProcessing`'s `plot()` call. A description of the plots it generates will be exposed in Section 4.4.

The described structure adds another layer of complexity to SixDeskDB, but in return provides the following advantages:

- **Fort files on request**: The `Fort` class implemented allows to obtain all the Fort files generated on-request indexed by angle and seed, with no need of writing redundant files on disc. A reference for what is contained in each `fort` file and what each variable written on them stands for can be found in Tables 4.1 and 4.2 respectively. It also provides a way to obtain all post-processing intermediate files that were deleted in the original `run_post` execution.
- **Plots on Request**: Instead of generating all plots during the run we access them on-request. This eliminates unnecessary information.
- **Clarity**: Python characteristics such as the mandatory indentation, specially if used with object-oriented programming, and the use of just one language to code (and SQL to query) makes the code easier to structure, maintain and understand.
- **Easily modifiable queries**: Having the queries for the `fort` files data retrieving and generation in a separate file allows a faster modification an manipulation of their results and makes it easier to implement the generation of a new one.
- **Control over the output**: The `Fort` class provides an interface to access the data from SixDesk's Fortran-generated files structured in the same way as SixDesk's output, adding useful features like variable indexing on top of that.
- **Faster manipulation and scripting**: Features like the clarity and modularization of the code, having an interface to obtain plots and `fort` files on request with variable indexing or just the fact of having a database with all the information stored makes it easier to gather the data needed in a structured way for simplifying manipulation of the code and the scripting to work with its functions and modules.

File	Columns
fort.11	achaos, al, amin, amax, achaos1
fort.12	rad, distp
fort.13	rad, dist
fort.14	achaos, alost3, turn_max, f14
fort.15	rad, sturns1, sturns2
fort.16	deltap, qx, qy
fort.17	deltap, qx, qy
fort.18	rad, smearx
fort.19	rad, smeary
fort.20	rad, qx_det
fort.21	rad, qy_det
fort.22	rad, rad1*sigxminnld
fort.23	rad, rad1*sigxavgnld
fort.24	rad,rad1*sigxmaxnld
fort.25	qx_det,qy_det,qx_det+qx,qy_det+qy
fort.26	achaos, alost2, amax
fort.27	al
fort.28	al
fort.40	achaos, al amin, amax

Table 4.1: Reference table for the output `fort` files and the variables tracked on them.

Column	Description
achaos	the chaotic boundary with slope method
achaos1	distance in the phase space with slope method
al	aperture lost
amin	minimum aperture
amax	maximum aperture
rad	ratio between ϵ_I/ϵ[a]
rad1	same quantity as rad but evaluated differently[b]
dist	distance in the phase-space
distp	maximum slope of distance in the phase-space
alost1	average initial condition, average corrected, of a lost particle
alost2	initial condition of lost particle
alost3	
turn_max	maximum number of turns
f14	
sturns1	survived turns of the first particle
sturns2	survived turns of the second particle
deltap	relative momentum deviation
qx	horizontal tune
qy	vertical tune
qx_det	horizontal detuning
qy_det	vertical detuning
qx_det+qx	
qy_det+qy	
smearx	horizontal smear
smeary	vertical smear
rad1*sigxminnld	
rad1*sigxavgnld	
rad1*sigxmaxnld	

[a] ratio between ϵ_I/ϵ (or ϵ_{II}/ϵ if `rat` > 1) if $\epsilon = \epsilon_I^2 + \epsilon_{II}^2$ normalized with the real emittance of the beam specified by the user. If emitI is small then emitI is estimated using the initial conditions;

[b] `rad1` evaluate as `rad` using the average amplitudes if emitI is small otherwise using the average amplitudes and the secondary beta functions

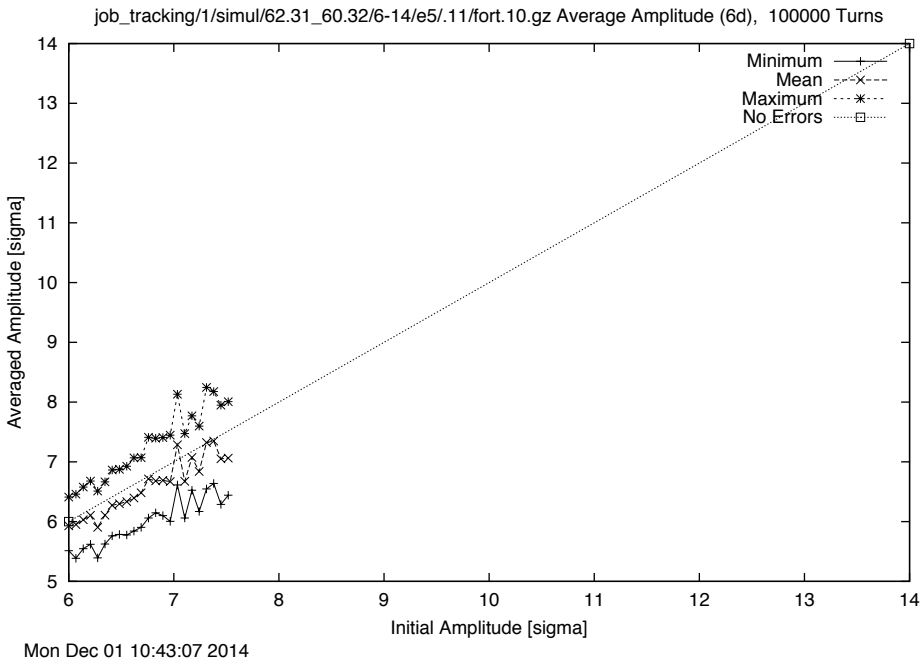
Table 4.2: Output `fort` files column names (variables) reference.

4.4 Results and Plotting

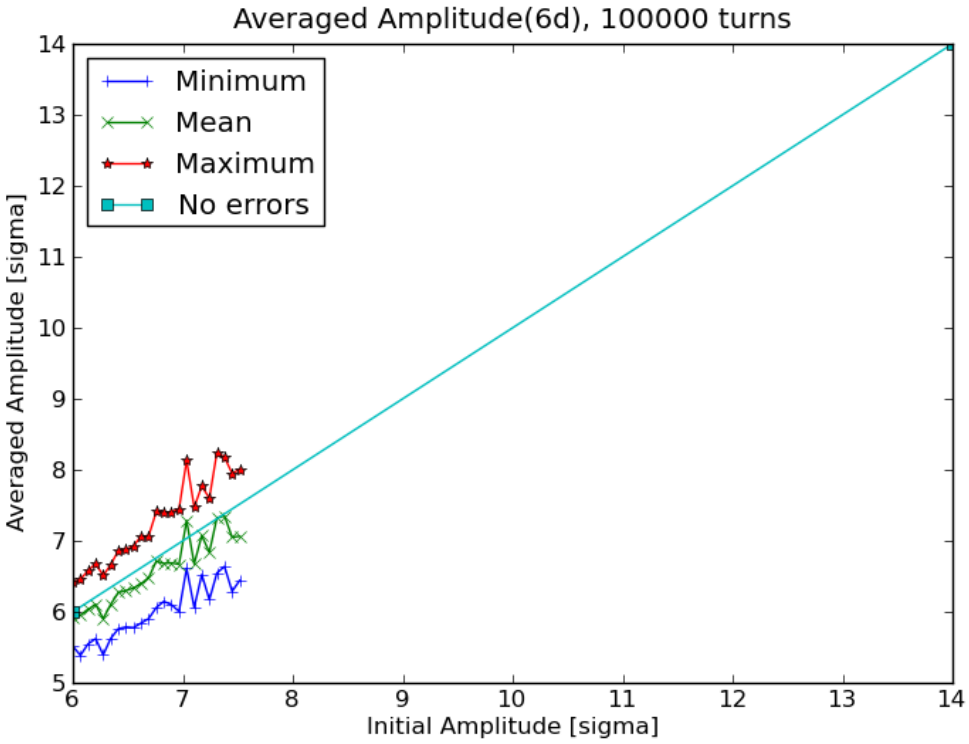
All the results have been proved to match the originals up to a precision of ten decimals, which is more than enough taking into account that the values are being stored as a string in the database.

With the results generation, a plotting module has been developed for their visualization, aiming to replace the entangled web of Bash and AWK scripts that conformed SixDesk's plotting functionality.

All the plots from the long run have been reproduced, matching the originals for each one of the cases. This also shows the accuracy of the results. Examples for each one of the plots to generate will be shown from Figure 4.8 to Figure 4.13. It is needed to point out that due to a bug in the Matplotlib library for the Numpy version used at CERN, the logarithmic scales for the axis can't be adopted.

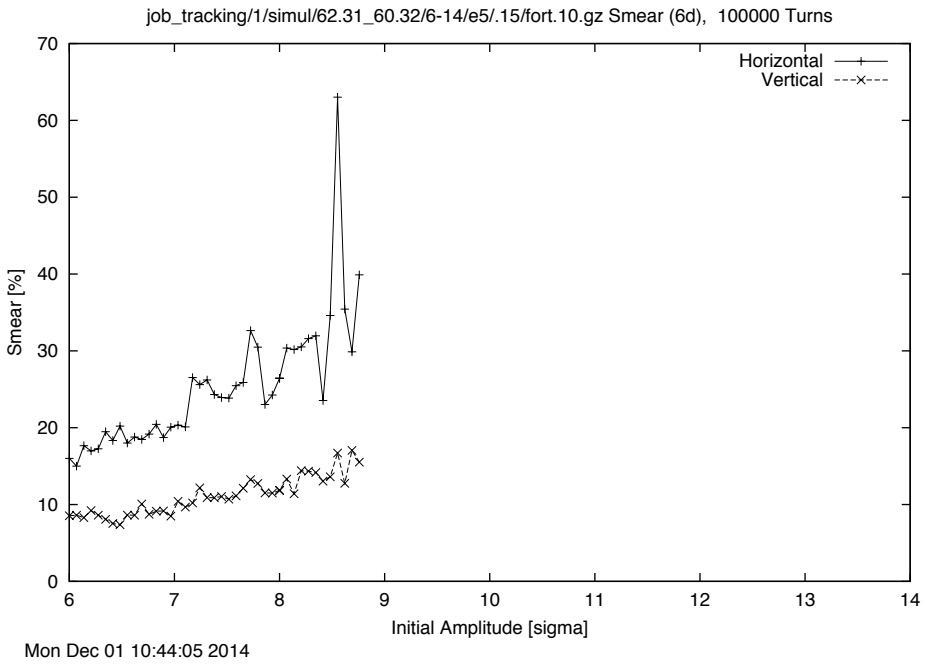


(a) SixDesk

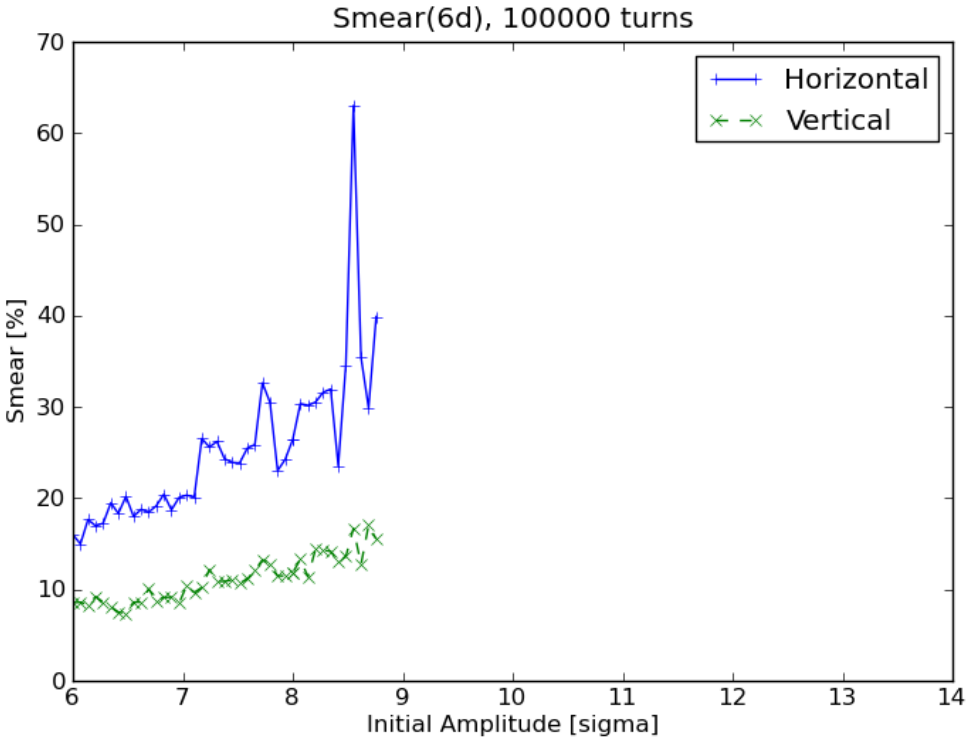


(b) SixDeskDB

Figure 4.8: **Average emmittance.** `seed=1`, `angle=15`.

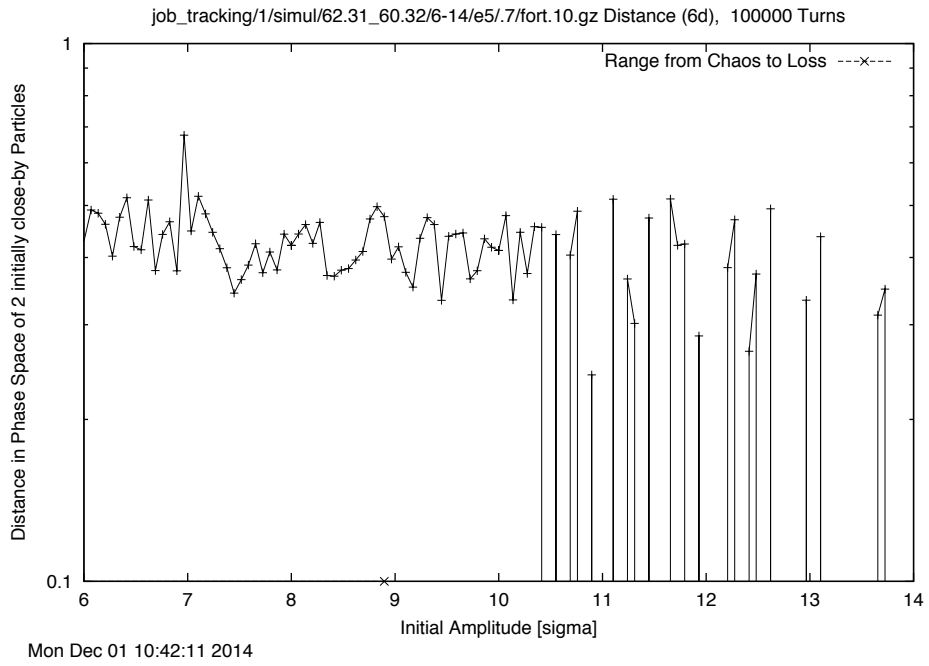


(a) SixDesk

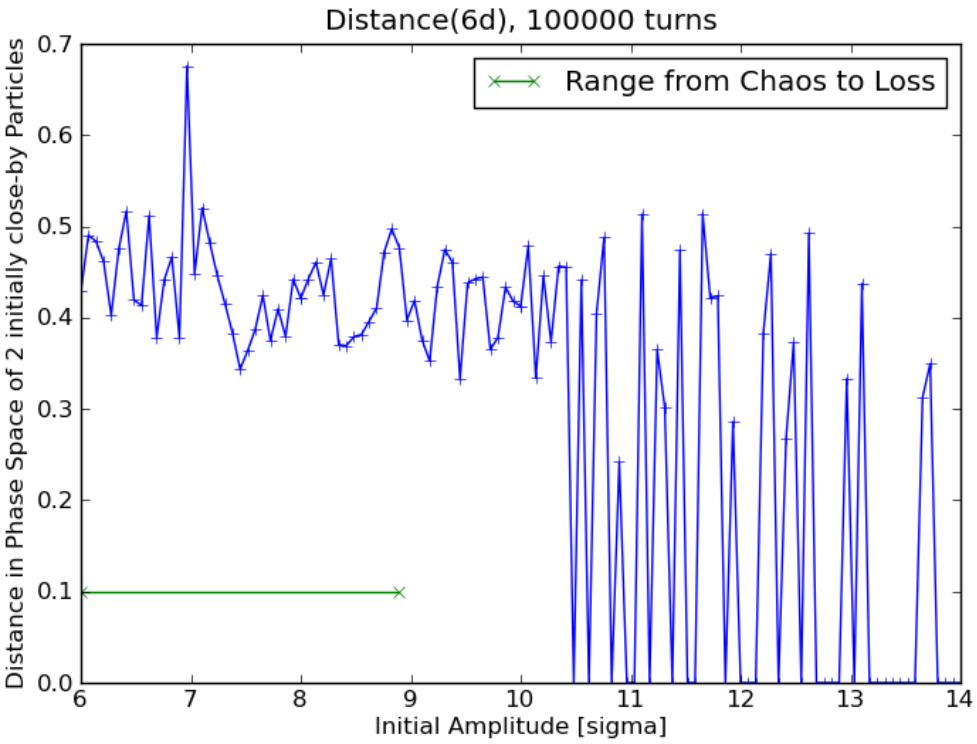


(b) SixDeskDB

Figure 4.9: **Smear in %.** Horizontal and vertical smear as a function of initial amplitude. `seed=1`, `angle=15`.

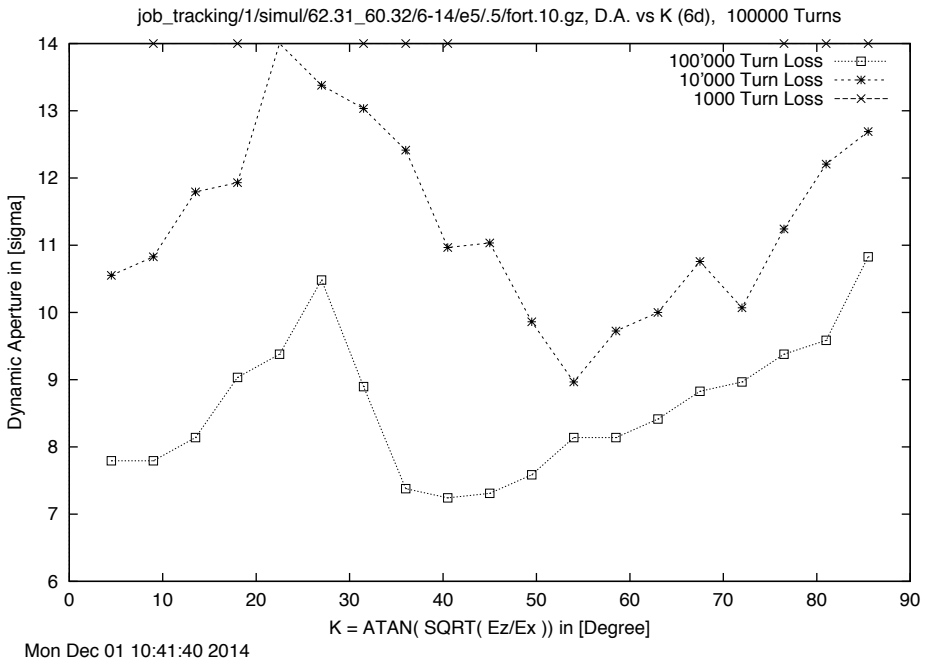


(a) SixDesk

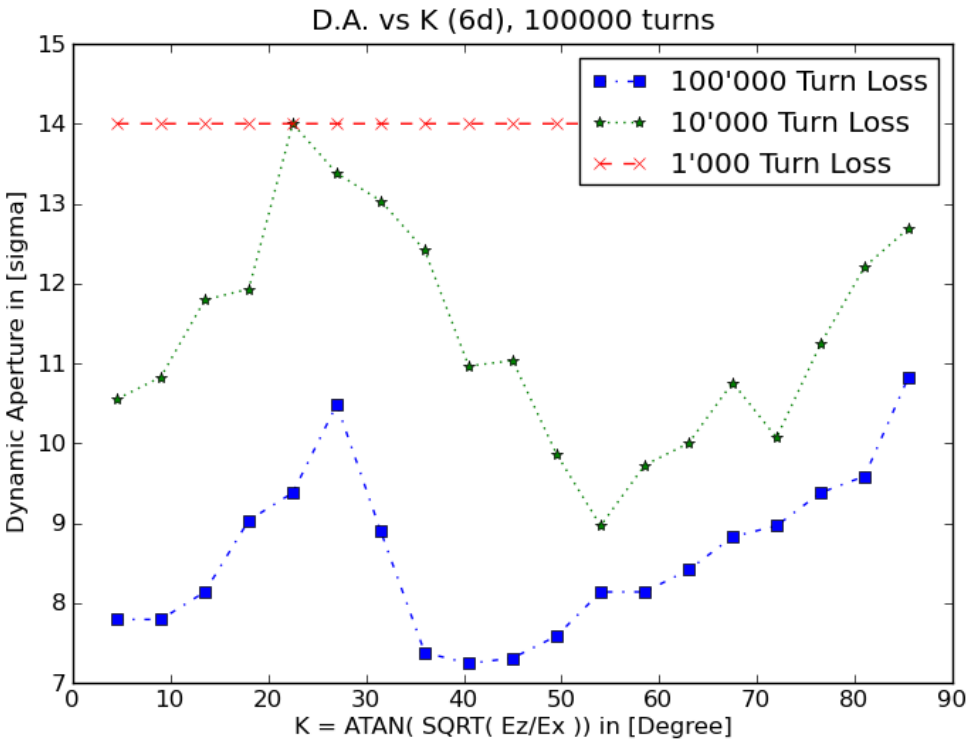


(b) SixDeskDB

Figure 4.10: **Distance in the phase-space of two initially close-by particles.** End value of the distance in phase space d(turns) of 2 initially closeby particles as a function of initial amplitude. `seed=1`, `angle=7`.

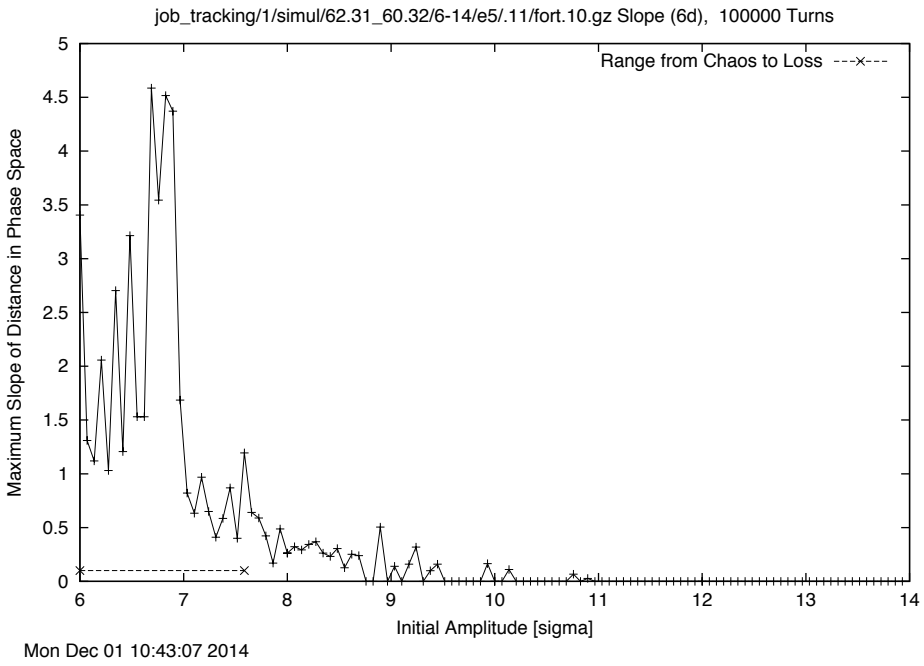


(a) SixDesk

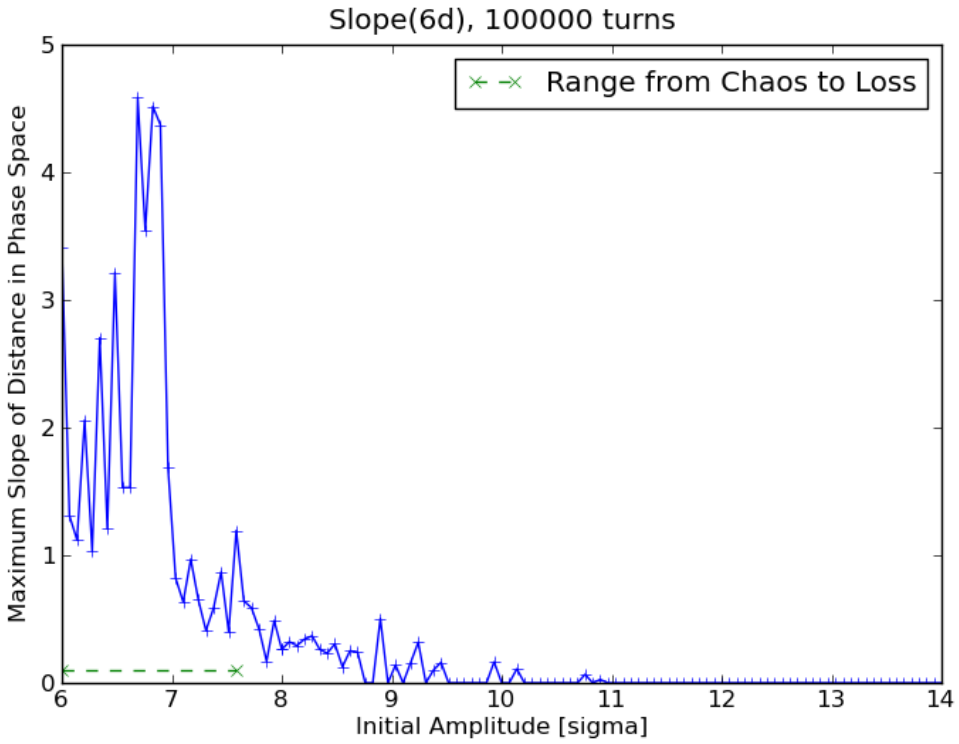


(b) SixDeskDB

Figure 4.11: **Dynamic aperture vs. K.** `seed=1`, `angle=5`.

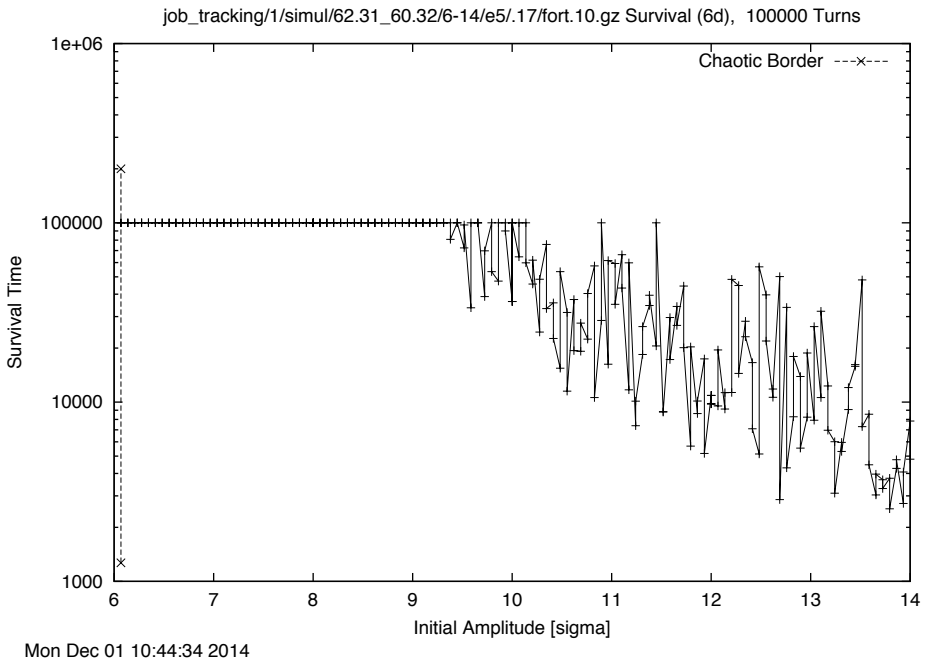


(a) SixDesk

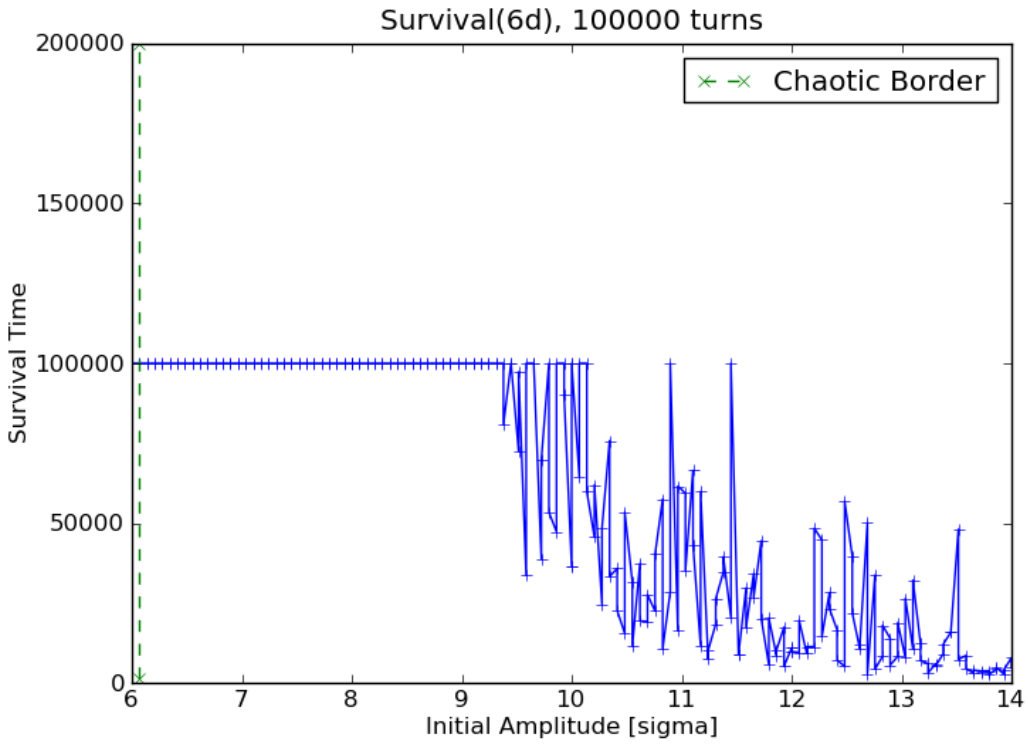


(b) SixDeskDB

Figure 4.12: **Maximum slope of distance in the phase-space.** Fitted slope of log d(turns) versus log (turns) of the distance in phase space of 2 initially close-by particles as a function of initial amplitude. `seed=1`, `angle= 11`.



(a) SixDesk



(b) SixDeskDB

Figure 4.13: **Survival Time.** Survival plot, i.e. survival time versus initial amplitude. `seed=1`, `angle=1`.

Chapter 5

Discussion and Conclusions

This project results in a step forward for the development of SixTrack: simplifying and speeding it up and, at the same time, providing new, improved and assembled technical documentation; developing a new, clearer, faster implementation for SixDesk's post-processing; and even building new interfaces and tools to analyze and extract the results.

The present Chapter discusses the impact of this whole project on SixTrack and SixDesk's post-processing flow, and states how an improved documentation helps SixTrack's development. Section 5.2 exposes and discusses the benefits of the new SixDeskDB post-processing routines implemented and, finally, Section 5.3 outlines the future stream of work and possible improvements to the code developed.

5.1 On Post-Processing and documentation

Post-processing is the key feature of SixDesk and one of the core parts of SixTrack. We've documented it so it's easy to trace back in a precise fashion what SixTrack is doing in each one of the steps the post-processing takes. This will improve the development speed, as the firsts collaborators of SixTrack were the only ones who completely understood how it worked, and they stopped working in the project 10 years ago.

Now, a thorough documentation for SixTrack's post-processing routines has been

made available, describing each step in the analysis and providing the theory behind them. This will speed up the development and will help understanding the whole process done after tracking: Sixtrack and SixDesk's post-processing, becoming a powerful tool for new scientists to use for the development of new analysis and studies.

Other than the code documentation, what was needed the most was an adaptation of the 4-Dimensional theoretical foundations for the 6-D phase space SixTrack works in. This adaptation has been obtained by thoroughly inspecting the Fortran code and its program flow, double-checking the results with the derivation of the original 4-D formulas and the theory foundations for better confidence. The fact that this is finally documented opens a whole new range of possibilities for the evolution of SixTrack, and its inclusion in the physics manual published with Sixtrack will solve and clarify doubts that had been there for years.

5.2 A New SixDeskDB Post-Processing Implementation

SixTrack's post-processing is the result of years of research and programming. SixDesk runs made the analysis of the results easier to develop and program. With SixDeskDB, that improvement has been taken several steps forward, using a modern, scientifically relevant, widely-used and clear language as Python.

As Python is one of the most, if not the most, used programming languages at CERN, this implementation opens the development of new SixDesk's analysis to a wider audience, capable of making the most of it. The post-processing analysis implemented in SixDeskDB is the core part of SixDesk, and porting it takes SixDeskDB a step closer to being able to fully substitute the original SixDesk.

This implementation proves to be faster and easier to develop, efficient using disk space and faster in execution time. On top of that, it adds new invaluable features like obtaining the plots and the Fort files on request through a friendly interface, hiding the SQL logic behind the queries and saving huge amounts of disk space by not auto-generating everything even if it is not needed.

5.3 Future Work and Possible Improvements

Further work in SixDeskDB involves implementing the to post-process short runs and integrate them in the program flow, refactoring the `readplot` function to share most of its code with the new `read10b` function to implement in SixDeskDB's Post-Processing module.

It would be interesting to integrate the functionality provided by the new `Fort` class in the rest of SixDeskDB's code and to provide a documentation for SixDeskDB and its variables specifically and not depend on SixDesk's one.

More future work could include implementing new physics models, integrating SixDeskDB in the LHC@Home platform, integrating all the homegrown alternative versions of SixTrack or improving the precision and use of floating point arithmetic.

Bibliography

[1] David P Anderson. Boinc: A system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 4–10. IEEE, 2004.

[2] M Berz. The differential algebra fortran precompiler dafor. *Los Alamos National Laboratory Technical Report AT-3: TN-87-32*, 1987.

[3] Catherine Daramy, David Defour, Florent de Dinechin, and Jean-Michel Muller. Cr-libm: a correctly rounded elementary function library. In *Optical Science and Technology, SPIE's 48th Annual Meeting*, pages 458–464. International Society for Optics and Photonics, 2003.

[4] R De Maria and M Fjellstrom. Sixtrack physics manual (draft). *cern. ch/sixtrack-ng/doc/physics_ manual/sixphys. pdf*, 2013.

[5] R De Maria, V Previtali, Y Levinsen, L Lari, F Schmidt, H Renshall, J Barranco, A Mereghetti, R Appleby, V Vlachoudis, et al. Recent developments and future plans for sixtrack. Technical report, 2013.

[6] Etienne Forest, John Irwin, and M Berz. Normal form methods for complicated periodic systems. *Part. Accel.*, 24:91–107, 1989.

[7] Massimo Giovanozzi, Igor Zacharov, and Leonid Rivkin. Lhc@ home: A volunteer computing system for massive numerical simulations of beam dynamics and high energy physics events. In *IPAC2012 Proceedings*, number EPFL-CONF-181616, 2012.

[8] John H Howard et al. *An overview of the andrew file system.* Carnegie Mellon University, Information Technology Center, 1988.

[9] John David Jackson. *Classical electrodynamics*, volume 3. Wiley New York etc., 1962.

[10] H Mais, G Ripken, A Wrulich, and F Schmidt. Particle tracking. 1986.

[11] E McIntosh and R De Maria. The sixdesk run environment for sixtrack. *CERN-ATS-Note-2012-089 TECH*, 2012.

[12] Eric Mcintosh, F Schmidt, F de Dinechin, et al. Massive tracking on heterogeneous platforms. In *2006 ICAP Conference in Chamonix, France*, 2006.

[13] Eric McIntosh and Andreas Wagner. Cern modular physics screensaver or using spare cpu cycles of cerns desktop pcs. In *Computing in High Energy and Nuclear Physics*, page 1055, 2004.

[14] Donald H Menzel. *Fundamental formulas of physics*, volume 2. Courier Corporation, 1960.

[15] Giuseppe Lo Presti, Olof Barring, Alasdair Earl, Rosa Maria Garcia Rioja, Sebastien Ponce, Giulia Taurelli, Dennis Waldron, and Miguel Coelho Dos Santos. Castor: A distributed storage resource facility for high performance data processing at cern. In *MSST*, volume 7, pages 275–280. Citeseer, 2007.

[16] Gerhard Ripken and Ferdinand Willeke. Methods of beam optics, 1988.

[17] F Schmidt. Sixtrack. Technical report, CM-P00049314, 1990.

[18] Frank Schmidt. Sixtrack, users reference manual. Technical report, CERN SL/94-56 (AP), 1994.

[19] Albin Wrulich. Racetrack-a computer code for the simulation of nonlinear particle motion in accelerators. 1984.