ETH ZÜRICH
PSI

Master Thesis

Measurement of beam energy spread from kicked beam decoherence at the SLS storage ring

---

Author:
Davit Mayilyan

Supervisor:
Dr. Andreas Streun
Dr. Andreas Adelmann

August 29, 2014

Abstract

The energy spread of the beam is important for understanding the collective effects like turbulent bunch lengthening and intra-beam scattering. Beam positions from all BPMs was obtained and merged using theoretical values of beta functions and phases. Local sine fits in a sliding window over the BPM data were used to interpolate the amplitude of the beam oscillation. A fit of the theoretical decoherence formula was performed to the amplitude data.

Vertical and horizontal beam excitation were studied with different pinger currents and for different chromaticity settings. The bunch current was varied to study the dependence of the energy spread on the bunch current and prove theoretical models and previous measurements on turbulent bunch lengthening.

Contents

1 Introduction

Beam physics studies require measurement of bunch parameters in all 6 dimensions: transverse emittance can be measured by optical beam size determination and by different methods of beam optics measurements. The bunch length can be measured using a streak camera. The energy spread of the beam is important for understanding the collective effects like turbulent bunch lengthening and intra-beam scattering, leading to current dependent blow-up of the electron bunch. There are several methods of measuring the energy spread: beam size measurements in dispersive and non-dispersive regions, widening of undulator spectra [1], and response of the beam to transverse excitation. The last one can be resonant [2] or kick excitation [3], which is the method used in this work.

The betatron oscillations of the particles in a transversely excited beam due to non-zero chromaticity will to go out of phase, resulting in decrease of the beam average position. This effect is called decoherence and will lead to the decrease of the apparent beam position measured by the beam position monitors. Along with this decoherence there is also a recoherence effect caused by the synchrotron oscillations over one period.

The time-variation of the beam position monitor signal due to decoherence and recoherence has been calculated in [11], analysing its dependence on energy spread, chromaticity, synchrotron tune and other parameters. In this work the energy spread of the beam is found using the information about chromaticity and synchrotron tune. To get the chromaticity value, tune values were measured using the correlation between the tune the and RF-cavity frequency through momentum compaction factor. To estimate the synchrotron tune, the Fourier spectrum of the beam position was examined. The amplitude of betatron oscillations was obtained by combining the data from all 73 BPMs of the SLS storage ring and fitting them with local sine function of betatron oscillations. The resulting behaviour of the amplitude was fitted with an envelope function taking the first order chromaticity as a given value, and second order chromaticity and synchrotron tune as variables with initial guesses discussed above. The resulting energy spread values were plotted against the beam current.

2 Beam dynamics

2.1 Summary of stored beam dynamics

2.1.1 Charged particle in the ring

To understand the influence of the magnets on the motion of the particles let's consider a particle in the magnetic field, moving perpendicular to the field lines of the magnet. The coordinate system is defined by z along the tangent to the design orbit, x transverse to the motion along the radial vector and y perpendicular to the plane of the circular trajectory (Frenet-Serret frame).

It is well known that such particle's motion will outline a circle of radius R and its motion will be described with the following formula

$$\frac{1}{R(x,y,z)} = \frac{e}{p}B_z(x,y,z). \tag{1}$$

Above B_z is the field perpendicular to the particle orbit, e is the charge of the particle and p is its momentum. Since transverse dimensions of the beam are negligible compared to the radius of curvature, we can expand both sides of the eq. (1) around the nominal trajectory:

$$\begin{aligned}\frac{e}{p}B_y(x) &= \frac{e}{p}B_{y0} + \frac{e}{p}\frac{dB_y}{dx}x + \frac{1}{2!}\frac{e}{p}\frac{d^2B_y}{dx^2}x^2 + \frac{1}{3!}\frac{e}{p}\frac{d^3B_y}{dx^3}x^3 + \cdots \\ &= \underbrace{\frac{1}{R}}_{\text{dipole}} + \underbrace{kx}_{\text{quadrupole}} + \underbrace{\frac{1}{2!}mx^2}_{\text{sextupole}} + \underbrace{\frac{1}{3!}ox^3}_{\text{octupole}} + \cdots\end{aligned} \tag{2}$$

Terms in the right hand side of the eq. (2) expansion correspond to different types of magnets: dipole, quadrupole, sextupole and octupole. Dipoles are used for the bending of the beam. Quadrupoles focus the beam. Sextupoles are used to compensate for the chromatic focusing errors of the quadrupoles. Chromaticity and chromaticity compensation will be discussed in detail in section 2.2. Detailed behaviour of charged particles in magnets can be analysed theoretically by solving the equation of the motion for the approximated Hamiltonian.

2.1.2 Periodic solution of the Equation Of Motion (EOM)

With discrete magnetic elements, the fields are piecewise constant and can be described by maps, relating the coordinates at entry to those at exit of the element.

In a storage ring, concatenation of the element maps results in the one-turn map. Stability requires the existence of a periodic solution of EOM: the closed orbit is a fix-point of the one-turn map. The periodic solution is given by the eigenvalues of the Jacobian of the map at the closed orbit (linearisation). The stable solution of the EOM is defined by the condition that eigenvalues are imaginary, which are given by the expression $\exp(\pm i 2\pi\nu)$, where ν is called *tune*, the number of oscillations of a particle in one turn.

The linear maps of dipoles and quadrupoles are matrices, so concatenation of these elements can simply be done by matrix multiplication. But this is not applicable to non-linear elements like sextupoles. However, once the orbit (fixpoint of non-linear one-turn map) has been found, the field of non-linear elements may be linearised locally in order to establish the Jacobian for calculation of the tunes. This is relevant for chromaticity correction with sextupoles, see figure 4.

To get the EOM we assume that $x \ll R$, $y \ll R$ and that the energy deviation is very small $\Delta p/p \ll 1$. Assuming high energies ($\gamma \gg 1$) hereafter we define $\delta \equiv \frac{\Delta p}{p} = \frac{\Delta E}{E}$. The EOM takes the form:

$$\begin{aligned} x''(z) + \left(\frac{1}{R^2} - k(z)\right) x(z) &= \frac{1}{R}\delta, \\ y''(z) + k(z)y(z) &= 0, \end{aligned} \tag{3}$$

where $k(z)$ is the strength of the quadrupole ($k < 0$ is defined as a horizontally focusing quadrupole, and $k > 0$ as a defocusing).

2.1.3 Dispersion and momentum compaction

In case of a pure dipole ($1/R \neq 0$ and $k = 0$) from eq. (3) for off-momentum particles ($\delta \neq 0$) we get:

$$x'' + \frac{1}{R^2}x = \frac{1}{R}\delta \tag{4}$$

For the case of $\delta = 1$ we define a special trajectory $D(z)$ and name it *dispersion function*. Rewriting eq. (4) we get:

$$D''(z) + \frac{1}{R^2}D(z) = \frac{1}{R}$$

Solving this we will get

$$\begin{pmatrix} D(z) \\ D'(z) \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\phi & R\sin\phi & R\,(1-\cos\phi) \\ -\frac{1}{R}\sin\phi & \cos\phi & \sin\phi \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} D_0 \\ D_0' \\ 1 \end{pmatrix} \tag{5}$$

where $\phi = \frac{z}{R}$ is the deflection angle and R is the dipole radius. Equation (5) represents the dispersion for the dipole. Particles with non-nominal energies follow dispersive trajectories and are described by $x_D = D(z)\delta$. Radial position of a particle in a dipole will be: $R = R_0 + x_D$. The length of the path travelled by an off-momentum particle will be $L = L_0 + \int R d\phi$, where integration is over all dipoles in the ring and L_0 is the length of straight lattice sections (quads, drifts etc.). So, dependence of pathlength L on momentum deviation δ can be found by differentiation and using $d\phi = dz/R$:

$$\frac{dL}{d\delta} = \int \frac{D(z)}{R} ds.$$

We define the *momentum compaction factor* α as

$$\alpha = \frac{\Delta L/L}{\delta}. \tag{6}$$

2.1.4 Beta function and betatron oscillations

As discussed above, the motion of a particle in the accelerator can be described by multiplication of individual element's transformation matrices. Dispersion gave us the off-momentum orbit behaviour. Now we investigate the motion of an ensemble of particles. We consider the on-momentum case ($\delta = 0$) of eq. (3) and include the dipole focusing terms $K_x(z) = \frac{1}{R^2} - k$ and $K_y(z) = +k$ for horizontal and vertical focusing respectively, with

$$u'' + K_u(z)u = 0, \tag{7}$$

where u is the horizontal or vertical axis (x or y) and $K_u(z)$ is the distribution of focusing along the beam line. We take the ansatz

$$u(z) = \mathcal{A}\sqrt{\beta(z)}\cos[\psi(z) - \psi_0], \tag{8}$$

where $\mathcal{A} = \sqrt{2J}$ is an invariant amplitude and J is the the *Courant-Snyder invariant*. Solving eq. (7) gives

$$\gamma u^2 + 2\alpha u u' + \beta u'^2 = \mathcal{A}^2, \tag{9}$$

where $\beta, \alpha = -\frac{1}{2}\beta'$ and $\gamma = \frac{1+\alpha^2}{\beta}$ are the Twiss parameters [4]. Remembering that for uniformly distributed betatron phases the average of $\sin^2(x)$ or $\cos^2(x)$ is $1/2$, we define the *beam emittance* ε. It is a statistical quantity calculated from eq. (8):

$$\varepsilon = \sqrt{\langle x^2 \rangle \langle x'^2 \rangle - \langle x x' \rangle^2} = \langle J \rangle \tag{10}$$

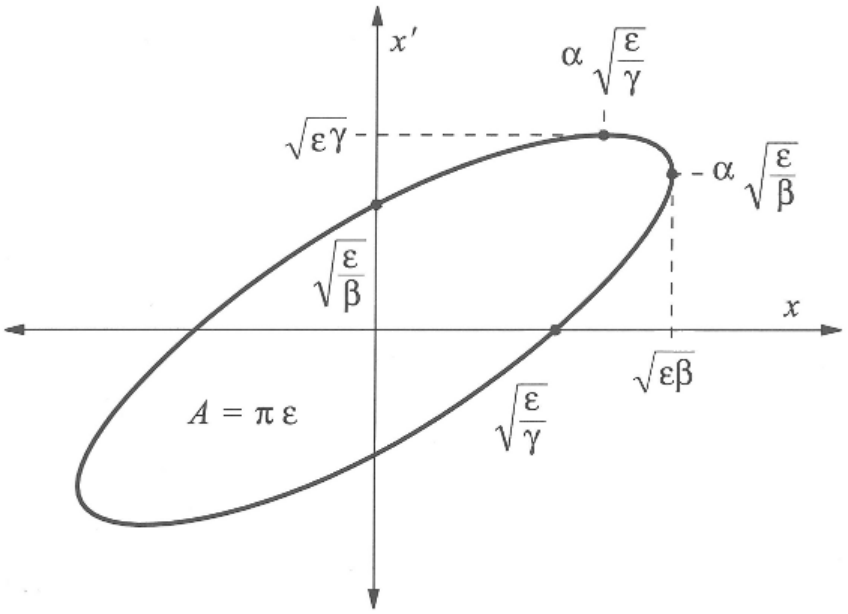

Figure 1: Ellipse of particle motion in the phase space defined by the Twiss parameters. Picture taken from [6].

We choose $\mathcal{A} = \sqrt{\varepsilon}$ as a characteristic amplitude defining the whole beam. Then, the eq. (9) describes an ellipse of area $\pi\varepsilon$ in phase space. Moreover, it represents a motion of a particle along the contour of the ellipse defined by Twiss parameters (fig. 1). Along the design orbit the shape of the ellipse will change due to the transformation of Twiss parameters along the ring. Furthermore, due to the Liouville's theorem, a particle starting on an ellipse will stay on it during its motion. Thus, the ellipse corresponding to a particle with some betatron amplitude will confine the motion of all particles with smaller betatron amplitudes. In other words, particles with smaller betatron amplitudes will oscillate within this ellipse.

Ignoring the phase term in eq. (8) we get the beam envelope

$$E(z) = \pm\mathcal{A}\sqrt{\beta(s)}, \tag{11}$$

where the sign $\pm$ indicates the presence of an envelope on two sides of the beam (fig. 2).

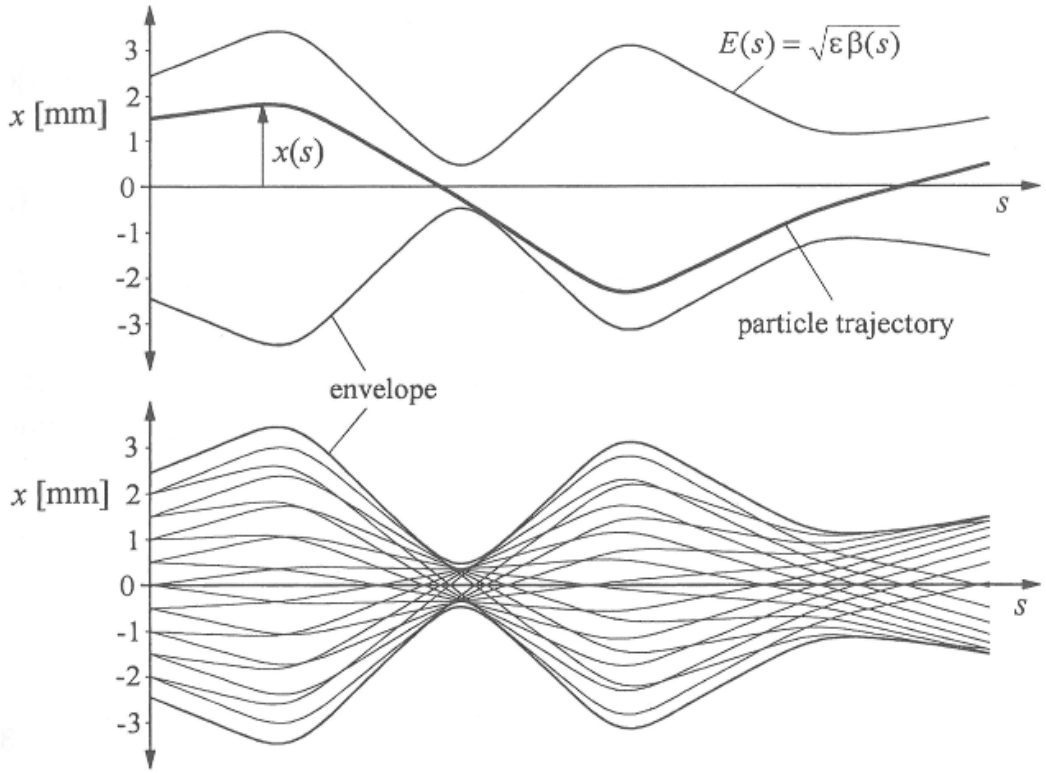

Figure 2: Top: the evaluation of the particle in the envelope. Bottom: trajectories of many particles in the envelope. Picture taken from [6].

2.1.5 Phase focusing and synchrotron oscillation

Radio frequency (RF) cavities are used in a synchrotron to accelerate particles and in a light source storage ring to compensate for the energy loss due to synchrotron radiation. The effect of the cavity on the beam is explained below.

For simplicity we assume that particles have some fixed velocity, for example velocity of light ($v = c$). An ideal on-momentum particle with $\delta = 0$ travels along the nominal trajectory fixed by lattice design of the ring and has the nominal phase Ψ_0 relative to the RF voltage when it passes through RF-cavity. Here the energy required by the particle to compensate for losses from synchrotron radiation and for acceleration, is supplied exactly. A particle whose momentum is too high ($\delta > 0$) will travel along an outer dispersive trajectory, travelling a distance longer

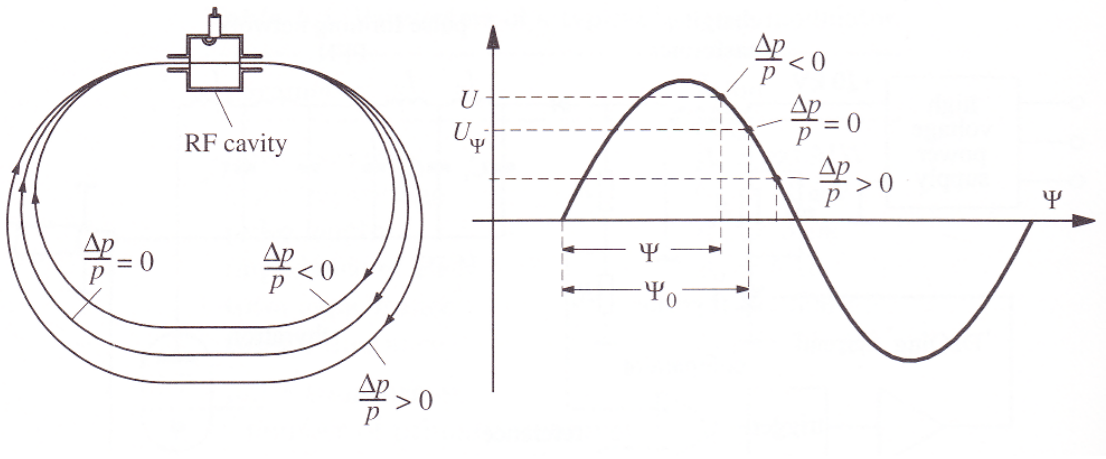

Figure 3: Phase focusing of particles in a circular accelerator. Figure is taken from [6].

than the nominal trajectory. Therefore, it will arrive at the cavity later, with a bigger phase $\Psi = \Psi_0 + \Delta\Psi$, and so it will see a correspondingly smaller voltage (see fig. 3). As a result it is accelerated less than an ideal particle and it will come closer to the nominal phase in the next turn. The opposite is happening to the particle with smaller energy. It gains more energy than the nominal and catches up with the nominal phase. So, particles oscillate around the ideal phase and so are kept stable within the accelerator. This longitudinal periodic particle motion around the nominal phase is called *synchrotron oscillation* [6].

2.2 Chromaticity

One of the parameter requirements for a light source like SLS is the photon beam brightness, which is preferable to be as high as possible. In order to get higher brightness, strong quadrupole magnets are used to focus the beam.

These strong quadrupole magnets give us lower emittance which results in higher brightness of outgoing photons. The spread of beam particle energies will result in different focusings by the quadrupole. Particles with higher energy are focused less than particles with lower energy. This phenomenon is called *chromaticity* and is illustrated in fig. 4. In order to correct this aberrations in the focusing one introduces sextupole magnets in the storage ring at points where the

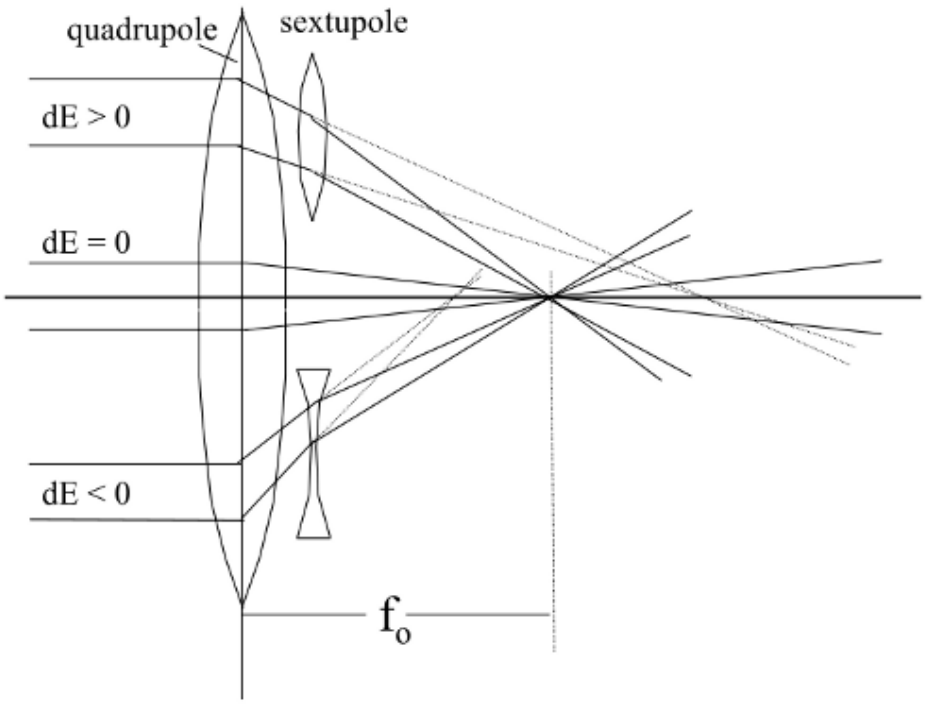

Figure 4: Picture shows the unequal focusing of quadrupoles depending on the energy. The sextupole correction of the chromaticity is also shwon. For high energies it has a focusing effect and for low energies it has defocusing effect. Figure is taken from [8].

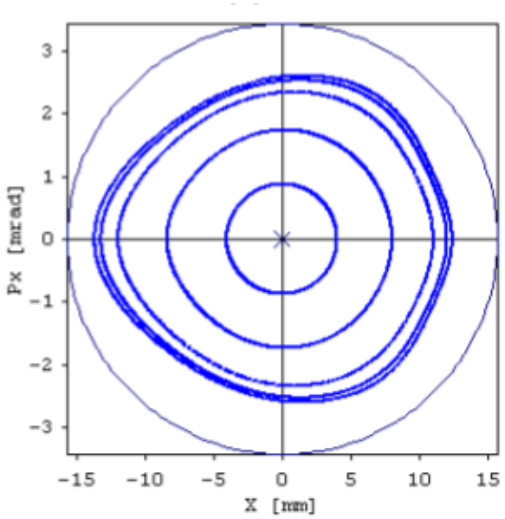

Figure 5: Non-linear elements like sextupoles perturb the elliptical shape and thus may introduce instabilities.

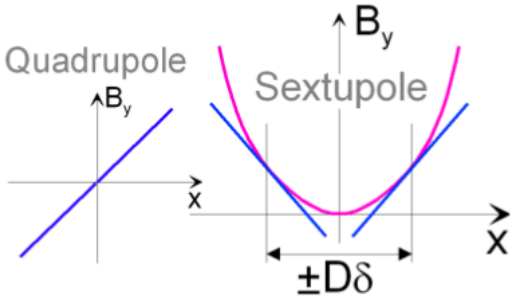

Figure 6: A deviation from the sextupole center provides a focusing or defocusing effect.

dispersion is not zero. So, sextupoles can be used both as a focusing element for higher energy particles and as a defocusing for lower energy particles (see fig. 4). Beside, eliminating the quadrupole induced chromaticity, sextupoles introduce other effects, particularly non-linear perturbations (figure 5). The dependence of quadrupole and sextupole position is shown in fig. 6. As it can be seen the field has a linear dependence on transverse position whereas the sextupole has a quadratic dependence.

2.3 Radiation equilibrium

Energy loss from the beam is a result of emission of γ-particles from the beam and is called synchrotron radiation. As we know from the theory it is a discrete process and particles are emitted in quanta of energies. Emission of a discrete energy portion gives a recoil on the trajectory of an electron. Emission of a quantum is very small and is less than $\rho/\gamma c$, where ρ is the radius of curvature of the trajectory and γ is the electron energy in units of its rest energy. Since the emission time is much smaller than the period of betatron or synchrotron oscillation, we can consider the emission as an instantaneous kick. Energy of the electron is much larger than the energy of the emitted photon, the emission of successive photons can be considered as purely random, statistically independent processes.

The emission of a photon disturbs the trajectory of the electron and excites oscillations. Cumulative disturbance of trajectories of many particles in the beam result in amplitude growth. The opposite effect is the radiation which is a dissipative process, leading to damping. At some point these two contradicting effects will be balanced, i.e. the excitation will be balanced by the oscillation damping [7].

2.3.1 Radiation damping

Photon emission causes loss of a particle's momentum both in longitudinal and in transverse planes. This process is called *damping* and affects the motion of the beam. Damping is a quasi-continuous process, where the particles lose momentum by radiation successively. Afterwards, the momentum of the particles is restored in the RF-cavity but since the electric field in the RF-cavity is oriented along the longitudinal direction, only the longitudinal component of the particle's momentum is restored. The interplay of these two effects leads to the process of radiation

damping, where the longitudinal momentum asymptotically approaches the design value while the transverse momentum approaches zero. Damping changes the synchrotron oscillation amplitude by reducing it by $A = A_0 e^{-\alpha_z t}$, where α_z is the damping coefficient along the trajectory [8]. The synchrotron oscillation damping time is defined as $\tau_z = 1/\alpha_z$. The rate of change of the amplitude will be

$$\left\langle \left. \frac{dA^2}{dt} \right|_q \right\rangle_z = -\frac{2}{\tau_z} \left\langle A^2 \right\rangle .$$

2.3.2 Quantum Excitation

Due to the recoil from photon emission, the particle momenta are randomly changed. This is a statistical process leading to an energy spread within the beam. To evaluate the effect of quantized emission of photons on the particles' energy spread we discuss particles performing synchrotron oscillations in a way that an energy deviation of A_0 at time t_0 will lead to an energy error of

$$A(t) = A_0 e^{i\Omega(t-t_0)}$$

at time t.

Due to transfer of momentum from the particle to a photon, emission of a photon will lead to a change of a particle's amplitude. The emission of a photon of energy ϵ at time t_1 will cause a change of synchrotron oscillation amplitude

$$A_1 = A_0 e^{i\Omega(t-t_0)} - \epsilon e^{i\Omega(t-t_1)}.$$

Because the times at which photon emission occurs is random we have for the average increase in oscillation amplitude due to the emission of a photon of energy ϵ

$$\left\langle \Delta\ A^2 \right\rangle = \left\langle A_1^2 - A_0^2 \right\rangle = \epsilon^2.$$

The rate of amplitude change due to the quantum excitations is obtained by averaging over the ring:

$$\left\langle \left. \frac{dA^2}{dt} \right|_q \right\rangle_z = \int_0^\infty \epsilon^2 \dot{n}(\epsilon) d\epsilon = \left\langle \dot{\mathcal{N}}_{ph} \left\langle \epsilon^2 \right\rangle \right\rangle_z , \tag{12}$$

where $\dot{\mathcal{N}}_{ph}$ is the total photon flux and $\dot{n}(\epsilon)$ is the number of photons of energy ϵ emitted per unit time and energy bin $d\epsilon$. Due to the dispersion diffusion of the longitudinal momentum translates to diffusion of the horizontal momentum. Therefore, we get excitation in the horizontal plane, but not in the vertical plane.

2.3.3 Equilibrium of damping and quantum excitation

Section 2.3.1 described a process which leads to loss of energy and decrease of the amplitude, while the section 2.3.2 described an effect resulting in an increase of amplitude. Therefore, in order to have a stable beam these two effects should compensate each other. Equilibrium state of excitation and damping will be

$$\left\langle \dot{\mathcal{N}}_{ph} \left\langle \epsilon^2 \right\rangle \right\rangle_z - \frac{2}{\tau_z} \left\langle A^2 \right\rangle = 0. \tag{13}$$

By evaluating eq. (13), the energy spread is derived [8]

$$\sigma_\delta = \frac{55}{32\sqrt{3}} \frac{hc}{m_0 c^2} \gamma^2 \frac{\langle |h^3| \rangle}{\langle h^2 \rangle J_s}, \tag{14}$$

where $C_q = \frac{55}{32\sqrt{3}} \frac{hc}{m_0 c^2} = 3.84 \cdot 10^{-13}\,\mathrm{m}$, $h = 1/R$ is the orbit curvature in the bending magnets, $J_s = 2\alpha_z T_0 E_0 / U_0$ the longitudinal damping partition number (T_0 is time for one turn, E_0 beam energy, U_0 radiated energy per turn) and $\gamma = E_o/mc^2$. Using the parameters for SLS ($R = 5.7\,\mathrm{m}$, $J_s = 2$, $E = 2.4\,\mathrm{GeV}$) we get the energy spread of $\sigma_\delta = 8.6 \cdot 10^{-4}$.

Due to the central limit theorem of statistics, the particle distribution has Gaussian shape:

$$\rho(x, y) = \frac{Ne}{2\pi\sigma_x\sigma_y} \exp\left(-\frac{x^2}{2\sigma_x^2} - \frac{y^2}{2\sigma_y^2} \right)$$

with the standard root mean square of $\sigma_u^2 = \frac{1}{2} \langle A_u^2 \rangle$ with $u = x, y$ and $\sigma_z^2 = 0$.

2.4 Decoherence theory

The energies of the particles in the beam have a Gaussian distribution with the rms energy spread given in eq. (14). After transverse excitation of the beam with a fast kicker (pinger magnet), all particles will perform betatron oscillations. In case of non-zero chromaticity, the individual betatron tunes of the particles depend on their momentum, so they accumulate different betatron phase while propagating, and their betatron oscillations get out of phase - a process named decoherence. As a consequence, the value for the beam position, i.e. the mean value of all particles' coordinates, as measured by a beam position monitor (BPM), will decrease. However, all particles have the same mean energy if we average over one period of the synchrotron oscillations, thus they accumulate the same betatron phase independent of their individual momentum, and a recoherence is observed

after each integer multiple of the synchrotron oscillation period. Due to non-linear effects like higher order chromatiticy and amplitude dependent tune shifts, the decoherence is only partially reversible, and a slow decay of the beam oscillation is observed. This effect can be seen on one BPM's data shown in figure 7. Dispersion in the horizontal plane will not be an issue as the mean over the dispersion orbits equals to zero.

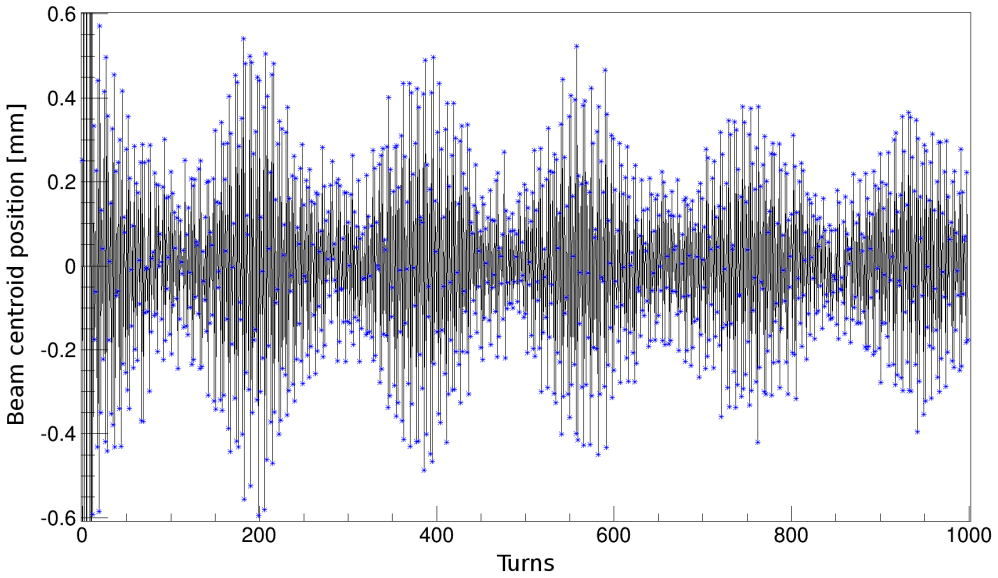

Figure 7: Signal from one BPM over 1000 turns. The coherence-decoherence pattern of the signal is clearly seen.

The particle position in the n^{th} turns is given by pseudo-harmonic oscillation formula

$$x(n) = r\cos\phi(n), \tag{15}$$

where $r = \sqrt{2J}\sqrt{\beta}$ with β the function at the BPM, J the Courant-Snyder invariant, and $\phi(n)$ the betatron phase at n^{th} turn. After N turns the phase advance is

$$\phi(N) - \phi(0) = 2\pi \int_0^N \nu(n)dn \tag{16}$$

where $\nu(n)$ is the betatron tune in n^{th} turn. $\phi(0)$ is the betatron phase at 0^{th} turn. For an ideal particle ($x = y = \delta = 0$), $\nu(n)$ is equal to the nominal betatron tune

ν_0. The non-zero values of the first and second-order chromaticities and amplitude dependent tune shifts result in different betatron tunes. The total tune shift with respect to the nominal tune will be expressed as

$$\Delta\nu_x(n) = \xi_{x1}\delta(n) + \xi_{x2}\delta^2(n) + \frac{\partial\nu_x}{\partial J_x}(J_x) + \frac{\partial\nu_x}{\partial J_y}(J_y), \tag{17}$$

where ξ_{x1} and ξ_{x2} are the first and second order chromaticities and the last two terms are tune shifts due to the Gaussian distribution of betatron amplitudes related to the beam emittances. Eq. (17) is for the horizontal tune, of course a corresponding equation exists for the vertical tune. Only the first term with ξ_1 is linear and reversible, the other terms are non-linear and always lead to decoherence only.

Momentum deviation and longitudinal τ coordinate change along the trajectory due to the synchrotron oscillation are expressed by

$$\begin{pmatrix}\delta(n)\\ \tau(n)\end{pmatrix} = \begin{pmatrix}\cos 2\pi\nu_s n & \sin 2\pi\nu_s n\\ -\sin 2\pi\nu_s n & \cos 2\pi\nu_s n\end{pmatrix}\begin{pmatrix}\delta_0\\ \tau_0\end{pmatrix}, \tag{18}$$

where $\tau = \Delta s/\sigma_s$ is the particle longitudinal coordinate normalized to the rms bunch length.

Motion of the beam is described by the formula

$$\begin{aligned}\langle x(N)\rangle = \sqrt{\beta_{BPM}}\sqrt{\beta_K}\Delta x' A(N)H(N)e^{-M(N)}\\ \sin(P(N) + Q(N) + 2\pi\nu_0 N),\end{aligned} \tag{19}$$

where β_{BPM} is the β function at the BPM, β_K is the β function at the pinger, $\Delta x'$ is the kick applied by the pinger.

Parameters $A(N), H(N), M(N)$ mentioned in the eq. (19) have the following form:

$$A(N) = \frac{1}{\sqrt{1 + (4\pi\mu_{xy}N)^2}(1 + (4\pi\mu_x N)^2)}\exp\left(-\frac{w^2}{2}\frac{(4\pi\mu_x N)^2}{1 + (4\pi\mu_x N)^2)}\right) \tag{20a}$$

$$H(N) = [1 + 2K_2^2(T^2 + \sin^2 T) + K_2^4(T^2 - \sin^2 T)^2]^{-(1/4)} \tag{20b}$$

$$M(N) = \frac{K_1^2(1 - \cos T)}{1 + 2K_2^2(T + \sin T)^2} \tag{20c}$$

where,

$$K_1 = \frac{\sigma_\delta \xi_1}{\nu_s} \tag{20d}$$

$$K_2 = \frac{\sigma_\delta^2 \xi_2}{\nu_s} \tag{20e}$$

$$T = 2\pi\nu_s N \tag{20f}$$

$$\mu_x = \varepsilon_x \frac{\partial \nu_x}{\partial J_x} \qquad \mu_y = \varepsilon_y \frac{\partial \nu_y}{\partial J_y} \tag{20g}$$

$$\mu_{yx} = \varepsilon_x \frac{\partial \nu_y}{\partial J_x} \qquad \mu_{xy} = \varepsilon_y \frac{\partial \nu_x}{\partial J_y} \tag{20h}$$

$$w = \frac{\beta_x \Delta x'}{\sigma_x}, \tag{20i}$$

and where $\Delta x'$ is the pinger amplitude and w is the kick in units of σ_x.

Here ξ_1 and ξ_2 are the first and second order chromaticities. ν_s is the synchrotron tune and σ_δ is the energy spread. H(N) gives irreversible, non-linear decoherence due to the second order chromaticity. As we see the modulation deepness M contains information about the energy spread. Particularly K_1 and K_2 contain all crucial parameters giving the modulations. The second order chromaticity in M is responsible for the decrease of modulation depth in time. A defines decay of decoherence due to a dependence of betatron tune on the oscillation amplitude and thus a diffusion due to the finite beam emittances. P and Q are slow phase modulations which are not used further, because we will evaluate only the envelope curve of the beam oscillation.

2.5 Turbulent bunch lengthening

Turbulent bunch lengthening (TBL) is the increase of bunch length and energy spread beyond a threshold value of bunch current, I_b^{th}. TBL is related to the interaction of the field of particles with the pipe.

The energy spread behaviour due to TBL is:

$$x = \left(\frac{I_b}{I_b^{th}}\right)^{1/3} for\ I_b > I_b^{th}, \tag{21}$$

where x is the energy spread or bunch length normalized to the value at zero current, I_b is the current per bunch, I_b^{th} is the threshold current for the onset of TBL and related to the broadband impedance of the machine. TBL leads to an increase of energy spread above a threshold current. For the SLS a threshold

current of $\approx$ 1.1 mA for the bunch had been found [5], so the energy spread behaves approximately according to the following relation:

$$\frac{\sigma_\delta}{\sigma_{e0}} \sim (I_b[mA]/1.1)^{\frac{1}{3}} for\ I_b > I_b^{th} \tag{22}$$

2.6 Third harmonic cavity

A usual requirement for the synchrotron source is high brightness of photons. High brightness means high currents which lead to the loss of particles, i.e. to beam lifetime decrease. For improving the beam lifetime in high brightness operation mode the bunches are lengthened by means of a third harmonic cavity [12], resulting in an improvement of beam Touschek lifetime by a factor of about 3 at the SLS.

3 Data acquisition

3.1 Setup

Data acquisition from the BPMs is done with a control system package named EPICS [13]. The Experimental Physics and Industrial Controls System (EPICS) is an established framework for the development of distributed control systems in the field of particle accelerators, particularly in SLS [14].

Horizontal and vertical plane data are obtained from the BPMs in turn-by-turn mode. Turn-by-turn mode means that there is no averaging over turns, which means that beam position at a given turn isn't affected by the previous turn(s). They were read out in one call in EPICS. One-call readout is important to examine possible correlation effects between coordinates.

The data from the horizontal plane are written first and the data for the vertical plane afterwards (see table 1 for the data file structure). Overall there are 73 BPMs, thus 73 $\times$ 2 lines (+1 information line). The first information line contains general information: in which mode the data were taken and the corresponding file-name including the path. the file-name in this line and time in general is written in the following format: DDMMMhh:mm:ss.dat. The first column indicates the plane of the kick: 0 for horizontal and 1 for vertical kick. The second column is the identifier of the BPM, which will be discussed below. The next columns contain the data, i.e. beam centroid position for 1000 turns.

To identify data coming from different modules of accelerators a general naming convention was introduced [15]. First two columns of file 1 are explained in the sketch below.

$$\underbrace{0}_{\substack{0/1\\ \text{horizontal}\\ \text{or}\\ \text{vertical}}} \sqcup \underbrace{\text{ARIDI}}_{\substack{\text{A = Accelerator}\\ \text{RI = Ring}\\ \text{DI = Diagnostics}}} - \underbrace{\text{BPM}}_{\substack{\text{module}\\ \text{identifier}}} - \underbrace{\text{01LB}}_{\substack{\text{sector no.}\\ \text{BPM indentifier}}} : \underbrace{\text{WF-X-1}}_{\text{attribute}}$$

For each parameter setup measurements were taken 10 times from statistical considerations, because sometimes BPMs fail to measure the beam position. All these files were cleaned up and prepared for further analysis using simple Mathematica code.

#	File content					
1	BPM Data (73 Waveforms) in Turn-Turn Mode,File: /path/30Apr11:24:45.dat					
2	0	ARIDI-BPM-01LB:WF-X-1	7.6025	0.2943	-0.887456	⋯
3	0	ARIDI-BPM-01LE:WF-X-1	11.0775	0.0368924	0.130838	⋯
4	0	ARIDI-BPM-01LD:WF-X-1	13.6225	0.203052	-0.546434	⋯
⋮						
75	1	ARIDI-BPM-01LB:WF-Y-1	7.6025	-0.0843718	-0.117673	⋯
76	1	ARIDI-BPM-01LE:WF-Y-1	11.0775	-0.160392	0.574342	⋯
77	⋮					⋮
⋮						
147	⋯	⋯	⋯	⋯	⋯	

Table 1: An example of data file structure. Horizontal data set comes first and vertical set afterwards.

As it was mentioned in sec. 2.4 the modulation depth is expressed by the function $M(N)$, particularly in eqns. (20d) and (20e). These two terms contain the energy spread with other parameters. Particularly in eqns. (20d) and (20e) we take the synchrotron tune and 2^{nd} order chromaticity as varying, whereas the 1^{st} order chromaticity is measured with sufficient precision to be taken as fixed. The only remaining untouched parameter is the energy spread, which is taken as a free parameter in the fit.

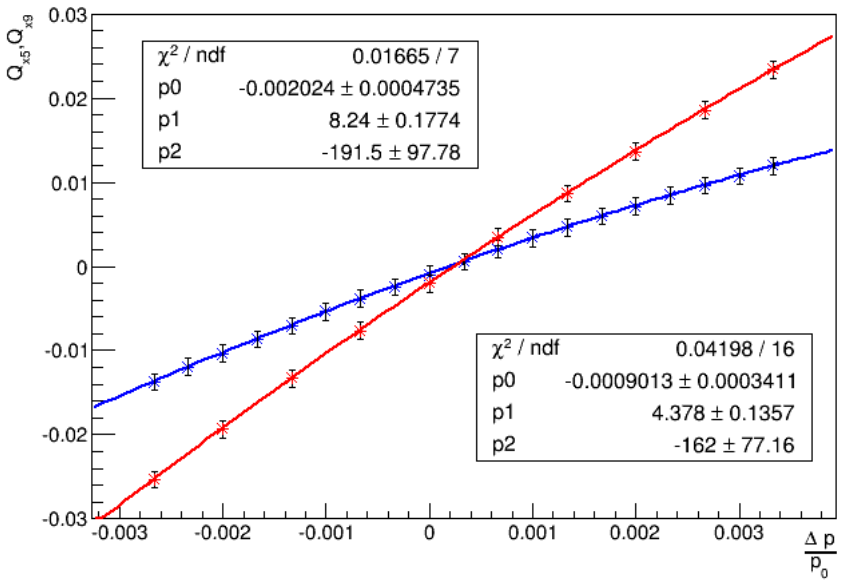

Figure 8: Horizontal frequency dependence. Red line corresponds to chromaticity of 9 and blue line to chromaticity of 5.

3.2 Chromaticity data analysis

The first set of parameters that should be given to the fit procedure is the first and second order chromaticities. While the first one can be determined with small error, the later one varies significantly and should be taken as a varying parameter.

In order to get the chromaticities first we would like to find the relationship between the change of frequency in the RF-cavity and the momentum of particles. Rewriting the momentum compaction factor (eq. (6)) we get:

$$\frac{\Delta L}{L} = \alpha\delta, \tag{23}$$

where L is the path length and δ is the relative momentum deviation. On the other hand the path length is equal to the RF wavelength multiplied by an integer number $L = h\lambda$, where h is called harmonic number. Assuming that energies are high enough to take the speed of particles equal to the speed of light, we can rewrite L as

$$L = h\frac{c}{f}, \tag{24}$$

where f is the frequency of the RF-cavity and h is the harmonic number. Path length change depends on the frequency change of the RF-cavity:

$$\Delta L = -\frac{hc}{f}\frac{\Delta f}{f} = -\frac{hc}{f^2}\Delta f. \tag{25}$$

The minus sign comes from the differentiation of eq. (24). Physically it corresponds to the fact that higher momentum results in longer path length and vice versa. Using equations (24) and (25) we get

$$\frac{\Delta L}{L} = -\frac{\Delta f}{f} \tag{26}$$

Merging eq. (26) with (23) results in

$$\delta = -\frac{1}{\alpha}\frac{\Delta f}{f} \tag{27}$$

This is the needed dependence of energy of particles on frequency change of the *RF-cavity*.

For further data analysis we need to remember first and second order chromaticity definitions:

$$\xi_1 \equiv \frac{d\nu}{d\delta} \qquad \xi_2 \equiv \frac{1}{2}\frac{d^2\nu}{d\delta^2}, \tag{28}$$

Where ν is the tune defined as:

$$\nu = \frac{\Delta\phi}{2\pi}. \tag{29}$$

with $\Delta\phi$ the phase advance over one turn. Connection between tune and chromaticity is given by:

$$\nu = \nu_0 + \xi_1\delta + \xi_2\delta^2. \tag{30}$$

From the plot of tune versus frequency change $(f_{nominal} - f_{measured})/f_{nominal}$ one can get the experimental chromaticity value. Measurements were done by starting from some nominal *RF-frequency* and varying the frequency around the nominal by `-800 Hz` and `+1 kHz`. Nominal values for horizontal and vertical tunes were $\nu_x = 20.4385$ and $\nu_y = 8.74538$. Chromaticity for both horizontal and vertical components were set to values of 5 and 9. The filling pattern was 150 mA in 150 bunches. The currents for the pingers were 60 A and 100 A respectively. The plot with polynomial fit for horizontal axis is presented in the figure 8. Polynomial fit of experimental data gives $\xi_1 = 4.38 \pm 0.14,\ \xi_2 = -162 \pm 77$ for a set of chromaticity of 5. Results for vertical kick are given in figure 9. For a set chromaticity of 9, the

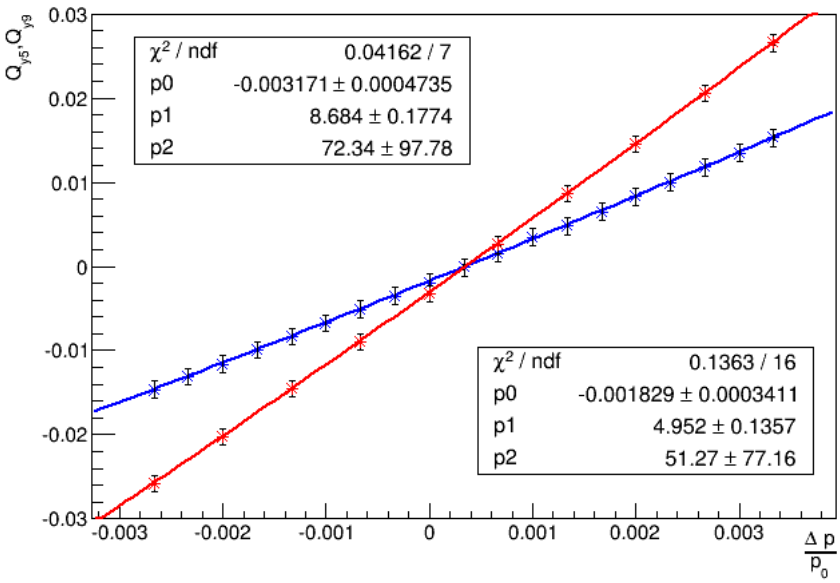

Figure 9: Vertical frequency dependence. Red line corresponds to chromaticity of 9 and blue line to chromaticity of 5.

results are $\xi_1 = 8.24 \pm 0.18,\ \xi_2 = -191 \pm 97$. First and second order chromaticity values for horizontal and vertical kicks are presented in the table 2.

As it is seen from the table, the tune was measured for only 2 chromaticity values, but BPM position data was taken also for other chromaticities. Hence, other chromaticity values were obtained from linear interpolation of theoretical versus experimental values of chromaticity. This scheme is valid only if we assume that the tune measurement quality is not affected by the chromaticity value. This scheme was used not only for obtaining the first order chromaticty but also for estimating its error value and for second order chromaticity.

3.3 Synchrotron tune

Synchrotron oscillations and the resulting synchrotron tune are responsible for phase focusing, explained in section 2.1.5. Synchrotron tune is the next parameter which will be needed for obtaining the energy spread. It can be analysed from the FFT spectrum of the turn by turn data from any BPM.

Modulation due to decoherence and recoherence with synchrotron tune in the

	ξ_5	ξ_9
x	4.38 ± 0.14	8.24 ± 0.18
y	4.95 ± 0.14	8.68 ± 0.18

(a) First order chromaticity values from the fit.

	ξ_5	ξ_9
x	-162 ± 77	-191 ± 98
y	51 ± 77	72 ± 0.18

(b) Second order chromaticity values from the fit.

Table 2: Chromaticity values obtained from the polynomial fit of tune versus RF-frequency.

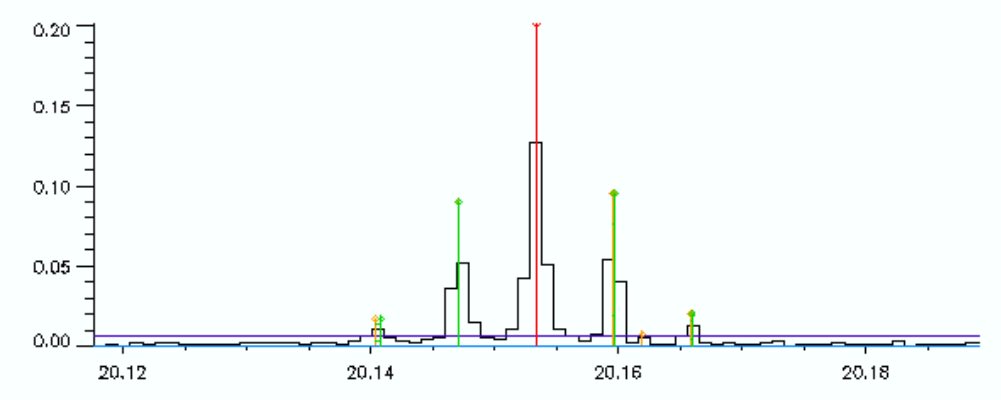

Figure 10: Betatron peak with side-bands, corresponding to synchrotron oscillation. Figure was taken from control room online beam analysis tool-kit.

Fourier spectrum results in side-bands (figure 10). The distance of these side-bands to the main peak gives the synchrotron tune, which can be obtained from an individual BPM data. In the Fourier spectrum of these data the peak with the highest intensity corresponds to the betatron oscillation. Synchrotron tune is used as a varying parameter in the fit procedure, and we need just an estimate for the initial guess of the tune. The fit procedure turned out to be robust to variation of the initial guess for the synchrotron tune.

4 Sine interpolation and evaluation of beam position envelope

As it was mentioned above SLS has 73 BPMs, which are located almost symmetrically in the ring (fig. 11). BPMs are distributed around the ring to cover each period of betatron oscillation horizontally and vertically by more than two BPMs in order to control the beam orbit. So, each BPM is located at a different betatron phase.

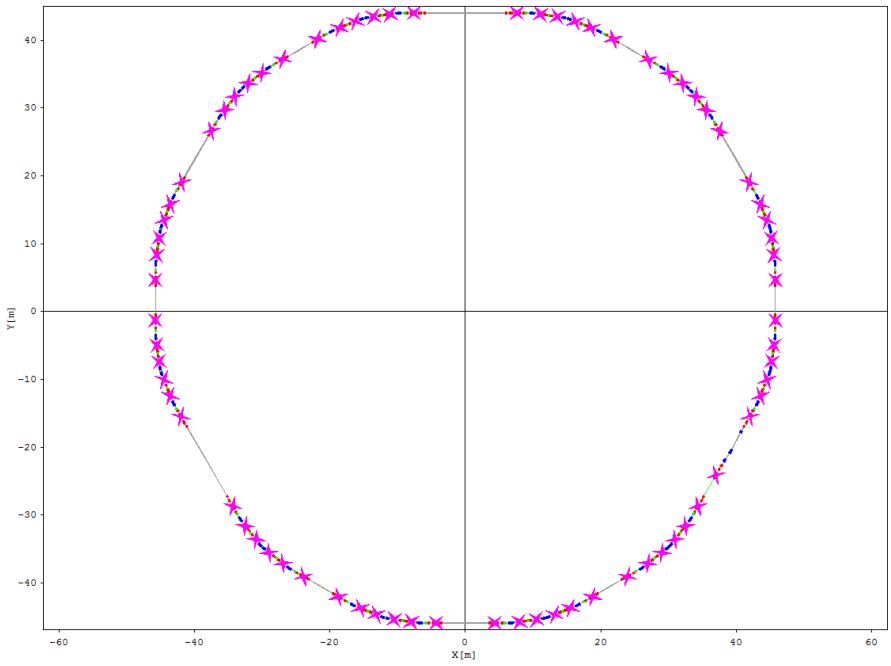

Figure 11: Positions of BPMs in the SLS storage ring

The turn by turn signal from the BPMs is composed from two kinds of oscillations, one coming from the fast betatron oscillation and the slow synchrotron oscillation. A global fit of synchrotron and betatron oscillations according to Eq. (19) turned out to be rather fragile. It worked only for very low pinger currents and only for a reduced set of turn by turn data. Therefore, the global fit was abandoned and another, more robust method was chosen.

The selected method is based on the fact that information on energy spread is contained in the modulation of the envelope curve of the beam oscillation, thus it is sufficient to perform a fit on the envelope only and do not consider the fast varying sine-term in eq. (19). This requires interpolation of the fast-varying betatron oscillation.

According to the theory betatron oscillations are described by a sinusoidal function (see eq. (8)). Hence, in order to get the real maxima for the betatron oscillations one needs to apply a sinusoidal function fit to the data in order to restore the complete oscillation. In fig. (12) one can see turn-by-turn data from a single BPM as well as the evaluation by many turns in fig. (7). One can see that the sine is defined only by several points and in reality these points usually were not enough to clearly fit the sine function. The fit was usually fragile and it was decided to change the method and combine data of all the BPMs, to get robust fit and better statistics.

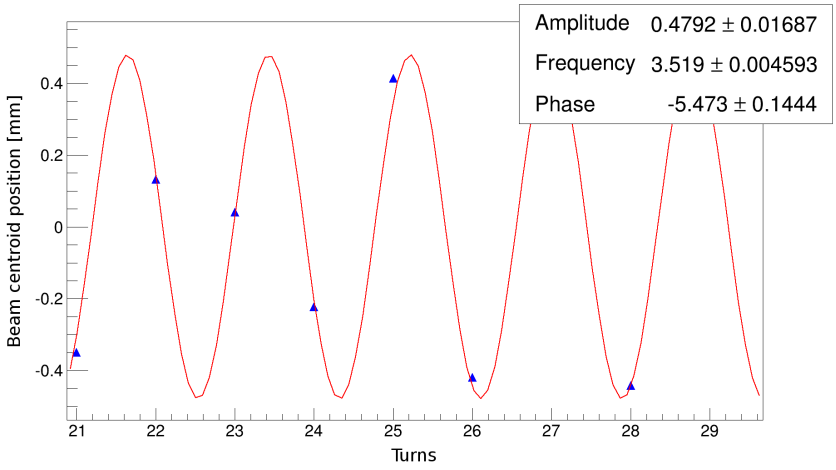

Figure 12: Sine fitted betatron oscillation points taken from one BPM (without error bars).

It was decided to merge all the BPMs to get more realistic turn-by-turn propagation of the beam which means an individual approach to each BPM. Merging the BPM data also has more support from point of view of physics, since the correlations between BPMs due to machine optics are known. Merging of data from 73 BPMs should be done with phase and amplitude normalizations, to get the same scaling. Let's find the coefficients which will lead us to the combined data.

The beam position in the i^{th} location during the n^{th} turn is described by

$$x_i = \mathcal{A}\sqrt{\beta_i}\sin(2\pi(n\nu_0 + \Delta\nu_i)), \tag{31}$$

where β_i is the beta function value at given BPM, $-0.5 \leq \nu_0 \leq 0.5$ is the fractional betatron tune of the ring and $\Delta\nu_i$ is the fractional betatron phase of the ith BPM. In order to normalize it we demand

$$\hat{x_i} \stackrel{!}{=} \mathcal{A}\sin[2\pi(n+t)\nu_0]. \tag{32}$$

So, we get for t and $\hat{x_i}$

$$t = \frac{\Delta\nu_i}{\nu_0} \qquad \hat{x_i} = \frac{x_i}{\sqrt{\beta_i}}. \tag{33}$$

So, for normalization by phase we need the β function value at each BPM. These values were obtained from MAD-X simulation program, where the SLS lattice was implemented and the β values were obtained. The corresponding output data file needs to be cleaned-up before reading it in the program, this was done by small Mathematica script. Applying the normalization, the betatron oscillation can be restored very well (fig. (13)).

For betatron oscillations normalization procedure gives us the clearly defined behaviour, which can be seen in the zoomed-in picture of the betatron oscillation (fig. (14)). This new data are well defined enough to have a robust fit. One can see in fig. 15 that the sine is fitted using many points and is a robust fit.

In order to get the envelope we take a sliding window and perform sine fits within this window on the betatron oscillations. Due to normalization there are many points within a step of one turn when moving the window. So, the sine fit was done moving on with one turn step and taking the window around this point. A step size corresponding to one turn provides sufficient resolution for the envelope of the betatron oscillation. Smaller step for moving the sliding window will only cause computational time increase. Taking the amplitudes of the sine interpolations we will get the envelope behaviour of the oscillation. After this procedure we can do

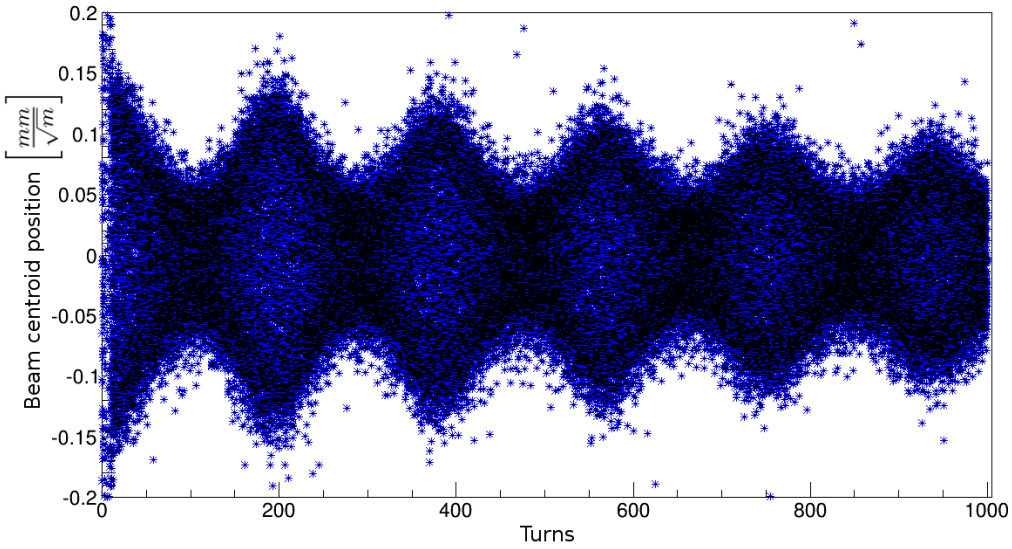

Figure 13: Summed data from 73 BPMs including the $20\,\mu m$ errors.

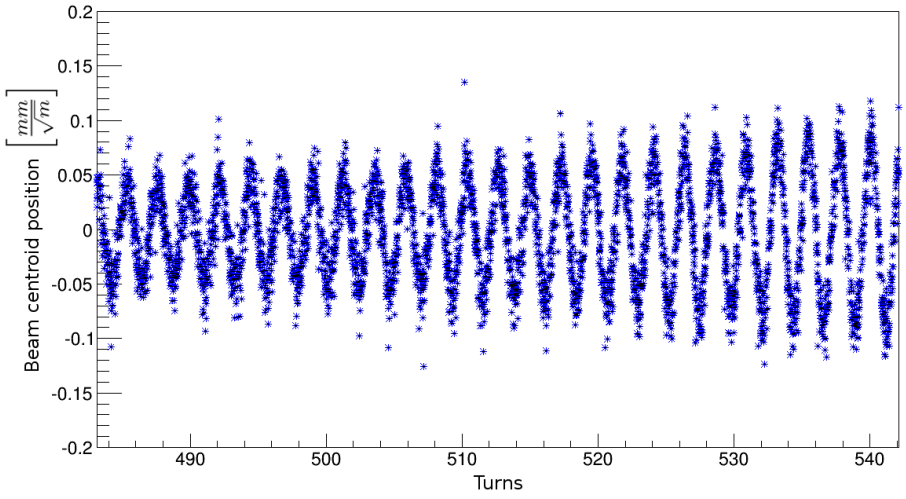

Figure 14: Magnification of a region of the fig. (13).

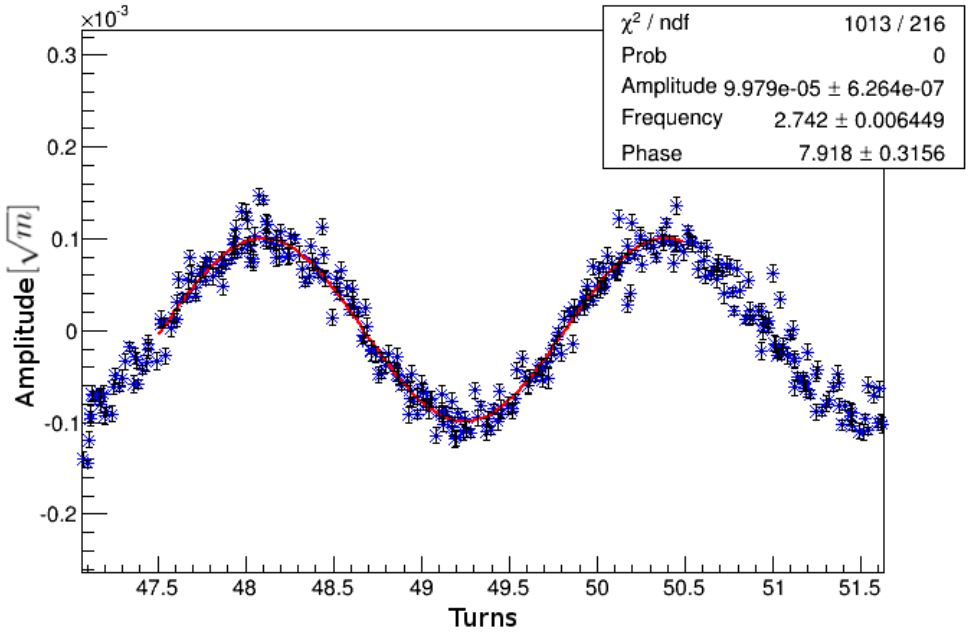

Figure 15: Combined data with the sine fit.

our envelope fit neglecting the fast varying betatron oscillation term in eq. (19):

$$\langle \hat{x}(N) \rangle = \mathcal{A}\, A(N) H(N) e^{-M(N)}. \tag{34}$$

Overall number of parameters to be fitted is equal to 7, 4 of which are crucial for the description of the envelope. These parameters are contained within the parameter M, particularly in the parameters K_1 and K_2. These 4 parameters are the 1^{st} and 2^{nd} order chromaticities, the synchrotron tune, and the energy spread itself. As it was mentioned in sec. 3.2 and sec. 3.3 chromaticity and synchrotron tune come from the measurements. For the 2^{nd} order chromaticity an estimate was taken from the measurements. Taking the measured value of ξ_2 as a fixed parameter was not resulting in better envelope fits, since its error bars were large. Having these parameters and fitting the envelope function of the beam centroid we obtain the required energy spread. A typical example of a good fit is presented in fig. (16).

5 Error calculation

Error propagation plays an important role in different stages of the data analysis. Particularly, passing the calculated errors to further steps of analysis allows to include inevitable position measurement errors, calibration issues etc.

There are two sources of errors: hardware based, coming from measurement accuracy and analysis based, coming from different fits during the analysis steps. First source of error is hardware based, i.e. measurement accuracy based. This includes three different effects which disturb the position measurement values. The first effect is the beam position measurement accuracy which is about 20 μm error [16]. The second effect is connected with the calibration of the BPMs. The third uncertainty is introduced by the measurement error of the β function value [17].

The errors are evaluated by the combined measured data error evaluation formula. Assuming that $x = f(a, b, c)$, the standard deviation σ is:

$$\sigma_{x_A} = \sqrt{\left[\left(\frac{\partial x}{\partial a}\right)_A \sigma_{\bar{a}}\right]^2 + \left[\left(\frac{\partial x}{\partial b}\right)_A \sigma_{\bar{b}}\right]^2 + \left[\left(\frac{\partial x}{\partial c}\right)_A \sigma_{\bar{c}}\right]^2 + \cdots} \tag{35}$$

Partial derivatives $\partial x/\partial a, \partial x/\partial b$, etc. in eq. (35) state how x would change, if one changes *only a, only b,* or *only c*, keeping the other quantities constant. The index A in the partial derivatives states that the numerical values of the partial derivatives are to be calculated at the means $\bar{a}, \bar{b}, \bar{c}$ of a, b, c. Dots in the end of (35) represent other mixed derivatives i.e. derivatives giving correlation between parameters. These parameters can be ignored to a good approximation based on negligible correlation effects. For our error propagation eq. (35) will take the form:

$$\Delta\hat{x}_{out} = \sqrt{\left(\frac{\partial \hat{x}_{in}}{\partial x} \cdot \Delta x_{noise}\right)^2 + \left(\frac{\partial \hat{x}_{in}}{\partial x} \cdot \Delta x_{calib}\right)^2 + \left(\frac{\partial \hat{x}_{in}}{\partial \beta} \cdot \Delta\beta\right)^2}, \tag{36}$$

where Δx_{noise} is the beam centroid position measurement accuracy, Δx_{calib} is the uncertainty for the calibration of the BPMs and $\Delta\beta$ is the uncertainty of the β function value. In (36) $\hat{x}_{in}$ is the normalized position of the centroid described by:

$$\hat{x}_{in} = \frac{x}{\sqrt{\beta}}. \tag{37}$$

For details on the normalization procedure see section 4.

Plugging eq. (37) into eq. (36) we get:

$$\Delta\hat{x}_{out} = \sqrt{\left(\frac{1}{\sqrt{\beta}} \cdot \Delta x_{noise}\right)^2 + \left(\frac{x}{\sqrt{\beta}} \cdot \frac{\Delta x_{calib}}{x}\right)^2 + \left(-\frac{x}{\sqrt{\beta}}\frac{1}{2} \cdot \frac{\Delta\beta}{\beta}\right)^2}, \tag{38}$$

where Δx_{noise} is of order $20\,\mu m$ (see table 1 in [16]), Δx_{calib} is the calibration error which was estimated to be of 1 % effect and $\frac{\Delta\beta}{\beta}$ is the β value error of 5 % (see [17] for more detail). For β error values of 4 % and 3 % are given for horizontal and vertical planes, therefore an approximate of $\approx 5\,\%$ is a reasonable assumption. Putting the mentioned error values into the eq. (38) and simplifying one gets:

$$\begin{aligned}\Delta\hat{x}_{out} &= \sqrt{\left(\frac{1}{\sqrt{\beta}}\cdot 20\cdot 10^{-6}\right)^2 + \left(\frac{x}{\sqrt{\beta}}\cdot 0.01\right)^2 + \left(\frac{x}{2\sqrt{\beta}}\cdot 0.05\right)^2} \\ &= \sqrt{\frac{1}{\beta}\left((20\cdot 10^{-6})^2 + (x\cdot 0.01)^2 + \left(\frac{x}{2}\cdot 0.05\right)^2\right)}.\end{aligned}$$

This "hardware based" error propagation can be seen in fig. 15.

As it was discussed in sec. 4 after combining the data from all BPMs we do a sine interpolation. Sine interpolation is actually a fit procedure which assumes errors. So, doing these fits in the sliding window we get the amplitude with error bars which will be passed further to the envelope fit (fig. (16)). The fit of the envelope function will introduce errors for the varying parameters, particularly for the energy spread.

The last step of error propagation is the "hardware based" error from the chromaticity measurement. Remembering the equation from sec. 3.2:

$$\delta = -\frac{1}{\alpha}\frac{\Delta f}{f}, \tag{39}$$

we see that there are three possible sources of uncertainities. Particularly, during the measurements we have uncertainities in the measurements of *tune* and *RF-frequency*. For the tune measurements with the FFT interpolation one has uncertainty of 0.001, which is a typical value of tune fluctuation in the control room [9]. For the RF-cavity frequency errors are much smaller and are of the order of 10^{-8}. Another possible source of error is the momentum compaction factor, which can be obtained at high precision from measurements of beam energy by spin depolarization for different path lengths, set by variation of the radio-frequency (RF) [10]. The value measured at the SLS is $\alpha = 6.0\cdot 10^{-4}$ which is in agreement with the designed value. Since the energy is measured at a precision of 10^{-5}, and the RF at 10^{-8}, the error of the momentum compaction factor may be neglected. Finally we are left with two sources of errors. Both errors where taken

into account during the polynomial fit procedure as it is seen in fig. (8) and (9). The error of the momentum measurement was taken into account but is too small to be seen in the figures.

The polynomial fit procedure introduces a new error which is contained in the chromaticity. Moreover, chromaticity is included in one of the important fitting parameters: eq. (20d) and (20e). The error from the chromaticity will evolve obeying the *Gaussian error propagation law* eq. (35). Thus, the final energy spread error will be changed by

$$\left(\Delta\sigma_{\delta\,total}\right)^2 = \left(\Delta\sigma_\delta\right)^2 + \left(\sigma_\delta \frac{\Delta\xi}{\xi}\right)^2 . \tag{40}$$

6 Results

Measurements of the motion of the beam centroid is examined for horizontal and vertical planes separately in order to avoid possible correlation effects due to coupling between the two plane kicks. For each (horizontal or vertical) measurement both plane data is taken, out of which only the kicked plane is examined. In each plane three different beam/bunch currents are analysed. For each of the planes three different values for chromaticity and pinger currents are taken. Choice of chromaticity and pinger current is not taken for high bunch currents because of beam loss. The measurement data sheet given in table 3 shows which combinations of parameters were taken and which were successful. The application of the fit procedure described in 4 to real data is quite time-consuming, as sometimes the initial guesses for the fit parameters need individual adjustments, so a full automation is not possible.

6.1 Envelope fits

In the following, we analyse the envelope fit behaviour depending on different beam/bunch currents and pinger kicks. Narration is sorted by the beam/bunch current. As mentioned fits were not always autonomous and sometimes required manual tweaking of the range of the fit. Health report of the fits done is given in the table 3.

I_b = 40 mA + 1.8 mA in 50 bunches

ξ	I_{PIX}	PIH	Report	PIV	Report
5	100	12:47:00	success	12:52:30	cut at 600
	300	12:49:21	success	12:54:13	success
	700	12:51:22	cut at 650	-	-
7	100	13:00:30	success	13:01:10	cut at 600
	300	12:58:31	success	13:02:16	success
	700	12:59:01	cut at 650	-	-
3	100	13:03:20	success	13:06:03	success
	300	13:04:43	success	13:07:02	success
	700	13:05:20	cut at 600	-	-

I_b = 20 mA + 2 mA in 12 bunches

ξ	I_{PIX}	PIH	Report	PIV	Report
5	100	13:15:37	success	13:18:03	cut at 800
	300	13:16:08	success	13:19:01	success
	700	13:17:04	cut at 450	-	-
7	100	13:21:03	success	13:24:02	cut at 700
	300	13:22:02	success	13:26:37	success
	700	13:23:04	cut at 450	-	-
3	100	13:27:13	success	13:31:12	success
	300	13:28:01	success	13:32:01	fail
	700	13:29:11	cut at 430	-	-

I_b = 15 mA + 2 mA in 3 bunches

ξ	I_{PIX}	PIH	Report	PIV	Report
5	100	13:35:04	fail	13:38:06	success
	300	13:36:41	cut at 300	13:39:00	success
	700	13:37:33	cut at 300	-	-
3	100	13:40:24	cut at 300	-	-
	300	13:41:03	cut at 300	-	-
	700	-	beam loss	-	-

Table 3: Data files' health report. Red file names correspond to files which were discussed.

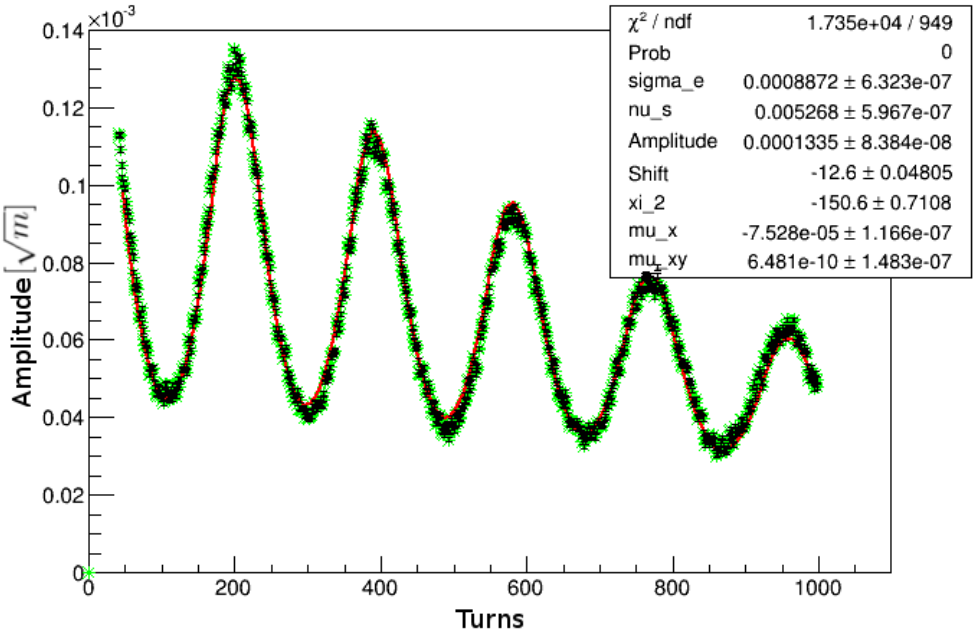

Figure 16: Fit of `22Jul12:47:05_0.png`. Envelope fit for horizontal excitation at 100 A pinger amplitude.

We will proceed with analysis of different fits. Only the cases which are extraordinary will be discussed in detail.

6.1.1 Beam current of 40 mA, 0.8 mA in a bunch

The first group of data were taken at a beam current of 40 mA with a variation of error of 1.8 mA due to top-up operation. The number of bunches in the beam was 50, which corresponds to a current of 0.8 mA per bunch.

First let's see a good case of a fit. A typical example of a good fit is at a set chromaticity of 5 and pinger current of 100 A is shown in fig. (16).

At a very large pinger current of 700 A distortions are visible after turn 650 which can not be fitted, see fig. (17).

However, we see that the data before turn 650 are informative and is sufficient to be used for the energy spread extraction. In figure 17 we see that the first part has still acceptable fit shape even though it is influenced by the later turns. Nevertheless, to avoid the distorting influence of the "tail" the fit was done only

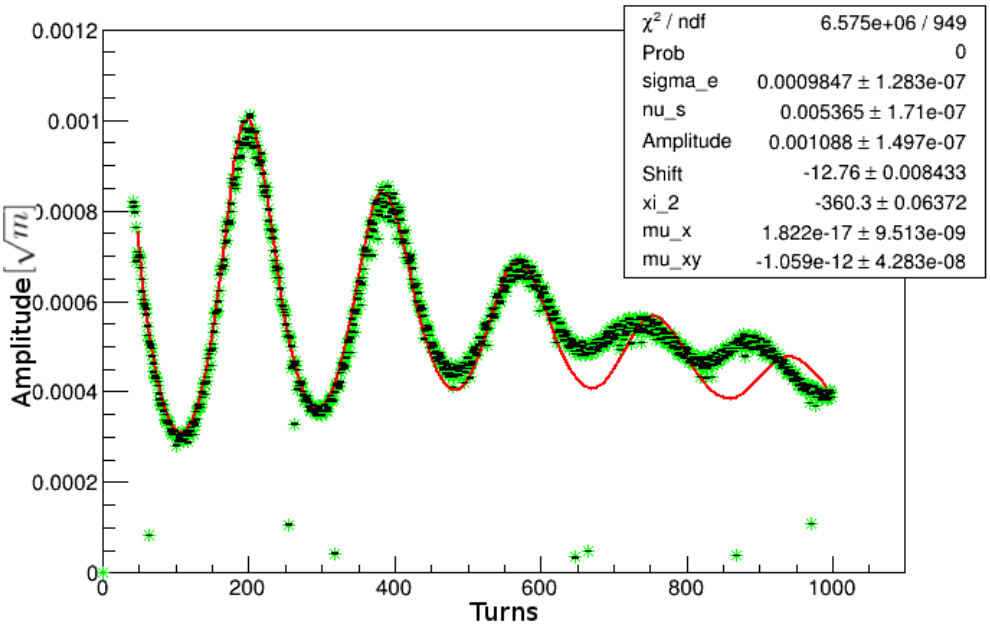

Figure 17: Fit of `22Jul12:51:28_0.png`. Envelope fit for horizontal excitation at 700 A pinger amplitude.

on the part of the modulations, particularly on first 650 turns.

The described techniques are usually successful. Examining the case for chromaticity of 7 with pinger current of 700 A in fig. (18), we see that even after the cut fit of the 650 turns is not perfect and around turn 480 and later we see mismatch with the data.

Now let's examine the data for vertical excitation. The data are more noisy, since the vertical pinger provides a smaller kick than the horizontal one at same current. So the amplitude of the betatron oscillation is smaller vertially and thus more affected by the BPM noise. Even though the fit is done for all 1000 turns, one can see that there is some periodic effect which disturbs the shape of the envelope modulations (fig. (19)). Perhaps it is due to a coupling between the vertical and the horizontal betatron oscillations.

b

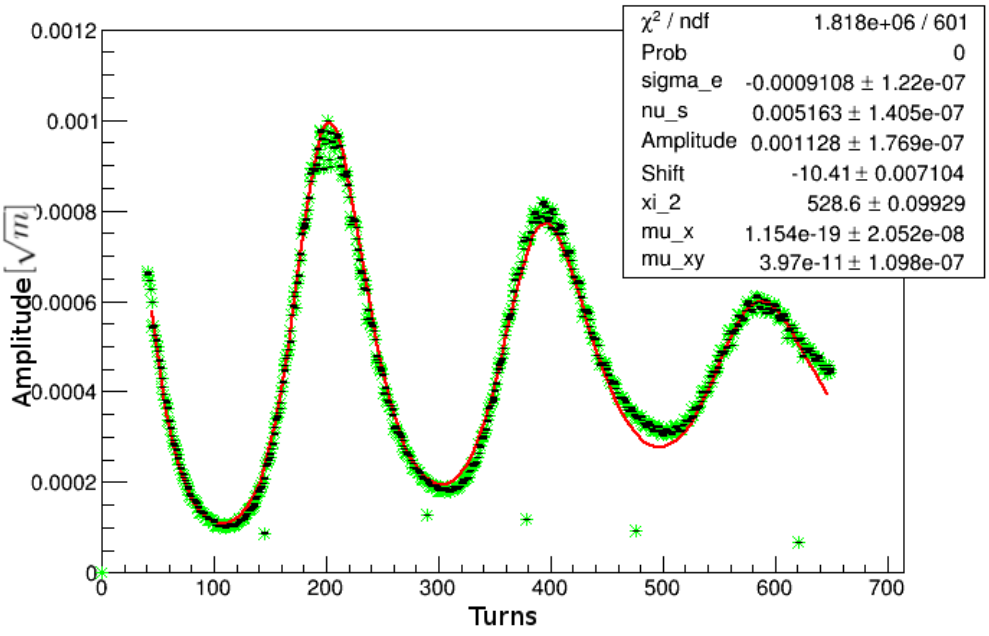

Figure 18: Fit of `22Jul12:59:13_0.png`. Envelope fit for horizontal excitation at 700 A pinger amplitude.

6.1.2 Beam current of 20 mA, 1.66 mA in a bunch

Let's start from an example of a good fit for this bunch current. Such an example is the fit for chromaticity of 3 with pinger kick current of 700 A (see fig. (20)).

Now let's examine a case where in contrast to fig. 17 the "tail" disturbs the fit to an unacceptable limit (see fig. (21)). As one can see the data is not meaningless for all turns. The first 430 turns represent expected modulations, while the "tail" fails to fit properly. Thus, we take only the first 430 turns in order to obtain an acceptable fit and required energy spread. The cut and fitted result is shown in fig. (22). Comparing the χ^2 values of the two fits we see an improvement by one order.

Another extreme case of a fit are the noisy data sets, where only 1 in 10 files gave an acceptable fit. Such an example is for a chromaticity of 5 with a pinger current of 100 A in the vertical plane (see fig. (23)). In this case, the turn by turn signal is obscured by noise due to too low excitation, since the vertical pinger is

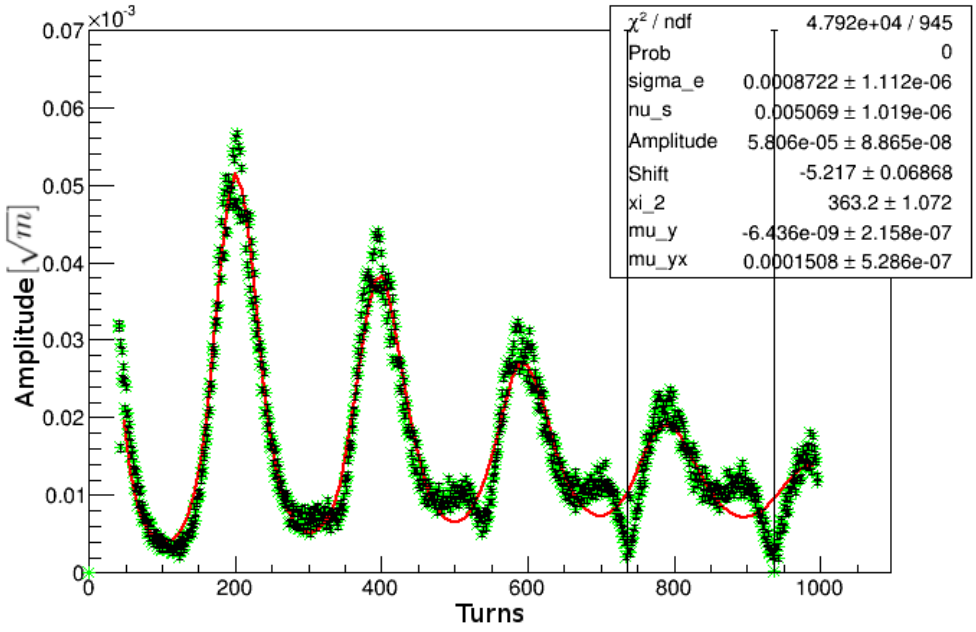

Figure 19: Fit of `22Jul13:02:27_1.png`. Envelope fit for vertical excitation at 300 A pinger amplitude.

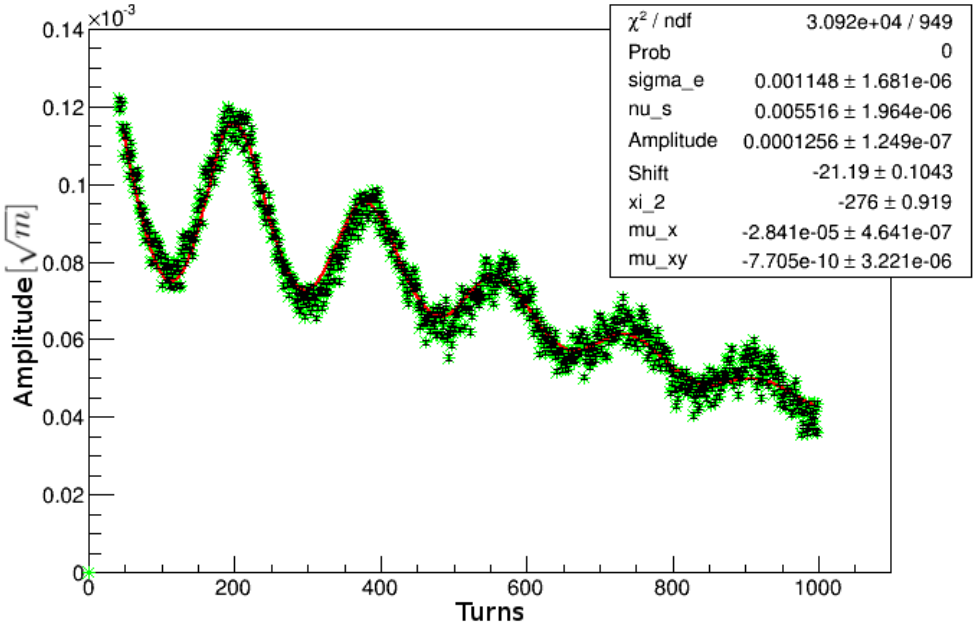

Figure 20: Fit of `22Jul13:27:21_0.png`. Envelope fit for horizontal excitation at 700 A pinger amplitude.

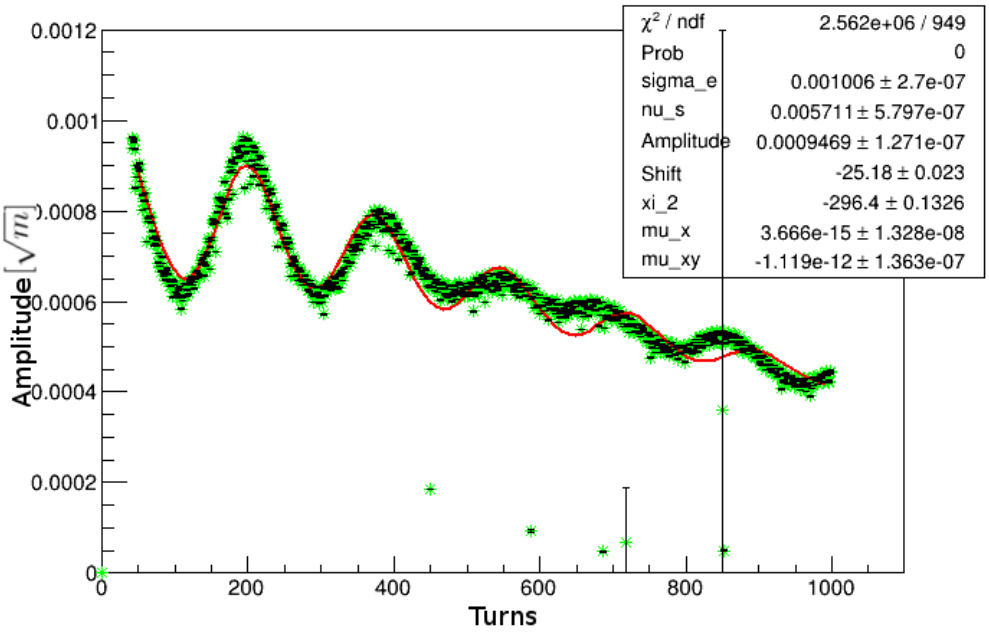

Figure 21: Fit of `22Jul13:29:17_0.png`. Envelope fit for horizontal excitation at 700 A pinger amplitude.

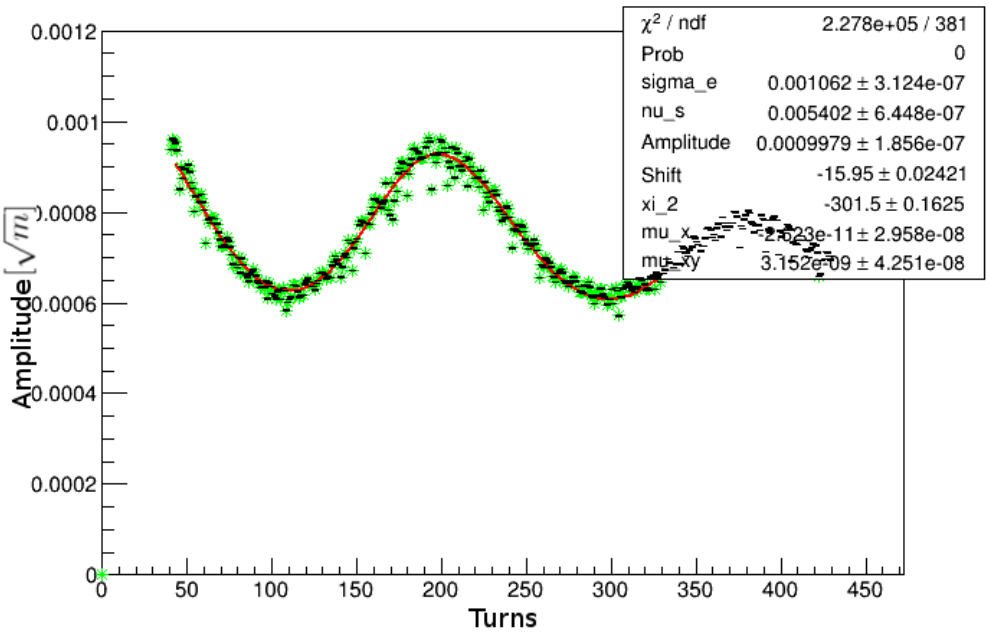

Figure 22: Fit of cut `22Jul13:29:17_0_cut.png` by first 430 turns. Envelope fit for horizontal excitation at 700 A pinger amplitude.

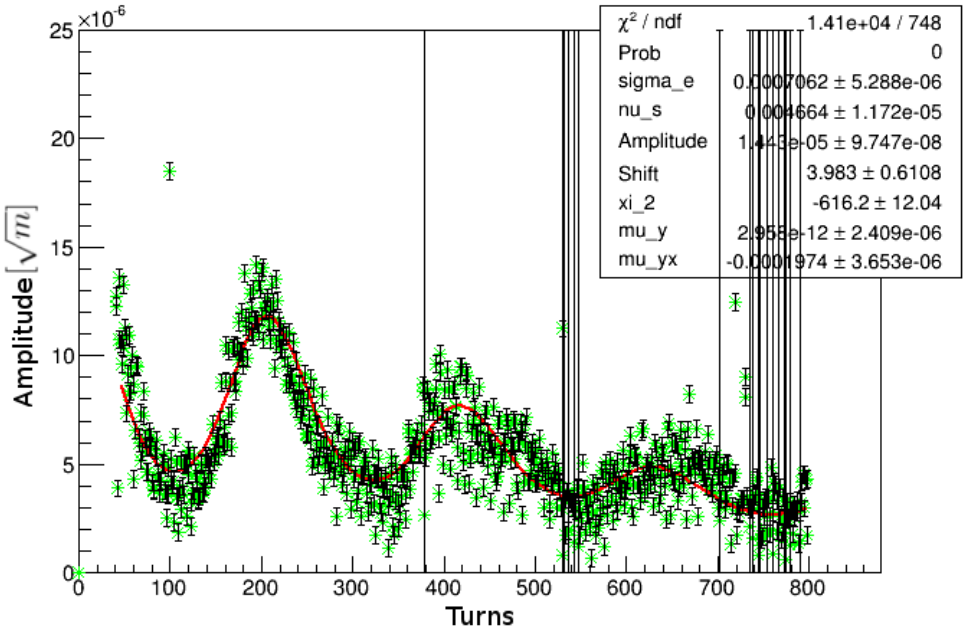

Figure 23: Fit of `22Jul13:18:04_1.png`. Envelope fit for vertical excitation at 100 A pinger amplitude.

weaker than the horizontal one.

Along successful fits there are also failures, where non of the 10 files had an acceptable fit. Such an example is for a set chromaticity of 3 with pinger current of 300 A because at low chromaticity, the modulation depth is rather small. The best fit of such data is given in fig. (24).

6.1.3 Beam current of 15 mA, 5 mA in a bunch

Finally let's discuss data sets for high beam currents of 5 mA per bunch. The case of chromaticity of 5 and kick of 300 A shows that the modulations are dominated by other effects, thus only the first 300 turns were taken as basis for the fit (fig. (25)).

Another extreme case for this beam energy is present in vertical data set, particularly for chromaticity of 5 with kick of 300 A (see fig. (26)).

We can conclude from the observations of the above fits, that the fit becomes more difficult, when we go to higher beam/bunch currents. Probably there are cur-

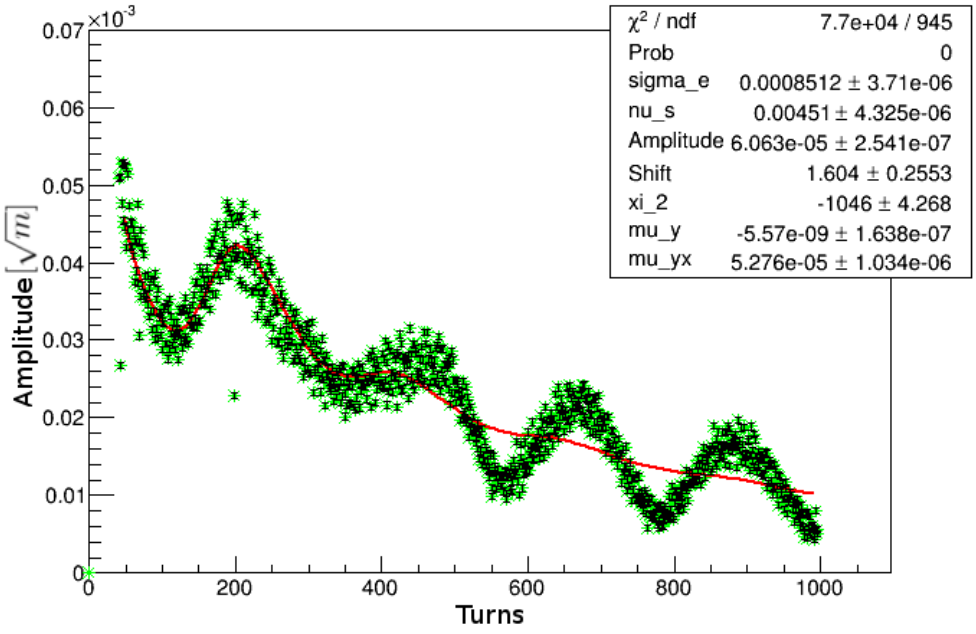

Figure 24: Fit of `22Jul13:32:11_1.png`. Envelope fit for vertical excitation at 300 A pinger amplitude.

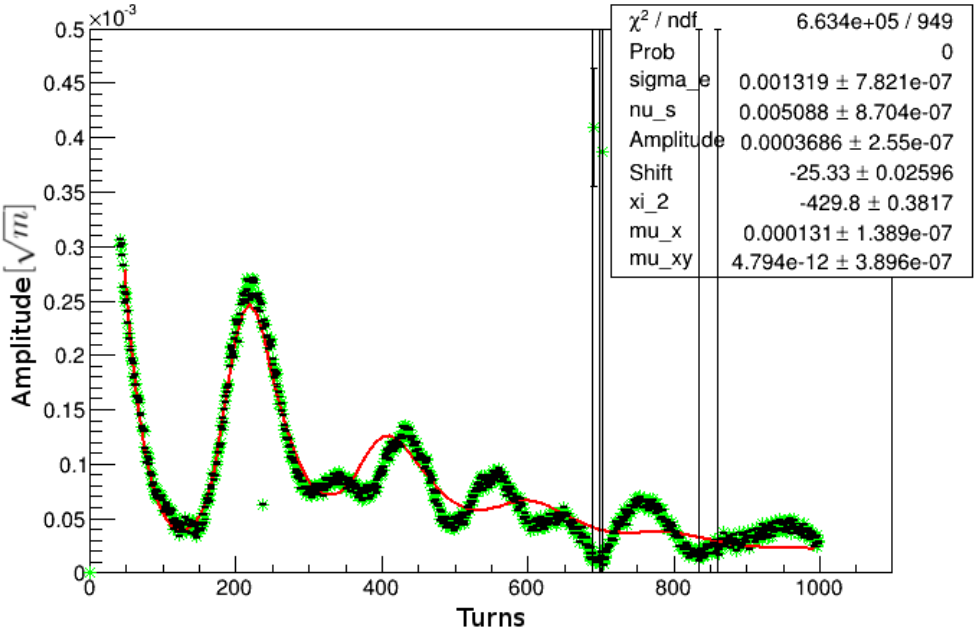

Figure 25: Fit of `22Jul13:36:43_0.png`. Envelope fit for horizontal excitation at 300 A pinger amplitude.

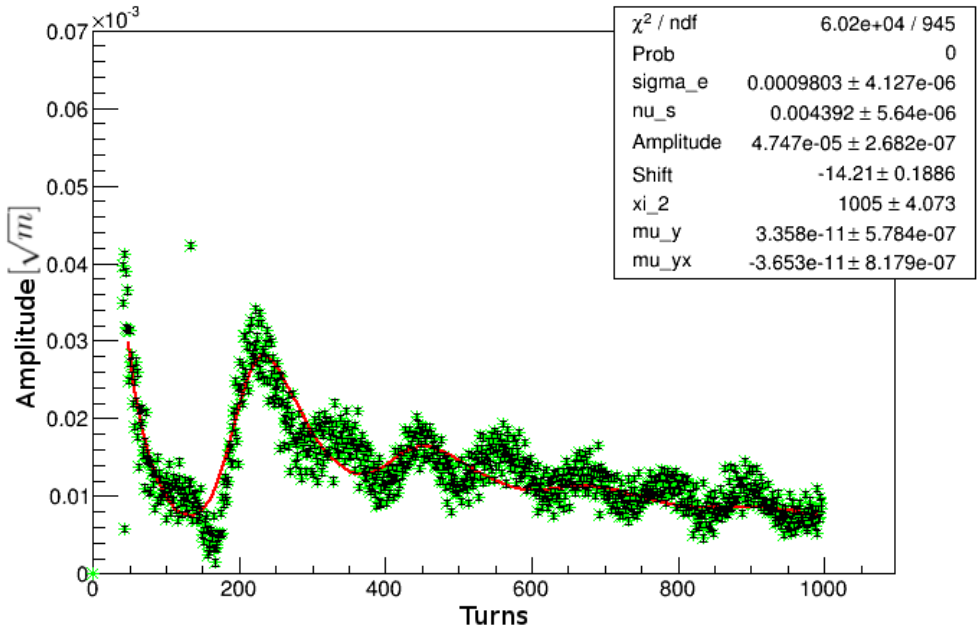

Figure 26: Fit of `22Jul13:39:00_1.png`. Envelope fit for vertical excitation at 300 A pinger amplitude.

rent dependent effects perturbing the betatron oscillation which were not included in the decoherence model we use.

6.1.4 Turbulent bunch lengthening and the results

In order to estimate the quality of our results of data processing it should be compared to the theoretical calculations. Theoretical calculations of bunch length and energy spread as a function of bunch current had been confirmed experimentally [5]. As it was mentioned in sec. 2.5 energy spread behaviour due to TBL is:

$$\frac{\sigma_\delta}{\sigma_{\delta 0}} \sim (I_b[mA]/1.1)^{\frac{1}{3}} for\ I_b > I_b^{th} \tag{41}$$

This work was done in 2001 and corresponds to the state of the machine in that year. Thus, it may not fully correspond to the present machine parameters.

In our measurement data sets with chromaticity values of 3, 5 and 7 were analysed. They are marked by different colors in fig. (27). The symbols represent the pinger current by which the beam was kicked. During this measurement three different pinger amplitude values were chosen: 100, 300 and 700 A. At each of the amplitudes the beam was kicked in 3 different accelerator setups. First setup was 40 mA beam current in 50 bunches with an error value of current from Top-Up of 1.8 mA. Second setup corresponded to beam current of 20 mA in 12 bunches and 2 mA current error. Third setup which was analysed was $15mA$ of beam current in 3 bunches and error value of Top-up current of 2 mA. In other words, bunch currents of 0.8 mA, 1.67 mA and 5 mA were taken. These currents are chosen to check the energy spread change due to TBL.

$$x_{rms} = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}} \tag{42}$$

The following table 4 compares results obtained and expected as a function of bunch current. As well as fit quality check RMS value of the obtained data was calculated by the eq. (42):

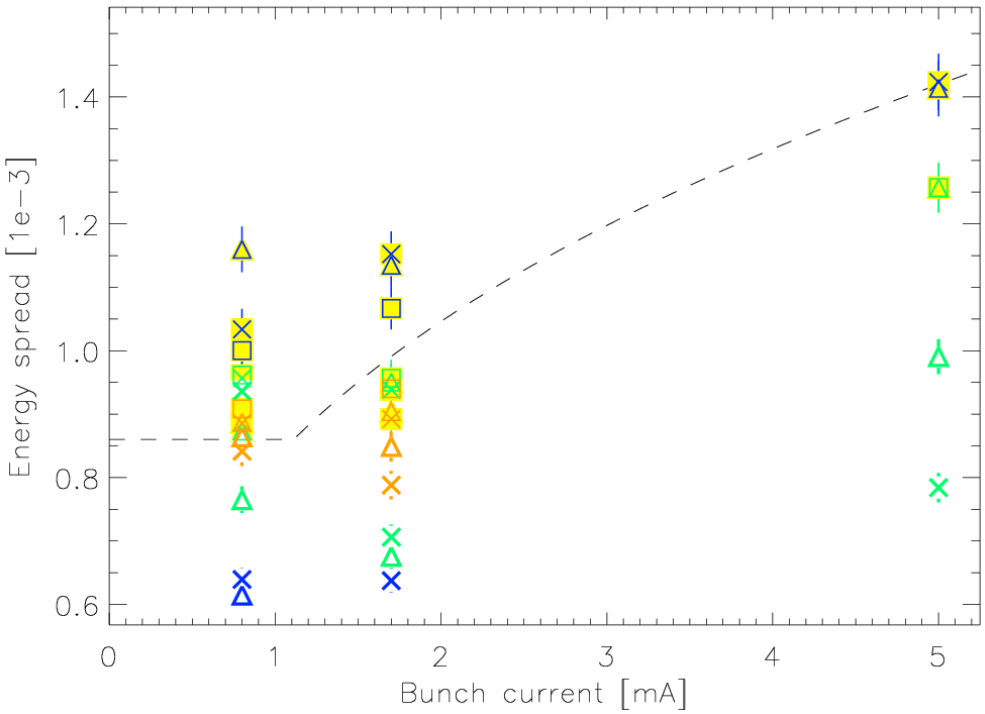

Figure 27: Energy dependence versus the beam current. Yellow background with thin lines correspond to the horizontal datasets and points without background and with thick lines correspond to the vertical datasets. Blue, green and orange are for chromaticities of 3, 5, and 7 respectively. Crosses, triangles and squares are for pinger currents 100, 300, 700 A respectively. Dashed line represents the expected theoretical variation of energy spread with bunch current from turbulent bunch lengthening.

	0.8 mA	1.67 mA	5 mA
Obtained	$0.89 \cdot 10^{-3}$	$0.9 \cdot 10^{-3}$	$1.22 \cdot 10^{-3}$
RMS	$0.24 \cdot 10^{-3}$	$0.22 \cdot 10^{-3}$	$0.34 \cdot 10^{-3}$
Expected	$0.86 \cdot 10^{-3}$	$1.05 \cdot 10^{-3}$	$1.4 \cdot 10^{-3}$

Table 4: Obtained and expected values of the energy spread.

For high beam currents measurement data was usually noisy and the energy spread values were usually obtained from 1-2 fits of the envelope. As it can be

seen in table 5 in the last row, some of the points were obtained from just one fit of the envelope.

During our analysis we have neglected the influence of the passive third harmonic cavity used at the SLS to lengthen the bunches, because the total beam currents used here are small compared to the 400 mA beam current in user operation mode. Nevertheless, if there would have been some influence, this could explain why the energy spread obtained at high bunch current is lower than expected.

The results show a systematic dependence on bunch current, but no correlation to pinger current and chromaticity - which is the expected behaviour of the energy spread.

7 Conclusion

The rms beam energy spread at the SLS was measured from the decoherence and recoherence after a transverse kick to the beam applied by a pinger magnet. Turn by turn signals were recorded from all 73 BPMs and merged using theoretical values of beta functions and phases. Local sine fits in a sliding window over the BPM data were used to interpolate the amplitude of the beam oscillation. A fit of the theoretical decoherence formula was performed to the amplitude data. Seven parameters were fitted: energy spread, synchrotron tune, 2^{nd} order chromaticity, two amplitude dependent tune shift terms, the pinger amplitude and a temporal delay. Initial guesses for these parameters were taken from theory, except for the synchrotron tune, which was taken from the side-bands of the beam's Fourier spectrum and 2^{nd} order chromaticity which was obtained from tune-RF-frequency dependence measurement fits. The first order chromaticity was used as a given parameter and measured before as variation of tune vs. radio frequency.

Beam excitation was performed vertically and horizontally with different pinger currents and for different chromaticity settings. The bunch current was varied to study the dependence of the energy spread on the bunch current and prove theoretical models and previous measurements on turbulent bunch lengthening.

At low current, the theoretical energy spread was confirmed very well. At higher currents, the measured values were slightly lower than the expected ones. The fit, i.e. the agreement of the theoretical decoherence function with the measured beam response, was best at low beam current and for moderate pinger currents.

8 Acknowledgements

I am thankful to my supervisors Andreas Streun and Andreas Adelmann for their encouragement, guidance and support during the period of my work on my thesis.

I am also thankful to Swiss Light Source staff for their patient and detailed explanations.

Դեպի լավը...

Translation: Towards prosperity.

Alternative translation: Towards well-being.

9 References

References

[1] T. Tarazona, P. Elleaume, "Measurements of the absolute energy and energy spread of the ESRF electron beam using undulator radiation", Rev. Scientific Instruments, 67(1996) p.3368.

[2] T. Nakamura et al, Chromaticity for energy spread measurement, Proc. PAC 2001, Chicago, USA, p. 1972.

[3] I.C.Hsu, The decoherence and recoherence of the betatron signal and an application , Particle Accelerators 34(1990), p.43

[4] E. D. Courant, H. S. Snyder "Theory of the Alternating-Gradient Synchrotron", Brookhaven National Laboratory, Upton, New York

[5] A. Streun, "Beam Lifetime in the SLS Storage Ring", Paul Scherrer Institut, Villigen, 2001

[6] K. Wille, "The Physics of Particle Accelerators an introduction," Oxford 2000.

[7] M.Sands, "The Physics Of Electron Storage Rings: An Introduction", University Of California, Santa Cruz, 1970.

[8] H. Wiedemann "Particle Accelerator Physics", Springer, 2007

[9] H. Press "Numerical recipes in C++ : the art of scientific computing", Cambridge, 2005

[10] S.C. Leeman, "Precise Beam Energy Calibration At The Sls Storage Ring", Proceedings of EPAC 2002, Paris, France.

[11] A. Sargsyan, "Transverse decoherence of the kicked beams due to amplitude and chromaticity tune shifts", Nuclear Instruments and Methods in Physics Research A, Elsevier 2011.

[12] P. Marchand, "Possible Upgrading of the SLS RF System for Improving The Beam Lifetime", Paul Scherrer Institute, Villigen PSI.

[13] EPICS, http://www.aps.anl.gov/epics

[14] J.Chrin, M.C. Sloan, "Cafe, A Modern C++ Interface To The Epics Channel Access Library" http://accelconf.web.cern.ch/AccelConf/icalepcs2011/papers/wepks024.pdf

[15] Naming Convention at SLS: http://epics.web.psi.ch/style/training/handouts/e_basehandouts/node20.html

[16] V. Schlott et al., New Digital BPM System for the Swiss Light Source, Proc. DIPAC 1999, Chester, UK, p. 168. In the web: http://accelconf.web.cern.ch/AccelConf/d99/papers/PT06.pdf

[17] M. Boege, A. Streun, V. Schlott, Measurement and correction of imperfections in the SLS storage ring, Proc. EPAC 2002, Paris, France, p. 1127. In the web: http://accelconf.web.cern.ch/AccelConf/e02/PAPERS/WEPLE007.pdf

A ROOT codes

Data analysis used in this work can be done using the two programs listed below. Prior to run the program one should set the correct parameters for his/her experimental data in lines 33-43.The next step is to check the files over which the program will run on lines 522 to 526. In case of manual fitting additionally lines 661, 702 and 734 should be corrected to the desired limits. If one wants to do the global fit, line

Some comments on the chromaticity measurement analysis. This program is included in the main fit analysis for getting the chromaticity values. It is called at the beginning of the `main()` function. Besides being called in main program, it can also be executed seperately by passing to the function the axis (0 or 1) and the order of chromaticity. Other parameters `k, b, k_error` and `b_error` can be put randomly.

Data analysis used in this work can be done using the two programs listed below. If one wants to replicate the results, he/she should check if the lines between 33 to 43 are correct. This lines are the parameter sets used during the experiment. The next thing that should be checked is the files over which the program will run. These can be adjusted on lines 522 to 526. In case of manual fitting additionally lines 661, 702 and 734 should be corrected to the desired limits.

The other is the chromaticity measurement analysis. This program is included in the main fit analysis for getting the chromaticity values. It called in the beginning of the `main()` function. Besides being called in main program, it can also be executed seperate by passing to the function the axis (0 or 1) and the order of chromaticity. Other parameters `k, b, k_error` and `b_error` can be put randomly.

A.1 Main data analysis code

Listing 1: Main fit code

```
#include <iostream>
using namespace std;
#include <fstream>
#include <cstdio>
#include <iomanip>
#include "TMath.h"
#include "TH1.h"
#include "TF1.h"
#include "TGraph.h"
#include "TGraphErrors.h"
#include "TStyle.h"

#include "chrom_analysis.C"

#define AV_POINTS 3
#define N 1000
#define N_all 73000; // 73 * 1000
#define excl_points 20
#define begin_cut 40 // 0-padding length from begining
#define num_files 10 // Number of files to be analysed

// Physical definitions
#define nu_x_0 20.436
#define nu_y_0 8.737
#define eps_x 5.5e-9 // Emmitance
#define eps_y 5e-12
#define dnu_xdJ_x -1300
#define dnu_ydJ_y -1100
#define dnu_ydJ_x 600
#define dnu_xdJ_y dnu_ydJ_x // mixed derivatives are the same
#define delta_ys 0.001 // Angle of kick

////////////////////////////////////////////////////////////
// Check this values before running the analysis

double axis = 0;
double chromaticity = 5;
double PIX = 100;
double I_b = 40;
double I_b_error = 1.8;
double num_bunch = 50;

////////////////////////////////////////////////////////////

double WINDOW = 0;

double xi_1,xi_2 = 0;
double xi_1_error,xi_2_error = 0;
double z,mu_u, mu_vu = 0;
double z_part = 0;
Double_t mu_x = eps_x * dnu_xdJ_x;
Double_t mu_y = eps_y * dnu_ydJ_y;
```

```
Double_t mu_xy = eps_y * dnu_xdJ_y;
Double_t mu_yx = eps_x * dnu_ydJ_x;

char fdsname[260] = {0};
char datasetname[260] = {0};

double mean_sigma_e_error = 0;
int ds_counter = 0;
double mean  = 0.;
double mean_Chisquare = 0.;
double mean_nu_s = 0.;
double mean_nu_s_error = 0.;
double mean_Amplitude = 0.;
double mean_Amplitude_error = 0.;
double mean_Shift = 0.;
double mean_Shift_error = 0.;
double mean_xi_2 = 0.;
double mean_xi_2_error = 0.;
double mean_mu_x = 0.;
double mean_mu_x_error = 0.;
double mean_mu_yx = 0.;
double mean_mu_yx_error = 0.;

I_b /= num_bunch;
I_b_error /= num_bunch;

Double_t chrom_val(int axis, int chrom_order, Double_t req_chrom, bool error)
{
  Double_t chrom_exp = 0.;
  Double_t chrom_exp_error = 0.;
  double k,k_error,b,b_error = 0.;

  chrom(axis,chrom_order,k,b,k_error,b_error);
  chrom_exp = k * req_chrom + b;
  chrom_exp_error = k_error * req_chrom + b_error;

  if (!error)
    return chrom_exp;
  else
    return chrom_exp_error;
}

Double_t fitf(Double_t *v, Double_t *par)
{
  Double_t arg = 0;
  arg = v[0];

  Double_t K_1 = par[0]* xi_1/par[1];
  Double_t K_2 = pow(par[0],2)* par[4]/par[1];
  Double_t T = 2*TMath::Pi()*par[1]*(arg +par[3]);
  Double_t a = K_1 * sin(T) / 2;
  Double_t A = K_2/4*(2*T+sin(2*T));
  Double_t B = K_2/4*(2*T-sin(2*T));
  Double_t C = K_2*sin(T)*sin(T)/2;
  Double_t lambda = B - 4*A*C*C/(1+4*A*A);
```

```
  Double_t f = 0.5 * (1 + 4*C*C/(1+4*A*A));
  Double_t h = 2*C/(1+4*A*A);
  Double_t gamma = tan(T/2)-4*A*C/(1+4*A*A);

  Double_t z = par[2]*par[2]*z_part;
  Double_t A_N = 1/(sqrt(1 + (4*TMath::Pi()*par[6] *(arg + par[3])) * (4*TMath::Pi
      ()*par[6] * (arg + par[3])))*
      (1 + (4 * TMath::Pi() * par[5] * (arg + par[3])) * (4*TMath::Pi()*par[5] * (
          arg + par[3])))) *
      exp(-1/2 * z*z * (4*TMath::Pi()*par[5] * (arg + par[3])) * (4*TMath::Pi()*
          par[5] * (arg + par[3])) /
      (1 + (4*TMath::Pi()*par[5] * (arg + par[3])) * (4*TMath::Pi()*par[5] * (arg
          + par[3]))));

  Double_t M_N = (K_1*K_1 *(1-cos(T)))/(1+K_2*K_2* (T+sin(T))*(T+sin(T)));
  Double_t H_N = pow(1+2*K_2*K_2*(T*T+sin(T)*sin(T)) + pow(K_2,4) *
      pow((T*T - sin(T)*sin(T)),2),-1./4);

  Double_t fitval = par[2] * A_N * H_N * exp(-M_N);

  return fitval;
}

Double_t fitf_global(Double_t *v, Double_t *par)
{
  Double_t arg = 0;
  arg = v[0];

  Double_t K_1 = par[0]* xi_1/par[1];
  Double_t K_2 = pow(par[0],2)* par[4]/par[1];
  Double_t T = 2*TMath::Pi()*par[1]*(arg +par[3]);
  Double_t a = K_1 * sin(T) / 2;
  Double_t A = K_2/4*(2*T+sin(2*T));
  Double_t B = K_2/4*(2*T-sin(2*T));
  Double_t C = K_2*sin(T)*sin(T)/2;
  Double_t lambda = B - 4*A*C*C/(1+4*A*A);
  Double_t f = 0.5 * (1 + 4*C*C/(1+4*A*A));
  Double_t h = 2*C/(1+4*A*A);
  Double_t gamma = tan(T/2)-4*A*C/(1+4*A*A);

  Double_t z = par[2]*par[2]*z_part;
  Double_t A_N = 1/(sqrt(1 + (4*TMath::Pi()*par[6] *(arg + par[3])) * (4*TMath::Pi
      ()*par[6] * (arg + par[3])))*
      (1 + (4 * TMath::Pi() * par[5] * (arg + par[3])) * (4*TMath::Pi()*par[5] * (
          arg + par[3])))) *
      exp(-1/2 * z*z * (4*TMath::Pi()*par[5] * (arg + par[3])) * (4*TMath::Pi()*
          par[5] * (arg + par[3])) /
      (1 + (4*TMath::Pi()*par[5] * (arg + par[3])) * (4*TMath::Pi()*par[5] * (arg
          + par[3]))));

  Double_t M_N = (K_1*K_1 *(1-cos(T)))/(1+K_2*K_2* (T+sin(T))*(T+sin(T)));
  Double_t H_N = pow(1+2*K_2*K_2*(T*T+sin(T)*sin(T)) + pow(K_2,4) *
      pow((T*T - sin(T)*sin(T)),2),-1./4);
  Double_t P_N = 0.5*atan(2*A) + 0.5*atan(lambda/f) - a*a*
      ((lambda*gamma*gamma - lambda*h*h + 2*f*gamma*h)/(f*f+lambda*lambda)+4*A
```

```
          /(1+4*A*A));
155   Double_t Q_N = z*z/2 * (4*TMath::Pi()*mu_u * (arg +par[3]))/(1+(4*TMath::Pi()*
          mu_u * (arg +par[3]))*(4*TMath::Pi()*mu_u * (arg +par[3]))) *
156        atan((8*TMath::Pi()*mu_u * (arg +par[3]))/(1-(4*TMath::Pi()*mu_u * (arg +par
              [3]))*(4*TMath::Pi()*mu_u * (arg +par[3])))) +
157        atan(4*TMath::Pi()*mu_yx * (arg +par[3]));
158
159   Double_t fitval = par[2] * A_N * H_N * exp(-M_N) * sin( par[7] * (P_N + Q_N + 2*
          TMath::Pi()*nu_x_0*(arg +par[3])) );;
160
161   return fitval;
162
163 }
164
165 Double_t tune_scaling(double tune)
166 {
167   double t = 0.;
168
169   t = tune - (int)tune;
170   if (t > 0.5)
171     t = t - 1;
172
173   return t;
174 }
175
176 void graph_compile()
177 {
178   sprintf(fdsname, "./Final_data_sets/Data.dat");
179
180   // Reading the data file
181   ifstream fdsf_read;
182   fdsf_read.open(fdsname, std::ifstream::in);
183
184   gr3 = new TGraphAsymmErrors();
185   gr5 = new TGraphAsymmErrors();
186   gr7 = new TGraphAsymmErrors();
187
188   gr3s = new TGraphAsymmErrors();
189   gr5s = new TGraphAsymmErrors();
190   gr7s = new TGraphAsymmErrors();
191
192   char name[250]={0};
193   int p = 0;
194   TH1F *en_spread = new TH1F("en_spread", "Energy spread distribution"
          ,1000,0.6,1.3);
195   double m,cou = 0;
196   while (fdsf_read.good())
197   {
198     fdsf_read >> axis >> name >> chromaticity >> (double)I_b >> PIX >> mean >>
            mean_sigma_e_error >> ds_counter >> mean_Chisquare >> mean_nu_s >>
            mean_nu_s_error >> mean_Amplitude >> mean_Amplitude_error >> mean_Shift >>
            mean_Shift_error >> mean_xi_2 >> mean_xi_2_error >> mean_mu_x >>
            mean_mu_x_error >> mean_mu_yx >> mean_mu_yx_error;
199     if (axis == 0)
200     {
```

```
201       if (chromaticity == 3)
202       {
203         gr3->SetMarkerColor(1);
204         gr3->SetMarkerStyle(22);
205         gr3->SetPoint(p,I_b, mean);
206         gr3->SetPointError(p,0,I_b_error,mean_sigma_e_error,mean_sigma_e_error);
207         if (p == 0)
208           gr3->Draw("AP");
209         else
210         {
211           gr3->GetXaxis()->SetRangeUser(0,5);
212           gr3->GetYaxis()->SetRangeUser(0.6,1.2);
213           gr3->Draw("0P");
214         }
215       }
216       if (chromaticity == 5)
217       {
218         gr5->SetMarkerColor(2);
219         gr5->SetMarkerStyle(23);
220         gr5->SetPoint(p,I_b, mean);
221         gr5->SetPointError(p,0,I_b_error,mean_sigma_e_error,mean_sigma_e_error);
222         if (p == 0)
223           gr5->Draw("AP");
224         else
225         {
226           gr5->GetXaxis()->SetRangeUser(0,5);
227           gr5->GetYaxis()->SetRangeUser(0.6,1.2);
228           gr5->Draw("0P");
229         }
230       }
231       if (chromaticity == 7)
232       {
233         gr7->SetMarkerColor(3);
234         gr7->SetMarkerStyle(20);
235         gr7->SetPoint(p,I_b, mean);
236         gr7->SetPointError(p,0,I_b_error,mean_sigma_e_error,mean_sigma_e_error);
237         if (p == 0)
238           gr7->Draw("AP");
239         else
240         {
241           gr7->GetXaxis()->SetRangeUser(0,5);
242           gr7->GetYaxis()->SetRangeUser(0.6,1.2);
243           gr7->Draw("0P");
244         }
245       }
246     }
247
248     if (axis == 1)
249     {
250       if (chromaticity == 3)
251       {
252         gr3s->SetMarkerColor(1);
253         gr3s->SetMarkerStyle(26);
254         gr3s->SetPoint(p,I_b, mean);
255         gr3s->SetPointError(p,0,I_b_error,mean_sigma_e_error,mean_sigma_e_error);
```

```
256         if (p == 0)
257           gr3s->Draw("AP");
258         else
259         {
260           gr3s->GetXaxis()->SetRangeUser(0,5);
261           gr3s->GetYaxis()->SetRangeUser(0.6,1.2);
262           gr3s->Draw("0P");
263         }
264       }
265       if (chromaticity == 5)
266       {
267         gr5s->SetMarkerColor(2);
268         gr5s->SetMarkerStyle(32);
269         gr5s->SetPoint(p,I_b, mean);
270         gr5s->SetPointError(p,0,I_b_error,mean_sigma_e_error,mean_sigma_e_error);
271         if (p == 0)
272           gr5s->Draw("AP");
273         else
274         {
275           gr5s->GetXaxis()->SetRangeUser(0,5);
276           gr5s->GetYaxis()->SetRangeUser(0.6,1.2);
277           gr5s->Draw("0P");
278         }
279       }
280       if (chromaticity == 7)
281       {
282         gr7s->SetMarkerColor(3);
283         gr7s->SetMarkerStyle(24);
284         gr7s->SetPoint(p,I_b, mean);
285         gr7s->SetPointError(p,0,I_b_error,mean_sigma_e_error,mean_sigma_e_error);
286         if (p == 0)
287           gr7s->Draw("AP");
288         else
289         {
290           gr7s->GetXaxis()->SetRangeUser(0,5);
291           gr7s->GetYaxis()->SetRangeUser(0.6,1.2);
292           gr7s->Draw("0P");
293         }
294       }
295     }
296
297     if (I_b == 5)
298     {
299       en_spread->Fill(mean);
300       m += ((mean-1.4) * (mean-1.4));
301       cou++;
302     }
303
304     p++;
305   }
306
307   cout << "RMS of current 5 " << sqrt(m/(cou-1)) << endl;
308   en_spread->Draw();
309
310   fdsf_read.close();
```

```
311 }
312
313 Double_t data_compile()
314 {
315
316   double sigma_e = 0.;
317   double sigma_e_error = 0.;
318   double Chisquare = 0.;
319   double nu_s = 0.;
320   double nu_s_error = 0.;
321   double Amplitude = 0.;
322   double Amplitude_error = 0.;
323   double Shift = 0.;
324   double Shift_error = 0.;
325   double mu_x_error = 0.;
326   double mu_yx_error = 0.;
327
328   for (int hour = 12; hour <= 13; hour++)
329     for (int min = 0; min < 60; min++)
330       for (int sec = 0; sec < 60; sec++)
331         for (int hv = 0; hv <= 1; ++hv)
332         {
333           mean = 0.;
334           mean_sigma_e_error = 0.;
335           mean_Chisquare = 0.;
336           mean_nu_s = 0.;
337           mean_nu_s_error = 0.;
338           mean_Amplitude = 0.;
339           mean_Amplitude_error = 0.;
340           mean_Shift = 0.;
341           mean_Shift_error = 0.;
342           mean_xi_2 = 0.;
343           mean_xi_2_error = 0.;
344           mean_mu_x = 0.;
345           mean_mu_x_error = 0.;
346           mean_mu_yx = 0.;
347           mean_mu_yx_error = 0.;
348
349           sprintf(datasetname, "./Data_sets/22Jul%02d:%02d:%02d_%1d.dat", hour,
                  min, sec, hv);
350
351           ifstream dsf_read;
352           char name[250] ={0};
353           dsf_read.open(datasetname, std::ifstream::in);
354           if (!dsf_read.is_open())
355             continue;
356
357           cout << datasetname << endl;
358           ds_counter = 0;
359
360           // cout << "Before reading ... check\n";
361           for (int i = 0; i < num_files; ++i)
362           {
363             dsf_read >> name >> sigma_e >> sigma_e_error >> Chisquare >> nu_s >>
                  nu_s_error >> Amplitude >> Amplitude_error >> Shift >> Shift_error
```

```
            >> xi_2 >> xi_2_error >> mu_x >> mu_x_error >> mu_yx >> mu_yx_error
            ;

        if (fabs(sigma_e) > 0.0001)
        {
          // cout << name << "\t" << sigma_e << endl;
          mean += fabs(sigma_e);
          mean_sigma_e_error += fabs(sigma_e_error);
          mean_Chisquare += Chisquare;
          mean_nu_s += nu_s;
          mean_nu_s_error += nu_s_error;
          mean_Amplitude += Amplitude;
          mean_Amplitude_error += Amplitude_error;
          mean_Shift += Shift;
          mean_Shift_error += Shift_error;
          mean_xi_2 += xi_2;
          mean_xi_2_error += xi_2_error;
          mean_mu_x += mu_x;
          mean_mu_x_error += mu_x_error;
          mean_mu_yx += mu_yx;
          mean_mu_yx_error += mu_yx_error;

          ds_counter++;
        }
      }

      dsf_read.close();

      mean /= ds_counter;
      mean_sigma_e_error /= ds_counter;
      mean_Chisquare /= ds_counter;
      mean_nu_s /= ds_counter;
      mean_nu_s_error /= ds_counter;
      mean_Amplitude /= ds_counter;
      mean_Amplitude_error /= ds_counter;
      mean_Shift /= ds_counter;
      mean_Shift_error /= ds_counter;
      mean_xi_2 /= ds_counter;
      mean_xi_2_error /= ds_counter;
      mean_mu_x /= ds_counter;
      mean_mu_x_error /= ds_counter;
      mean_mu_yx /= ds_counter;
      mean_mu_yx_error /= ds_counter;

      ofstream fdsf;

      sprintf(fdsname, "./Final_data_sets/Data.dat");

      cout << scientific << fixed;

      xi_1 = chrom_val(hv,1,chromaticity,false);
      cout << "Expected value of 1st order chromaticity " << xi_1 << endl;
      xi_1_error = chrom_val(hv,1,chromaticity,true);
      cout << "Expected error value of 1st order chromaticity " << xi_1_error
         << endl;
```

```
416
417             xi_2 = chrom_val(hv,2,chromaticity,false);
418             cout << "Expected value of 2nd order chromaticity " << xi_2 << endl;
419             xi_2_error = chrom_val(hv,2,chromaticity,true);
420             cout << "Expected error value of 2nd order chromaticity " << xi_2_error
                    << endl;
421
422             fdsf.open(fdsname, std::ifstream::app);
423             // Mutiplying with 1000 to express in promils
424             mean *= 1000;
425             mean_sigma_e_error *= 1000;
426             mean_sigma_e_error = sqrt(mean_sigma_e_error*mean_sigma_e_error + mean*
                    xi_1_error/xi_1*mean*xi_1_error/xi_1);
427             cout << "Before writing ... check\n";
428             // cout << scientific;
429             fdsf << hv << "\t" << datasetname << "\t" << chromaticity << "\t" << (
                    double)I_b << "\t" << PIX << "\t" << mean << "\t" <<
                    mean_sigma_e_error << "\t" << ds_counter << "\t" << mean_Chisquare <<
                     "\t" << mean_nu_s << "\t" << mean_nu_s_error << "\t" <<
                    mean_Amplitude << "\t" << mean_Amplitude_error << "\t" << mean_Shift
                    << "\t" << mean_Shift_error << "\t" << mean_xi_2 << "\t" <<
                    mean_xi_2_error << "\t" << mean_mu_x << "\t" << mean_mu_x_error << "\
                    t" << mean_mu_yx << "\t" << mean_mu_yx_error << endl;
430             cout << "After writing ... check\n";
431
432             fdsf.close();
433         }
434
435   return 0;
436 }
437 main()
438 {
439   cout << "\n*****************************************\n\n";
440   if (axis == 0)
441     cout << "Analysing horizontal axis!\n";
442   else
443     cout << "Analysing vertical axis!\n";
444
445   cout << "\n*****************************************\n\n";
446
447   cout << "Chromaticity is equal " << chromaticity << endl;
448
449   cout << "\n*****************************************\n\n";
450
451
452   cout << scientific << fixed;
453
454   xi_1 = chrom_val(axis,1,chromaticity,false);
455   cout << "Expected value of 1st order chromaticity " << xi_1 << endl;
456   xi_1_error = chrom_val(axis,1,chromaticity,true);
457   cout << "Expected error value of 1st order chromaticity " << xi_1_error << endl;
458
459   xi_2 = chrom_val(axis,2,chromaticity,false);
460   cout << "Expected value of 2nd order chromaticity " << xi_2 << endl;
461   xi_2_error = chrom_val(axis,2,chromaticity,true);
```

```
462   cout << "Expected error value of 2nd order chromaticity " << xi_2_error << endl;
463
464   char line[1024] = {0};
465   char str[1024] = {0};
466   char fname[260] = {0};
467   char figname[260] = {0};
468   char termfigname[260] = {0};
469   char datasetname0[260] = {0};
470   char termdatasetname[260] = {0};
471
472
473   double buf = 0.;
474   double x[N_all] = {0.,};
475   double data[N_all] = {0.,};
476   double data_errors[N_all] = {0.,};
477
478   double bet_x[73] = {0,};
479   double bet_y[73] = {0,};
480   double bpm_ph_adv_x [73] = {0,};
481   double bpm_ph_adv_y[73] = {0,};
482
483   double fig_hour[num_files] = {0.,};
484   double fig_min[num_files] = {0.,};
485   double fig_sec[num_files] = {0.,};
486
487   cout << "I_b value is " << I_b << endl;
488   cout << "I_b error value is " << I_b_error << endl;
489
490   cout << "Reading beta_x and beta_y..." << endl;
491
492   ifstream g;
493   g.open("./twiss.model_cleaned.dat");
494
495   // 1387 is the number of elements in the file
496   for (int i = 0; i < 1387; ++i)
497   {
498     g >> buf;
499     // Reading beta X
500     if (i%19 == 1)
501       bet_x[i/19] = buf;
502
503     // Reading beta Y
504     if (i%19 == 2)
505       bet_y[i/19] = buf;
506
507     // Reading phase advance of BPMs X
508     if (i%19 == 9)
509       bpm_ph_adv_x[i/19] = buf;
510
511     // Reading phase advance of BPMs Y
512     if (i%19 == 10)
513       bpm_ph_adv_y[i/19] = buf;
514   }
515   g.close();
516
```

```
517
518   cout << "Started reading..." << endl;
519
520   ifstream file[num_files]; // Number of files in one measurement
521   for (int hour = 12; hour <= 12; hour++)
522     for (int min = 47; min < 48; min++)
523     {
524       int file_num = 0;
525       for (int sec = 0; sec < 60; sec++)
526       {
527         sprintf(fname, "./bpmdata_cleaned/22Jul%02d:%02d:%02d_cleaned.dat", hour,
              min, sec);
528
529         // File reading using buffer. Checking file existence
530         ifstream f_buffer;
531         f_buffer.open(fname);
532         if(!f_buffer.is_open())
533           continue;
534         // file exists process it
535
536         f_buffer.close();
537
538         fig_hour[file_num] = hour;
539         fig_min[file_num] = min;
540         fig_sec[file_num] = sec;
541
542         file[file_num].open(fname);
543         ++file_num;
544       }
545
546       // If nothing with given time name is read continue to next minute
547       if(file_num == 0 && file_num != 9) continue;
548
549       // Creating Data Set File for afterfit data
550       ofstream dsf;
551
552       // cout << bet_x[0] ` bet_x[1] << " !@#" << endl;
553       // File evaluation over num_files number files
554       for (int file_i = 0; file_i < num_files; ++file_i)
555       {
556
557         cout << "Reading file number " << file_i << endl;
558
559         if (axis == 0)
560         {
561           WINDOW = 3;
562           // Line number in the file
563           for (int line_num = 0; line_num < 73; ++line_num)
564           {
565             // cout << "Processing line " << line_num << endl;
566
567             double sum = 0.;
568             mean  = 0.;
569             for (int i = 0; i < N; ++i)
570             {
```

```
571               file[file_i] >> buf;
572               x[line_num+(i%N)*73] = i+tune_scaling(bpm_ph_adv_x[line_num])/
                      tune_scaling(nu_x_0);
573               // Converting BPM data from mm to m
574               data[line_num+(i%N)*73] = buf / 1000 /sqrt(bet_x[line_num]);
575               // Calculation of error bar
576               data_errors[line_num+(i%N)*73] = sqrt(1/bet_x[line_num] * ( (20e-6)
                      *(20e-6) +
577                 (data[line_num+(i%N)*73]*0.01)*(data[line_num+(i%N)*73]*0.01) +
578                 (data[line_num+(i%N)*73]/2 * 0.05)*(data[line_num+(i%N)*73]/2 *
                        0.05) ) );
579               if (i >= begin_cut)
580                 sum += data[line_num+(i%N)*73];
581             }
582             mean = sum / (N - begin_cut);
583
584             for (int j = 0; j < N; ++j)
585               data[line_num+(i%N)*73] -= mean;
586
587           }
588           // End of X axis analysis
589         }
590
591         if (axis == 1)
592         {
593           WINDOW = 7;
594
595           cout << "Skipping the horizontal section of the file...\n";
596           // Setting pointer to the 73th line
597           for (int fc = 0; fc < 73000; ++fc)
598             file[file_i] >> buf;
599
600
601
602           // Line number in the file
603           for (int line_num = 0; line_num < 73; ++line_num)
604           {
605             // cout << "Processing line " << line_num << endl;
606
607             double sum = 0.;
608             double mean  = 0.;
609             for (int i = 0; i < N; ++i)
610             {
611
612               file[file_i] >> buf;
613               x[line_num+(i%N)*73] = i+tune_scaling(bpm_ph_adv_y[line_num])/
                      tune_scaling(nu_y_0);
614               // Converting BPM data from mm to m
615               data[line_num+(i%N)*73] = buf / 1000 /sqrt(bet_y[line_num]);
616               // Calculation of error bar
617               data_errors[line_num+(i%N)*73] = sqrt(1/bet_y[line_num] * ( (20e-6)
                      *(20e-6) +
618                 (data[line_num+(i%N)*73]*0.01)*(data[line_num+(i%N)*73]*0.01) +
619                 (data[line_num+(i%N)*73]/2 * 0.05)*(data[line_num+(i%N)*73]/2 *
                        0.05) ) );
```

```
620             if (i >= begin_cut)
621               sum += data[line_num+(i%N)*73];
622           }
623           mean = sum / (N - begin_cut);
624
625           for (int j = 0; j < N; ++j)
626             data[line_num+(i%N)*73] -= mean;
627         }
628         // End of Y axis analysis
629       }
630
631       // Sorting the x axis for fitting procedure to be easier
632       int idx[N_all];
633       TMath::Sort(N_all,x,idx,false);
634       double x_sorted[N_all] = {0.,};
635       for (int i = 0; i < N_all; ++i)
636         x_sorted[i] = x[idx[i]];
637
638       double amp[N_all] = {0.,};
639       double amp_x[N_all] = {0.,};
640       double freq[N_all] = {0.,};
641       double phase[N_all] = {0.,};
642       double temp[N_all] = {0.,};
643
644       // Fitting betatron oscillations with sine function
645       TF1 *sinus = new TF1("sinus", "[0]*sin([1]*x+[2])");
646
647       sinus->SetParName(0,"Amplitude");
648       sinus->SetParName(1,"Frequency");
649       sinus->SetParName(2,"Phase");
650
651       TGraph *gr1 = new TGraphErrors(N_all, x, data,0,data_errors);
652       gr1->SetMarkerColor(kBlue);
653       // gr1->GetYaxis()->SetRangeUser(-0.2,0.2);
654       gr1->GetXaxis()->SetRangeUser(0,1005);
655       // gr1->Draw("A*");
656       double amp_errors[N_all] = {0.,};
657       double x_min = TMath::MinElement(N_all,x_sorted);
658       double x_max = TMath::MaxElement(N_all,x_sorted);
659       // cout << "Loc Maximum" << x_max << endl;
660       for (int i = /*400*/ WINDOW/2+begin_cut+x_min; i < /*300*/ x_max-WINDOW/2;
              ++i)
661       {
662         // sinus->SetParameter(0,240);
663         sinus->SetParameters(100,3,-5.6);
664
665         gr1->Fit("sinus","RNQ","",i - WINDOW/2,i + WINDOW/2);
666         gr1->SetTitle("");
667         amp[i] = fabs(sinus->GetParameter("Amplitude"));
668         amp_x[i] = x[i*73];
669         freq[i] = sinus->GetParameter("Frequency");
670         phase[i] = sinus->GetParameter("Phase");
671         amp_errors[i] = sinus->GetParError(0); // Getting Amplitude errors
672       }
673       gStyle->SetOptFit(1111);
```

```
674
675
676         gr1 = new TGraphErrors(N_all, amp_x, amp,0,amp_errors);
677         gr1->SetMarkerColor(kGreen);
678         gr1->Draw("A*");
679         if (PIX == 300)
680           gr1->GetYaxis()->SetRangeUser(0,0.00007);
681         if (PIX == 700)
682           gr1->GetYaxis()->SetRangeUser(0,0.0012);
683
684         // Uncomment for adjusting sine parameters
685         // sinus->Draw("SAME");
686
687         if (axis == 0)
688         {
689           z_part = 1/eps_x;
690           mu_u = mu_x;
691           mu_vu = mu_xy;
692         }
693         else
694         {
695           z_part = 1/eps_y;
696           mu_u = mu_y;
697           mu_vu = mu_yx;
698         }
699
700         // If necessary change the range of ffit function
701         TF1 *exp_fit = new TF1("exp_fit", fitf,/*400*/ WINDOW/2+begin_cut,/*300*/
              N-WINDOW/2-1,7);
702
703         if(axis == 0)
704           exp_fit->SetParNames("sigma_e","nu_s","Amplitude","Shift","xi_2","mu_x",
                "mu_xy");
705         else
706           exp_fit->SetParNames("sigma_e","nu_s","Amplitude","Shift","xi_2","mu_y",
                "mu_yx");
707
708         cout << scientific;
709         // Passing beta x and y for given bpm.
710         cout << xi_2 << "\t" << mu_u << "\t" << mu_vu << endl;
711         if (axis == 0)
712         {
713           if (PIX == 100)
714             exp_fit->SetParameters(0.0011,0.0053,0.00014,-5,xi_2,mu_u,mu_vu);
715           if (PIX == 300)
716             exp_fit->SetParameters(0.0011,0.0053,0.0004,-5,xi_2,mu_u,mu_vu);
717           if (PIX == 700)
718             exp_fit->SetParameters(0.0011,0.0053,0.0008,-5,xi_2,mu_u,mu_vu);
719         }
720         else
721         {
722           exp_fit->SetParameters(0.0011,0.0053,0.000014,-5,xi_2,mu_u,mu_vu);
723           if (axis == 1 && hour == 13 && min == 31)
724             exp_fit->SetParameters(0.0011,0.0045,0.000014,-5,xi_2,mu_u,mu_vu);
725           if (PIX == 300)
```

```
            exp_fit->SetParameters(0.001,0.005,0.000055,-5,xi_2,mu_u,mu_vu);
        }

        sprintf(figname, "./Fit_figs/22Jul%02d:%02d:%02d_%1d.png", fig_hour[file_i
            ], fig_min[file_i], fig_sec[file_i], axis);

        gStyle->SetOptFit(1111);
        // If necessary change the range of fit
        gr1->Fit("exp_fit","R","",/*400*/ WINDOW/2+begin_cut,/*300*/ N-WINDOW/2-1)
            ;
        double sigma_e = exp_fit->GetParameter("sigma_e");
        double sigma_e_error = exp_fit->GetParError(0);
        double Chisquare = exp_fit->GetChisquare();
        double ndf = exp_fit->GetNDF();
        double nu_s = exp_fit->GetParameter("nu_s");
        double nu_s_error = exp_fit->GetParError(1);
        double Amplitude = exp_fit->GetParameter("Amplitude");
        double Amplitude_error = exp_fit->GetParError(2);
        double Shift = exp_fit->GetParameter("Shift");
        double Shift_error = exp_fit->GetParError(3);
        double xi_2 = exp_fit->GetParameter("xi_2");
        double xi_2_error = exp_fit->GetParError(4);
        if (axis == 0)
        {
          double mu_x = exp_fit->GetParameter("mu_x");
          double mu_x_error = exp_fit->GetParError(5);
          double mu_yx = exp_fit->GetParameter("mu_xy");
          double mu_yx_error = exp_fit->GetParError(6);
        }
        else
        {
          double mu_x = exp_fit->GetParameter("mu_y");
          double mu_x_error = exp_fit->GetParError(5);
          double mu_yx = exp_fit->GetParameter("mu_yx");
          double mu_yx_error = exp_fit->GetParError(6);
        }

        if (axis == 0)
        {
          // gr1->GetYaxis()->SetRangeUser(0,0.00006);
          gr1->GetYaxis()->SetRangeUser(0,0.00014);
          if (PIX == 300)
            gr1->GetYaxis()->SetRangeUser(0,0.0005);
          if (PIX == 700)
            gr1->GetYaxis()->SetRangeUser(0,0.0012);
        }
        else
        {
          gr1->GetYaxis()->SetRangeUser(0,0.000025);

          if (PIX == 300)
            gr1->GetYaxis()->SetRangeUser(0,0.00007);
        }
        // For manual fitting draw the function
        // exp_fit->Draw("SAME");
```

```
779         c1->SaveAs(figname);
780
781         if (file_i == 0)
782           sprintf(datasetname0, "./Data_sets/22Jul%02d:%02d:%02d_%1d.dat",
                fig_hour[0], fig_min[0], fig_sec[0], axis);
783         else
784           sprintf(datasetname, "./Data_sets/22Jul%02d:%02d:%02d_%1d.dat", fig_hour
                [file_i], fig_min[file_i], fig_sec[file_i], axis);
785
786
787         sprintf(termdatasetname, "gnome-open ./Data_sets/22Jul%02d:%02d:%02d_%1d.
              dat", fig_hour[0], fig_min[0], fig_sec[0], axis);
788         sprintf(termfigname, "gnome-open ./Fit_figs/22Jul%02d:%02d:%02d_%1d.png",
              fig_hour[file_i], fig_min[file_i], fig_sec[file_i], axis);
789
790         if (file_i == 0)
791           dsf.open(datasetname0, std::ifstream::out);
792
793         if (file_i == 0)
794           dsf << datasetname0 << "\t" << sigma_e << "\t" << sigma_e_error << "\t"
                << Chisquare/ndf << "\t" << nu_s << "\t" << nu_s_error << "\t" <<
                Amplitude << "\t" << Amplitude_error << "\t" << Shift << "\t" <<
                Shift_error << "\t" << xi_2 << "\t" << xi_2_error << "\t" << mu_x <<
                "\t" << mu_x_error << "\t" << mu_yx << "\t" << mu_yx_error << "\n";
795         else
796           dsf << datasetname << "\t" << sigma_e << "\t" << sigma_e_error << "\t"
                << Chisquare/ndf << "\t" << nu_s << "\t" << nu_s_error << "\t" <<
                Amplitude << "\t" << Amplitude_error << "\t" << Shift << "\t" <<
                Shift_error << "\t" << xi_2 << "\t" << xi_2_error << "\t" << mu_x <<
                "\t" << mu_x_error << "\t" << mu_yx << "\t" << mu_yx_error << "\n";
797
798
799
800         // Opening the image and the dataset files for data quality check
801         if (file_i == num_files - 1)
802         {
803           system(termfigname);
804           system(termdatasetname);
805         }
806
807         if (file_i == num_files - 1)
808         {
809           // Closing dataset file
810           dsf.close();
811
812           char approval;
813           cout << "Be sure to check the data quality!\n Are you ready to proceed?\
                n";
814           cin >> approval;
815         }
816         else
817           continue;
818
819         ifstream dsf_read;
820         char name[250] ={0};
```

```
821         dsf_read.open(datasetname0, std::ifstream::in);
822         ds_counter = 0;
823         mean = 0;
824         mean_sigma_e_error = 0;
825         cout << "Before reading ... check\n";
826         for (int i = 0; i < num_files; ++i)
827         {
828           dsf_read >> name >> sigma_e >> sigma_e_error >> Chisquare >> nu_s >>
                nu_s_error >> Amplitude >> Amplitude_error >> Shift >> Shift_error >>
                 xi_2 >> xi_2_error >> mu_x >> mu_x_error >> mu_yx >> mu_yx_error;
829
830           // Checking if the line should be analysed
831           if (fabs(sigma_e) > 0.0001)
832           {
833             mean += sigma_e;
834             mean_sigma_e_error += sigma_e_error;
835             mean_Chisquare += Chisquare;
836             mean_nu_s += nu_s;
837             mean_nu_s_error += nu_s_error;
838             mean_Amplitude += Amplitude;
839             mean_Amplitude_error += Amplitude_error;
840             mean_Shift += Shift;
841             mean_Shift_error += Shift_error;
842             mean_xi_2 += xi_2;
843             mean_xi_2_error += xi_2_error;
844             mean_mu_x += mu_x;
845             mean_mu_x_error += mu_x_error;
846             mean_mu_yx += mu_yx;
847             mean_mu_yx_error += mu_yx_error;
848
849             ds_counter++;
850           }
851         }
852         // Calculating the mean for the parameters
853         mean /= ds_counter;
854         mean_sigma_e_error /= ds_counter;
855         mean_Chisquare /= ds_counter;
856         mean_nu_s /= ds_counter;
857         mean_nu_s_error /= ds_counter;
858         mean_Amplitude /= ds_counter;
859         mean_Amplitude_error /= ds_counter;
860         mean_Shift /= ds_counter;
861         mean_Shift_error /= ds_counter;
862         mean_xi_2 /= ds_counter;
863         mean_xi_2_error /= ds_counter;
864         mean_mu_x /= ds_counter;
865         mean_mu_x_error /= ds_counter;
866         mean_mu_yx /= ds_counter;
867         mean_mu_yx_error /= ds_counter;
868
869         // Writing final data to the file
870         ofstream fdsf;
871
872         sprintf(fdsname, "./Final_data_sets/Data.dat");
873         sprintf(figname, "./Fit_figs/22Jul%02d:%02d:%02d_%1d.png", fig_hour[0],
```

```
            fig_min[0], fig_sec[0], axis);
874
875         fdsf.open(fdsname, std::ifstream::app);
876         // Mutiplying with 1000 to express in promils
877         mean *= 1000;
878         mean_sigma_e_error *= 1000;
879         mean_sigma_e_error = sqrt(mean_sigma_e_error*mean_sigma_e_error + mean*
                xi_1_error/xi_1*mean*xi_1_error/xi_1);
880         cout << "Before writing ... check\n";
881         fdsf << axis << "\t" << figname << "\t" << chromaticity << "\t" << (double
                )I_b << "\t" << PIX << "\t" << mean << "\t" << mean_sigma_e_error << "\
                t" << ds_counter << "\t" << mean_Chisquare << "\t" << mean_nu_s << "\t"
                 << mean_nu_s_error << "\t" << mean_Amplitude << "\t" <<
                mean_Amplitude_error << "\t" << mean_Shift << "\t" << mean_Shift_error
                << "\t" << mean_xi_2 << "\t" << mean_xi_2_error << "\t" << mean_mu_x <<
                 "\t" << mean_mu_x_error << "\t" << mean_mu_yx << "\t" <<
                mean_mu_yx_error << endl;
882         cout << "After writing ... check\n";
883
884         fdsf.close();
885
886         // Uncomment for the global fit
887
888         // gr1 = new TGraphErrors(N_all, x, data,0,data_errors);
889         // // gr1 = new TGraph(N_all, x_sorted, data);
890         // gr1->SetMarkerColor(kBlue);
891         // gr1->Draw("A*");
892         // gr1->SetTitle("");
893         // gr1->GetYaxis()->SetRangeUser(-0.0003,0.0003);
894         // gr1->GetXaxis()->SetRangeUser(0,1005);
895
896         // if (axis == 0)
897         // {
898         //  z_part = 1/eps_x;
899         //  mu_u = mu_x;
900         //  mu_vu = mu_xy;
901         // }
902         // else
903         // {
904         //  z_part = 1/eps_y;
905         //  mu_u = mu_y;
906         //  mu_vu = mu_yx;
907         // }
908
909
910         // TF1 *exp_fit = new TF1("exp_fit", fitf_global, WINDOW/2+begin_cut,
                800/*N-WINDOW/2-1*/,8);
911
912         // if(axis == 0)
913         //  exp_fit->SetParNames("sigma_e","nu_s","Amplitude","Intercept","xi_2","
                mu_x","mu_xy","Sin_Freq");
914         // else
915         //  exp_fit->SetParNames("sigma_e","nu_s","Amplitude","Intercept","xi_2","
                mu_y","mu_yx","Sin_Freq");
916
```

```
917         // exp_fit->SetParameters(0.0011,0.0053,0.00028,-5,xi_2,mu_u,mu_vu
                ,0.02153);
918
919         // gStyle->SetOptFit(1111);
920         // gr1->Fit("exp_fit","R","",WINDOW/2+begin_cut, 800/*N-WINDOW/2-1*/);
921         // gr1->SetTitle("Fit with error bars");
922         // exp_fit->Draw("SAME");
923
924         dsf_read.close();
925
926       }
927
928       // Closing 10 files
929       for (int i = 0; i < num_files; ++i)
930         file[i].close();
931
932     }
933
934   graph_compile();
935
936
937 return 0;
938 }
```

A.2 Chromaticity analysis code

Listing 2: Chromaticity fit code

```
#include <iostream>
using namespace std;
#include <fstream>
#include "TMath.h"

void chrom(int axis, int chrom_order, double& k, double& b, double& k_error,
    double& b_error)
{
  if (axis == 0)
    cout << "Horizontal data is processed...\n";
  else
    cout << "Vartical data is processed...\n";

  double freq0 = 499636249.1;

  double num1,num2,num3 = 0.;
  char *fname[2] = {"./rf_freq_c_5_5_pc_60_100_f_150_nb_150.dat", "./
      rf_freq_c_9_9_pc_60_100_f_150_nb_150.dat"};

  double freq[19] = {0.,};
  double freq_errors[19] = {0.,};
  double Q[19] = {0.,};
  double Q_errors[19] = {0.,};
  double mean = 0.;

  // Reading first file
  ifstream f;
  f.open(fname[0]);
  if (f.fail())
    return 1;

  for (int i = 0; i < 19; ++i)
  {
    f >> num1 >> num2 >> num3;
    // After this operations it is actually not frequency anymore, but momentum
    freq[i] = (freq0 - num1) / freq0 / 6e-4;
    freq_errors[i] = 10e-8;
    Q_errors[i] = 0.001;
    if (axis == 0)
      Q[i] = num2;
    else
      Q[i] = num3;

    mean += Q[i];
  }
  f.close();

  mean /= 19;

  for (int i = 0; i < 19; ++i)
```

```
    Q[i] -= mean;


  chrom_gr = new TGraphErrors(19, freq, Q, freq_errors, Q_errors);
  chrom_gr->SetMarkerColor(kBlue);
  chrom_gr->GetYaxis()->SetRangeUser(-0.03,0.03);
  chrom_gr->Draw("A*");
  chrom_gr->SetTitle(" ");
  chrom_gr->GetXaxis()->SetTitle("#frac{#Delta p}{p_{0}}");
  chrom_gr->GetYaxis()->SetTitleOffset(1.4);
  if (axis == 0)
    chrom_gr->GetYaxis()->SetTitle("Q_{x5},Q_{x9}");
  else
    chrom_gr->GetYaxis()->SetTitle("Q_{y5},Q_{y9}");

  chrom_gr->Fit("pol2", "F");
  chrom_gr->GetFunction("pol2")->SetLineColor(kBlue);
  double xi_5_1 = chrom_gr->GetFunction("pol2")->GetParameter("p1");
  double xi_5_2 = chrom_gr->GetFunction("pol2")->GetParameter("p2");
  double xi_5_1_error = chrom_gr->GetFunction("pol2")->GetParError(1);
  double xi_5_2_error = chrom_gr->GetFunction("pol2")->GetParError(2);

  gStyle->SetOptFit(0001);

  mean = 0.;
  for (int i = 0; i < 19; ++i)
  {
    freq[i] = 0.;
    Q[i] = 0.;
  }

  // Reading second file
  ifstream f;
  f.open(fname[1]);
  if (f.fail())
    return 1;

  for (int i = 0; i < 10; ++i)
  {
    f >> num1 >> num2 >> num3;
    // After this operations it is actually not frequency anymore, but momentum
    freq[i] = (freq0 - num1) / freq0 / 6e-4;
    if (axis == 0)
      Q[i] = num2;
    else
      Q[i] = num3;

    mean += Q[i];
  }
  f.close();

  mean /= 10;


  for (int i = 0; i < 10; ++i)
```

```
    Q[i] -= mean;


  chrom_gr = new TGraphErrors(10, freq, Q, freq_errors, Q_errors);
  chrom_gr->SetMarkerColor(kRed);
  chrom_gr->Draw("*");

  chrom_gr->Fit("pol2", "F");
  chrom_gr->GetFunction("pol2")->SetLineColor(kRed);
  double xi_9_1 = chrom_gr->GetFunction("pol2")->GetParameter("p1");
  double xi_9_2 = chrom_gr->GetFunction("pol2")->GetParameter("p2");
  double xi_9_1_error = chrom_gr->GetFunction("pol2")->GetParError(1);
  double xi_9_2_error = chrom_gr->GetFunction("pol2")->GetParError(2);

  if (chrom_order == 1)
  {
    // Calculating linear dependence based chromaticity of 5 and 9 first order
    double k = (xi_9_1-xi_5_1)/(9 - 5);
    double b = -(xi_9_1-xi_5_1)/(9 - 5)*5+xi_5_1;
    cout << "First order chromaticity:\nk = " << k << "\nb = " << b << endl;
    // Calculating error bars by interpolation of 5 and 9 error bars
    double k_error = (xi_9_1_error-xi_5_1_error)/(9 - 5);
    double b_error = -(xi_9_1_error-xi_5_1_error)/(9 - 5)*5+xi_5_1_error;
    cout << "First order chromaticity error:\nk error = " << k_error << "\nb error
        = " << b_error << endl;

    double x[2] = {5,9};
    double y[2] = {xi_5_1,xi_9_1};
    double ey[2] = {xi_5_1_error,xi_9_1_error};
    TGraphErrors *lint = new TGraphErrors(2,x,y,0,ey);
    lint->SetFillColor(4);
    lint->SetFillStyle(3006);
    lint->SetTitle("");
    lint->GetXaxis()->SetTitle("#xi_{x}^{theor}");
    lint->GetYaxis()->SetTitle("#xi_{x}^{exp}");
    lint->GetYaxis()->SetTitleOffset(1.4);
    lint->Draw("aL*3");
  }
  else
  {
    // Calculating linear dependence based chromaticity of 5 and 9 second order
    double k = (xi_9_2-xi_5_2)/(9 - 5);
    double b = -(xi_9_2-xi_5_2)/(9 - 5)*5+xi_5_2;
    cout << "Second order chromaticity:\nk = " << k << "\nb = " << b << "\t" <<
       endl;
    // Calculating error bars by interpolation of 5 and 9 error bars
    double k_error = (xi_9_2_error-xi_5_2_error)/(9 - 5);
    double b_error = -(xi_9_2_error-xi_5_2_error)/(9 - 5)*5+xi_5_2_error;
    cout << "Second order chromaticity error:\nk error = " << k_error << "\nb
       error = " << b_error << endl;

    double x[2] = {5,9};
    double y[2] = {xi_5_2,xi_9_2};
    double ey[2] = {xi_5_2_error,xi_9_2_error};
    TGraphErrors *lint = new TGraphErrors(2,x,y,0,ey);
```

```
    lint->SetFillColor(4);
    lint->SetFillStyle(3006);
    lint->SetTitle("");
    lint->GetXaxis()->SetTitle("#xi_{x}^{theor}");
    lint->GetYaxis()->SetTitle("#xi_{y}^{exp}");
    lint->GetYaxis()->SetTitleOffset(1.4);
    lint->Draw("aL*3");
  }
}
```

B Data tables

	Axis	Identifier of the file	ξ	I_b	PIX/PIY	σ_δ	$\Delta\sigma_\delta$	# fits
a	0	./Data_sets/22Jul12:47:00_0.dat	5	0.8	100	0.956435	0.0296502	6
b	0	./Data_sets/22Jul12:49:21_0.dat	5	0.8	300	0.873277	0.0270725	7
c	0	./Data_sets/22Jul12:51:22_0.dat	5	0.8	700	0.961614	0.0298104	3
d	1	./Data_sets/22Jul12:52:30_1.dat	5	0.8	100	0.935943	0.026003	2
e	1	./Data_sets/22Jul12:54:13_1.dat	5	0.8	300	0.765104	0.0210024	4
f	0	./Data_sets/22Jul12:58:31_0.dat	7	0.8	300	0.887056	0.0274995	7
g	0	./Data_sets/22Jul12:59:01_0.dat	7	0.8	700	0.908417	0.0281613	5
h	0	./Data_sets/22Jul13:00:30_0.dat	7	0.8	100	0.885627	0.0274604	9
i	1	./Data_sets/22Jul13:01:10_1.dat	7	0.8	100	0.841562	0.023321	6
j	1	./Data_sets/22Jul13:02:16_1.dat	7	0.8	300	0.864119	0.0237073	6
k	0	./Data_sets/22Jul13:03:20_0.dat	3	0.8	100	1.03368	0.0320841	5
l	0	./Data_sets/22Jul13:04:43_0.dat	3	0.8	300	1.15967	0.0359564	1
m	0	./Data_sets/22Jul13:05:20_0.dat	3	0.8	700	1.00032	0.031011	6
n	1	./Data_sets/22Jul13:06:03_1.dat	3	0.8	100	0.639407	0.0178955	2
o	1	./Data_sets/22Jul13:07:02_1.dat	3	0.8	300	0.614561	0.016854	5
p	0	./Data_sets/22Jul13:15:37_0.dat	5	1.7	100	0.938446	0.0291057	5
q	0	./Data_sets/22Jul13:16:08_0.dat	5	1.7	300	0.950425	0.0294648	4
r	0	./Data_sets/22Jul13:17:04_0.dat	5	1.7	700	0.956435	0.0296502	6
s	1	./Data_sets/22Jul13:18:03_1.dat	5	1.7	100	0.706157	0.0200628	1
t	1	./Data_sets/22Jul13:19:01_1.dat	5	1.7	300	0.675801	0.0185573	4
u	0	./Data_sets/22Jul13:21:03_0.dat	7	1.7	100	0.892101	0.0276663	6
v	0	./Data_sets/22Jul13:22:02_0.dat	7	1.7	300	0.903895	0.0280219	6
w	0	./Data_sets/22Jul13:23:04_0.dat	7	1.7	700	0.940242	0.0291482	4
x	1	./Data_sets/22Jul13:24:02_1.dat	7	1.7	100	0.788141	0.0222452	1
y	1	./Data_sets/22Jul13:26:37_1.dat	7	1.7	300	0.848953	0.023325	8
z	0	./Data_sets/22Jul13:27:13_0.dat	3	1.7	100	1.15221	0.0357584	2
aa	0	./Data_sets/22Jul13:28:01_0.dat	3	1.7	300	1.13485	0.0351848	5
ab	0	./Data_sets/22Jul13:29:11_0.dat	3	1.7	700	1.06679	0.0330722	5
ac	1	./Data_sets/22Jul13:31:12_1.dat	3	1.7	100	0.637489	0.0179001	1
ad	0	./Data_sets/22Jul13:36:41_0.dat	5	5	300	1.25651	0.0389614	7
ae	0	./Data_sets/22Jul13:37:33_0.dat	5	5	700	1.25739	0.0389827	5
af	1	./Data_sets/22Jul13:38:06_1.dat	5	5	100	0.784229	0.0231864	1
ag	1	./Data_sets/22Jul13:39:00_1.dat	5	5	300	0.990736	0.0274605	5
ah	0	./Data_sets/22Jul13:40:24_0.dat	3	5	100	1.42354	0.0446386	9

Table 5: Fit data without less important parameters and their error values.

	$\langle\chi^2\rangle$	ν_s	$\Delta\nu_s$	Amplitude	Δ Amplitude	Shift	Δ Shift	
a	1429.55	0.00525669	3.21711e-07	0.000999186	2.19377e-07	-12.4349	0.0121948	
b	718.947	0.00528392	2.1519e-07	0.000421398	9.00435e-08	-12.869	0.0166352	
c	1772.35	0.00528414	1.79687e-07	0.00108215	1.65243e-07	-10.9262	0.00910284	
d	15.9885	0.00520262	6.44495e-06	1.84685e-05	9.29158e-08	-9.61054	0.300608	
e	158.925	0.00485358	1.71991e-06	5.60429e-05	7.46853e-08	1.24239	0.109374	
f	411.347	0.00517391	1.45394e-07	0.000413596	1.03229e-07	-11.4856	0.0119911	
g	1172.55	0.00515972	1.41111e-07	0.00112995	1.7667e-07	-10.1477	0.00723061	
h	39.9532	0.00517203	4.302e-07	0.000131731	9.97327e-08	-11.0719	0.0361731	
i	10.4344	0.00509558	3.27515e-06	1.71834e-05	7.73928e-08	-5.53207	0.222915	
j	102.093	0.0050644	1.01538e-06	5.84069e-05	8.87164e-08	-4.99154	0.0687652	
k	21.1471	0.00530406	1.13262e-06	0.00013379	9.05901e-08	-16.2706	0.086931	
l	1565.49	0.00526784	3.87387e-07	0.000478547	1.84897e-07	179.396	0.0396218	
m	1108.63	0.00528834	4.08909e-07	0.00104278	1.39832e-07	-13.5309	0.0204192	
n	13.1847	0.00459863	5.79823e-06	1.75138e-05	5.67105e-08	7.51392	0.581217	
o	242.836	0.00454554	1.10464e-06	5.54492e-05	2.72516e-08	10.5074	0.121843	
p	41.3835	0.00536303	9.55837e-07	0.000113153	9.93702e-08	-18.0116	0.0592604	
q	271.883	0.00537651	3.31694e-07	0.000365855	1.07298e-07	-18.3404	0.0194986	
r	1429.55	0.00525669	3.21711e-07	0.000999186	2.19377e-07	-12.4349	0.0121948	
s	18.8454	0.0046643	1.17194e-05	1.44298e-05	9.74665e-08	3.9831	0.61084	
t	95.4474	0.00455973	1.80832e-06	4.72874e-05	9.90046e-08	6.09014	0.134711	
u	48.4323	0.00524643	7.09645e-07	0.000116085	1.17214e-07	-13.815	0.0461741	
v	369.035	0.00528919	2.16124e-07	0.000385506	1.14983e-07	-14.0246	0.0140988	
w	583.262	0.00520331	2.49338e-07	0.00102672	2.36096e-07	-11.8499	0.00954985	
x	19.0695	0.0051018	8.80154e-06	1.47108e-05	1.02189e-07	-6.50597	0.39952	
y	68.7638	0.00502567	2.30704e-06	5.10848e-05	1.11782e-07	-4.39774	0.113684	
z	32.9085	0.00552644	1.99112e-06	0.000126421	1.26151e-07	-21.4627	0.104081	
aa	505.432	0.00553753	6.81951e-07	0.000406936	1.33773e-07	-21.3402	0.0353435	
ab	599.642	0.00540193	6.4521e-07	0.000999316	1.8634e-07	-15.6965	0.0241644	
ac	40.8837	0.0043601	7.18125e-06	1.77131e-05	8.78365e-08	16.5107	0.785391	
ad	320.122	0.00514198	2.37383e-06	0.000322312	5.53372e-07	-26.8715	0.0658478	
ae	477.825	0.00486073	4.28871e-07	0.000989728	5.61955e-07	-18.3557	0.0129649	
af	14.3817	0.00449627	1.73316e-05	1.11999e-05	1.36977e-07	-14.3756	0.64754	
ag	76.4361	0.00446938	5.56064e-06	4.67044e-05	2.56572e-07	-15.2692	0.18166	
ah	69.3646	0.00362665	2.40556e-05	0.00012305	1.51148e-06	-20.4022	0.760659	

Table 6: Second part of fit parameters' table 5.

	ξ_2	$\Delta\xi_2$	μ_u	$\Delta\mu_u$	μ_{uv}	$\Delta\mu_{uv}$
a	-346.945	0.180949	7.78694e-12	8.36392e-08	9.93395e-11	1.77683e-07
b	-225.994	0.211263	-3.7732e-05	8.07484e-08	2.36825e-05	8.34917e-08
c	-368.705	0.0908371	-4.55453e-14	1.51764e-08	-3.96673e-12	8.48835e-08
d	-5.2815	4.83325	-4.00088e-10	9.16254e-06	0.000183798	2.44453e-06
e	456.507	1.88733	-6.44681e-05	2.34273e-07	3.50669e-05	1.07171e-06
f	-219.252	0.151589	-5.93788e-05	4.07246e-08	5.70259e-10	2.68206e-07
g	314.882	0.0995307	4.90438e-12	2.8446e-08	-2.08215e-12	9.64275e-08
h	-143.727	0.631985	-8.21485e-05	1.26238e-07	-4.84942e-11	8.54376e-07
i	366.071	3.54255	-2.78975e-05	2.3553e-06	5.17073e-08	2.75827e-06
j	360.999	1.08113	-5.48627e-09	1.1598e-06	0.000155051	5.35263e-07
k	-142.764	1.27671	-5.52751e-05	3.70306e-07	-3.06516e-10	7.70419e-07
l	151.607	0.152386	-2.72704e-05	8.96146e-08	7.55449e-07	1.81227e-07
m	-228.557	0.0965973	-6.40071e-13	3.23769e-08	1.99546e-12	1.44125e-07
n	-87.3465	37.1483	-1.51437e-05	1.30169e-06	1.4998e-05	2.20291e-06
o	0.175815	2.54697	-6.03388e-05	0.0164045	-6.86537e-05	8.00607e-07
p	-214.386	0.913901	-9.96222e-05	1.83159e-07	3.5027e-11	7.47098e-07
q	-287.142	0.286517	-7.55559e-05	6.36146e-08	2.76771e-10	1.27793e-07
r	-346.945	0.180949	7.78694e-12	8.36392e-08	9.93395e-11	1.77683e-07
s	-616.228	12.0388	2.95785e-12	2.40944e-06	-0.000197427	3.65255e-06
t	305.638	3.91899	-1.00356e-09	5.82938e-07	0.000261206	9.74746e-07
u	-251.694	0.7841	-0.000105199	1.82747e-07	1.50856e-11	9.59485e-07
v	-307.624	0.202109	-7.77722e-05	4.99932e-08	8.78714e-10	1.47527e-07
w	-667.22	0.181785	2.01397e-06	2.00152e-07	1.92696e-09	4.97031e-08
x	728.692	8.36478	1.09597e-10	4.14111e-06	-0.000126415	3.12666e-06
y	628.699	2.01147	2.56514e-11	4.9241e-07	0.000178014	1.11386
z	-279.616	0.931554	-1.42049e-05	1.15751e-06	1.7968e-05	2.00578e-06
aa	-287.267	0.282308	-1.51101e-05	2.37915e-07	-2.31112e-09	9.05529e-07
ab	-301.865	0.161278	7.91974e-15	9.11711e-08	2.5882e-09	1.28185e-07
ac	0.000543478	9.39468	3.09048e-11	1.47456e-06	0.000298728	1.95846e-06
ad	-323.423	0.65018	-4.22332e-05	3.2621e-06	5.76262e-10	1.11176e-06
ae	-518.262	0.293747	-4.37678e-12	1.42058e-07	9.19037e-10	2.17083e-07
af	981.692	13.2822	-1.52931e-10	1.43265e-06	-5.01474e-11	2.05973e-06
ag	-182.689	3.93888	-1.61964e-11	5.7411e-07	-9.5678e-13	8.00855e-07
ah	-391.708	2.92348	0.000109826	1.03283e-05	3.21639e-05	2.71505e-05

Table 7: Last part of the fit parameters.