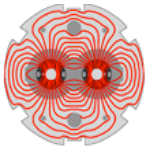
EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH
European Laboratory for Particle Physics



Large Hadron Collider Project

LHC Project Report 1055

Digital Generation of Noise-Signals with Arbitrary Constant or Time-Varying Spectra

(A noise generation software package and its application)

Joachim Tückmantel

Abstract

Artificial creation of arbitrary noise signals is used in accelerator physics to reproduce a measured perturbation spectrum for simulations but also to generate real-time shaped noise spectra for controlled emittance blow-up giving tailored properties to the final bunch shape. It is demonstrated here how one can produce numerically what is, for all practical purposes, an unlimited quantity of *non-periodic* noise data having any predefined spectral density. This spectral density may be constant or varying with time. The noise output never repeats and has excellent statistical properties, important for very long-term applications. It is difficult to obtain such flexibility and spectral cleanliness using analogue techniques. This algorithm was applied both in computer simulations of bunch behaviour in the presence of RF noise in the PS, SPS and LHC and also to generate real-time noise, tracking the synchrotron frequency change during the energy ramp of the SPS and producing controlled longitudinal emittance blow-up. This successful experience indicates that this method can also be applied in the LHC.

CERN AB-RF, joachim.tuckmantel@cern.ch

CERN
CH - 1211 Geneva 23
Switzerland

Geneva, February 2008

1. Introduction

Noise in the components of an accelerator (e.g. magnets, RF) generally can cause emittance blow-up. Computer simulations to study the implications have to faithfully reproduce the measured noise spectrum. In another context artificially introduced noise can be applied to create a desired emittance blow-up. Here, often, a noise spectrum shaped to provide a given function – e.g. triangular or flat within a given frequency range – is desired to get a particular shape of the blown-up bunches. This noise has to be produced first for the computer studies but secondly *identically* for the truly injected artificial noise *in real-time*. A further problem arises if the blow-up should take place while the beam-controlling elements slowly change in a controlled way, for example during acceleration or when the RF voltage changes. To be efficient the noise spectrum has to follow the corresponding change of the synchrotron frequencies in position (shift) and width (scale).

The first digital noise-generation of arbitrary spectral shape was designed by the author to simulate longitudinal emittance blow-up due to time-constant RF noise, especially for long LHC coasts requiring a very long non-periodic noise data string of high statistical quality: required are one value per machine turn, i.e. about 10^9 values for a 24 h coast in LHC. It is an integral part of the simulation program 'NoisySync'[1] including also the generation of monochromatic lines.

In parallel, ways to produce a controlled blow-up of the longitudinal emittance in the SPS and later in the LHC (by a factor 2.5) were looked for. The blow-up in the SPS has to be done during acceleration where the synchrotron frequency changes and so the excitation frequencies have to follow. To roughly realize this for the first tests with minimum effort the noise source of an (outdated) dynamic system analyzer was used to create a flat noise spectrum between 0 Hz and a nominal upper frequency, $\Delta F_{nom,}$ chosen in coarse steps on the instrument and adjusted normally to just cover the synchrotron frequency spread within the bunch. However the spectrum showed long tails extending above ΔF_{nom}. This basic spectrum was mixed in an analogue multiplier with a (monochromatic) carrier at F_0, the nominal output being the same spectrum between F_0 and $f_{up}=F_0+\Delta F_{nom}$ but having also a mirror image between $f_{low}=F_0-\Delta F_{max}$ and F_0. Furthermore the tails show up above f_{up} and below f_{low} as a mirror image. (Fig. 1)

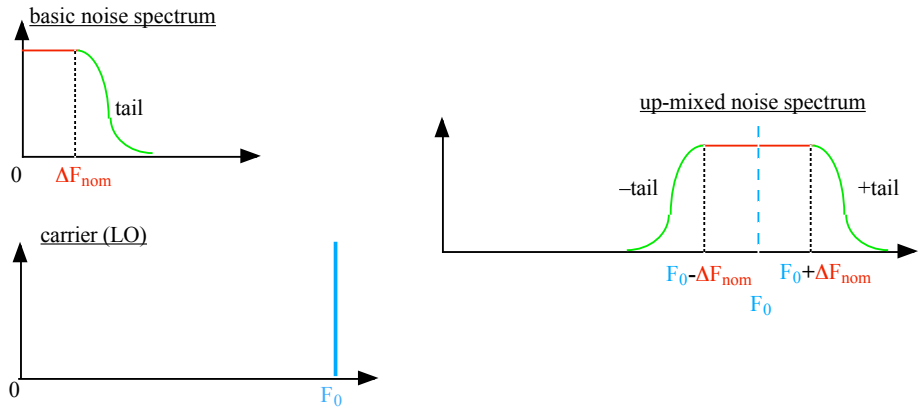


Fig. 1: The first band-limited noise generation to blow up bunches realized in hardware: mixing of the basic noise spectrum between 0 and ΔF_{nom} – having undesired tails – with a carrier at F_0

F_0 was changed as a function of time during the cycle i.e. all noise frequencies were shifted by the same amount. In MD sessions using the SPS as an LHC test-bed the blow-up was initially rapid but later levelled off [2] and was always at the limit between beam loss and insufficient blow-up for the desired effect.

Then a laboratory test was done. The noise-generation part of the simulation program was isolated, provided with an on-screen user interface and made an independent program. It accepts a trapezoidal definition of the noise spectrum – in practice sufficient to define all desirable shapes – and outputs a string of corresponding noise data. These are read into an arbitrary wave generator (AWG) and played back (at a 10 kHz rate) from there. The *measured* spectrum (on a HP 3562A) showed exactly the expected shape without any tails or side-lobes. This gave enough confidence to create a LabView© user-interface for the SPS [3], collecting the user parameters, calling the generator

program and uploading the (60,000) noise data onto an AWG. Triggering the AWG then gives a real time noise signal (of 6 seconds duration, determined by a 'linear gate') with the desired spectrum.

For a stable noise spectrum the noise generation program can directly define the 'shifted' spectrum between f_{low} ($\neq 0$) and f_{up}, avoiding the up-mixing and the undesired creation of the mirrored spectral part. But to 'slide' and/or 'scale' the noise, mixing and the inherent mirror spectrum is unavoidable for an initial constant spectrum.

In the meantime the possibilities given by simulated blow-ups using *fixed* spectra were about exhausted. Hence variable spectra, 'brushing' through the bunch in coast and/or more precisely following the synchrotron frequency changes during acceleration – not only shifting a constant spectrum but also scaling its width – became desirable, implying that the same noise could be generated later in real-time. The noise-generation programme was therefore upgraded to provide this facility and this 'noise package' (with small adaptations) is now applied for simulation purposes in "NoisySync" and also in real time noise generation for the SPS. To use it in the latter way the LabView interface has also been upgraded to produce variable spectra. To describe the functional dependencies for the SPS case it is sufficient to use two contiguous straight lines describing the upper frequency bound and two contiguous straight lines describing the lower, i.e. 3 data pairs (f_{Low}, f_{Up}) at t=0, $t=t_{end}$ and one between are sufficient (see also example in Fig. 6).

In LHC a similar method can be applied but can have better parameterization due to the much longer blow-up time available and also should have specific hardware calculating and generating (DAC) the noise signal in 'parallel', thus avoiding the lengthy upload time and an expensive AWG. Also feedback from measured synchrotron frequencies might be used to slightly correct the emitted noise data in real-time if the beam reacts slightly differently than anticipated.

In the above procedure the time-variation of the spectrum is limited to the scaling and shift of a given arbitrary, but fixed, spectral shape, the shift and scale magnitudes being defined by arbitrary functions. This degree of freedom seems sufficient for all needs and no further efforts were made in this direction. However, by slowly merging various spectra having different spectral shapes while conserving the statistical average value, practically any arbitrary case could be constructed if required one day.

2. Noise with Arbitrary Stable Spectral Density

2.1 Generating a Finite Noise Data Set with Constant Arbitrary Spectral Density

The standard method to create 'coloured' noise in hardware is to filter white noise (from an appropriate source) with a suitable filter. The complexity of the filter and its parameters depends on the spectral shape and the required spectral cleanliness. Smooth change to the output spectral density in real-time is then in general a major task.

One might adapt this method to a digital version by applying a digital filter but this does not allow defining easily any spectral density without complicated parameter calculations and tuning. Therefore we use another implementation – with only finite length non-periodic data output in a first step – which consists of transforming a time-domain white noise data string of finite length N by Fourier transform into the frequency domain, multiplying the spectral amplitudes by the desired relative amplitude $\rho(f)$ and then transforming back into time-domain.

White noise can be produced numerically by creating sequential numbers $\{G_k\}$ that are mutually statistically independent and have a zero balanced Gaussian probability distribution. It is well known (see Appendix B) that such a sequence with standard deviation $\sigma = 1$ can be created from 2N standard random numbers $\{rand_m\}$, having equidistribution in]0,1[, by using

$$G_k = \cos(2\pi \cdot rand_{2k}) \cdot \sqrt{-2 \cdot \ln(rand_{2k+1})}$$

For reasons that become clear later (avoiding mirror spectra when mixing) we want to distinguish between positive and negative noise frequencies, hence we use a complex noise description from

which at the end only the real part will be used. Therefore we apply the corresponding complex description $\{g_k\}$ with $G_k=\mathrm{Re}[g_k]$

$$g_k \;=\; \exp(2\pi\cdot i\cdot rand_{2k})\cdot\sqrt{-2\cdot\ln(rand_{2k+1})} \;=\; \left[\cos(2\pi\cdot rand_{2k}) \;+\; i\cdot\sin(2\pi\cdot rand_{2k})\right]\cdot\sqrt{-2\cdot\ln(rand_{2k+1})}$$

We Fourier transform this string of N complex data into the frequency domain, multiply the amplitudes by the real (absolute) amplitude distribution function $\rho(f)$ (proportional to the square-root of the desired spectral noise density) and transform back into the time domain to get the set $\{r_k\}$ with the real part $\mathrm{Re}[r_k]=R_k$. Evidently for constant $\rho=1$ one gets back the initial white noise.

This set has an average expectation value $<r_k>$ of zero (both real and imaginary part). Only the real parts $R_k=\mathrm{Re}(r_k)$ of the complex data set $\{r_k\}$ of magnitude N is used as noise data set $\{ R_k \}$; it has the desired spectral density. We assume for the following that $\{R_k\}$ has been scaled to an rms value of unity (in fact $\{r_k\}$ including the imaginary part is scaled by this factor), hence we have

$$< R_k > \;=\; 0 \qquad \text{and} \qquad < R_k^2 > \;=\; 1$$

The result has a finite length N but can be used as the basis for an improved method.

2.2 Creating a Quasi-Infinite[1] Number of Non-Periodic Noise Data

The real data set $\{R_k\}$ created above has a finite length N. From practical considerations (CPU time per data increasing as approximately $\log_2(N)$ but more critically memory limitation) N is limited. $\{R_k\}$ is periodic over its full length hence re-using it from the beginning would show no discontinuity but would result in periodic noise output, to be avoided in our context. Switching to a new set created from new random numbers will create in general a discontinuity, equivalent to a delta-pulse, which will appear as an undesired signal at all frequencies. To avoid this effect, we always use in parallel two statistically independent finite sets $\{r_k\}$ and $\{s_k\}$ of identical length N and spectral density. One can now merge $\{r_k\}$ and $\{s_k\}$ while keeping the statistical properties unchanged. For any given angle ψ we use $\cos(\psi)$ and $\sin(\psi)$ to define the complex variable

$$u_k \;= r_k\cdot\cos(\psi) \;+\; s_k\cdot\sin(\psi)$$

and for the real parts

$$U_k \;= R_k\cdot\cos(\psi) \;+\; S_k\cdot\sin(\psi)$$

From $<R_k> = <S_k> = 0$ it is evident that

$$< U_k > \;=\; < R_k\cdot\cos(\psi) \;+\; S_k\cdot\sin(\psi) > \;=\; < R_k >\cdot\cos(\psi) \;+\; < S_k >\cdot\sin(\psi) > \;=\; 0$$

i.e. also $\{U_k\}$ is zero-balanced. Since $\{R_k\}$ and $\{S_k\}$ are statistically independent, zero-balanced and both have the rms value 1, one has

$$< U_k^2 > \;=\; < R_k^2 >\cdot\cos^2(\psi) \;+\; < S_k^2 >\cdot\sin^2(\psi) \;+2\cos(\psi)\;\cdot\sin(\psi)\cdot< R_k\cdot S_k > \;=\; 1$$

The sets $\{R_k\}$ and $\{S_k\}$ both have the given design noise power $p(f)\delta f$ in the frequency range $[f, f+\delta f]$, i.e. the expectation values of the squares of the filtered signal between f and $f+\delta f$ – having the noise sets $< \hat{R}_k^2 \,|\, [f, f+\delta f] >$ and $< \hat{S}_k^2 \,|\, [f, f+\delta f] >$ are

$$< \hat{R}_k^2 \,|\, [f, f+\delta f] > \;=\; < \hat{S}_k^2 \,|\, [f, f+\delta f] > \;=\; p(f)\;\delta f$$

The filtered sets are also statistically independent and zero balanced, hence one obtains for any f

$$< \hat{U}_k^2 \,|\, [f, f+\delta f] > \;=\; < \hat{R}_k^2 \,|\, [f, f+\delta f] >\cdot\cos^2(\psi) + \; < \hat{S}_k^2 \,|\, [f, f+\delta f] >\cdot\sin^2(\psi) \;+$$
$$< \hat{R}_k \,|\, [f, f+\delta f]\cdot\hat{S}_k \,|\, [f, f+\delta f] > \sin(\psi)\cos(\psi) \;=\; p(f)\cdot df$$

---

[1] It can easily be proven that there cannot be a truly infinite non-periodic data set with a finite number of bits but the periodicity length here is many orders of magnitude longer than the number of revolutions of all particles in all synchrotrons for the age of the universe, hence it is *practically* infinite.

The (unfiltered) set $\{U_k\}$ therefore has the same spectral density as $\{R_k\}$ and $\{S_k\}$, i.e. is an equivalent set.

The same chain of arguments holds for the imaginary parts of $\{r_k\}$, $\{s_k\}$ and $\{u_k\}$, and so one can use the complex variables r, s and u during the transformations taking the real part at the end.

We consider now an angle ψ that changes with time with the slowest possible constant speed, i.e. from 0 to 2π while all N available data are used, i.e. $\psi_k=2\pi k/N$, k=0 to N-1. This will create a minimal and also smooth perturbation to the spectrum of $\{u_k\}$ with the lowest possible frequency corresponding to the lowest present frequency. At the instant $\psi=2\pi$ one has $u_k=r_k$ independent of s_k, hence at this instant one can switch to a new independent set $\{s_k\}$ without discontinuity. Similarly at $\psi=3\pi/2$ $u_k=s_k$ independent of r_k, and one can switch to a new independent $\{r_k\}$.

While this method of array switching excludes any discontinuity for the function values themselves, there is no guarantee for derivatives. However, around the array-switching time the weighting functions of the switched set $\sin(\psi)$ or $\cos(\psi)$ are very small so that the discontinuities of derivatives in the weighted sum of r_k and s_k are not precisely zero but at least very small. Furthermore the output data (see later) can be obtained by 6-point interpolation smoothing the transition over ±3 data.

One might have thought of letting ψ move at half the speed from 0 to π only; this would also allow switching the sets at $\psi=\pi$ and $\psi=\pi/2$. However, the full turn from 0 to 2π creates only a single very low frequency modulation whilst this solution would correspond to a weight/modulation function $|\sin(\psi)|$ and generate higher harmonics at multiples of this lowest frequency. Therefore the first solution is preferred.

Practically the sets $\{r_k\}$ and $\{s_k\}$ are used starting at k=0 to build the corresponding u_0 up to $u_{3N/4-1}$ (N has to be a multiple of 4). Before $r_{3N/4}$ and $s_{3N/4}$ are used, $\{r_k\}$ is replaced completely by a new, independent, set, $\{s_k\}$ remains as it is. Once r_{N-1} and s_{N-1} are used, $\{s_k\}$ is completely replaced while $\{r_k\}$ remains as it is and k is reset to zero. This guarantees the output of a quasi-infinite non-periodic series of noise data u_k with the desired (constant) spectral density $\{U_k\}$ based on a chain of sets of finite length N. Of the first applied set $\{r_k\}$ the fourth quarter of the data are not used but this is not a serious waste.

Advancing ψ and creating new arrays, as well as updating constants and indices, is done automatically without user intervention.

2.3 Absolute Amplitude Normalization

Noise spectra are expressed in absolute terms by their local noise 'power' density at the frequency f, hence if the signal is measured in the abstract unit X (e.g. [º] or [s] for RF phase noise, [V] for RF amplitude noise), it is $<X^2>$/Hz. It also might be expressed by its square root, $X_{rms}/\sqrt{Hz}$. In practice one can measure it by filtering the frequency range between [f, f+δf], and determining $\sqrt{<X^2>}=X_{rms}$ of the remaining signal. The local noise power density around f is then $X^2_{rms}/\delta f$, using a small but finite δf; mathematically one uses the limit as $\delta f \to 0$. The noise signal here has as yet no units (they will be inserted later by the user), hence the local noise 'power' density is expressed in [1/Hz].

A priori nothing is known about the shape and range of spectra that have to be produced, hence one cannot use the local noise 'power' in [1/Hz] at a certain frequency for calibration. The only reasonable absolute amplitude definition is done by fixing the total rms-value of the whole signal over all frequencies. The software package is made such that the user prescribes the rms value U_0 of the total spectrum and the output signal then has a total noise 'power' $P_{tot}=U_0^2$, i.e. $U_0=U_{rms,tot}$.

Any local 'power' density p(f) can then be calculated for the user-defined *relative rms-amplitude distribution* function[2] ρ(f) – proportional to the *square-root* of p(f) – and total calibration factor $U_{rms,tot}=U_0$, by

[2] generic name `'RelShape( )'` in the program

$$p(f) \;=\; U_0^2 \cdot \frac{\rho^2(f)}{\int_0^{+\infty} \rho^2(f')\; df'} \qquad \left[\frac{U^2}{Hz}\right]$$

while respecting the calibration relation

$$P_{tot} \;=\; \int_0^{+\infty} p(f')\; df' \;=\; U_0^2$$

An equivalent form is

$$\sqrt{p(f)} \;=\; U_0 \cdot \frac{\rho(f)}{\sqrt{\int_0^{+\infty} \rho^2(f')df'}} \qquad \left[\frac{U}{\sqrt{Hz}}\right]$$

2.4 Remarks on the Usable Range of the Spectral Density Definition

In our case the noise data are observed by the bunch stroboscopically (e.g. when the particle passes the 'noisy' cavity) with the time tick[3] $T=1/f_{Clock}$. This means that an observing analyzer in the range $0 \le f \le f_{Clock}$ detects the same spectrum as an analyzer observing in the range $m \cdot f_{Clock} \le f \le (m+1) \cdot f_{Clock}$ for any integer m, positive or negative. Therefore a spectral definition in $0 \le f \le f_{Clock}$ already defines the spectrum seen by the particles 'up to infinity'.

Only the real part of the complex calculated data will be used as noise converted to e.g. phase or voltage. This means that the complex contributions $\exp(2\pi i f \cdot t)$ and $\exp(-2\pi i f \cdot t)$ both generate the same (observed) real value $\cos(2\pi f \cdot t)$ and are hence indistinguishable to the observer. But any frequency f might be shifted, indistinguishably, to $f+m \cdot f_{Clock}$ for any integer m. Doing this with m=-1 for $\exp(-2\pi i f \cdot t)$ we get the two identical contributions from $\exp(2\pi i f)$ and $\exp(2\pi i (f_{Clock}-f))$, see Fig. 2.

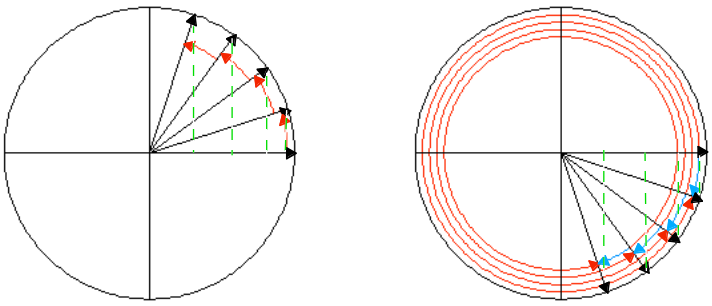


Fig. 2: Movement of a complex vector in the complex plane (real=x, imaginary=y) with frequency f; positive frequencies advance counter-clockwise. Observed is the real part (projection on x-axis, green dashed lines) in stroboscopic mode with the rate f_{Clock}. If $f = f_{Clock}$ the vector makes one full turn per step. At left a 'low' positive frequency with $f_1 < f_{Clock}/2$ (red counter-clockwise arcs) is shown. At right a 'high' positive frequency $f_2 = f_{Clock} - f_1 > f_{Clock}/2$ (red counter-clockwise arcs) and "high" negative frequency with $f=-f_1$ (blue clockwise arcs) are shown. All three have the same series of real parts (projection on the x-axis, green dashed lines) and are therefore indistinguishable for the particle passing a cavity once per machine turn, $f_{Rev}=f_{Clock}$. Evidently all these frequencies ± any multiple of f_{Clock} appear indistinguishable from those shown.

This means also that all lines in $0 \le f \le f_{Clock}$ have a symmetric 'twin' with respect to $f_{Clock}/2$, as also shown in Fig. 3 with the basic rectangular spectrum in 2-4 kHz and its twin in 6-8 kHz. It is only necessary to define the spectral density for the range f=0 to $f = f_{Clock}/2$, all other parts of the (real) spectrum are then uniquely defined. Therefore one should pay attention so that the upper frequency band limit – called f_{up} – of any such band (especially for variable noise) never passes above $f_{Clock}/2$.

[3] For simulations T has to be identical to T_{rev}; to create real-time noise it has to be much smaller than $1/f_{noise,max}$ to avoid granularity of the digital output.

The program does *not* check this condition and if it occurs, the resulting output spectrum is not as expected but depends on random factors.

In this spirit the definition of perfect white noise is done by setting the noise band from f_{low}=0 to $f_{up}=f_{Clock}/2$ (not f_{Clock} !) and the spectral absolute amplitude function (`RelShape,` see later) has to give any constant non-zero value for the full definition range from 0 to 1.

Fig. 4 shows examples of chosen spectra and their 'measurement' displayed in $0 \le f \le f_{Clock}/2$; this avoids showing the upper mirror spectrum as is done in Fig. 3.

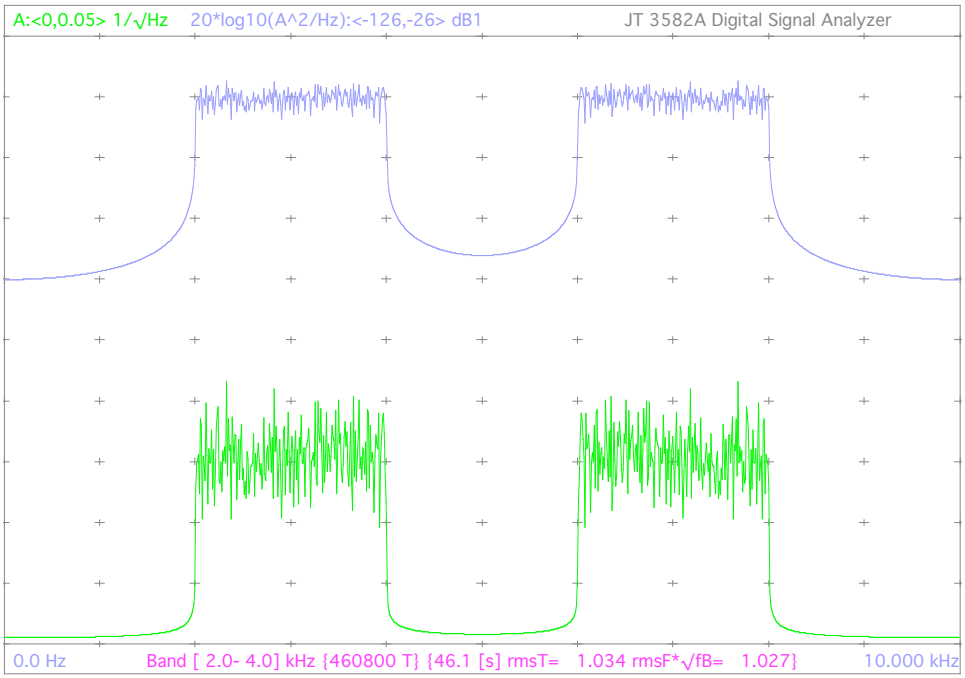


Fig. 3: Numerically created signal with rectangular noise spectrum with sharp limits, display 0 – 10 kHz = f_{Clock}. This display range shows the direct spectrum in the band from 2 to 4 kHz but also the always-present mirror spectrum (10-4=6 kHz to 10-2=8 kHz). The 'measurement' is done with a 'software digital signal analyzer' (incorporated in the simulation program [1] for checks) in exp-average mode. Each new spectrum is created by collecting 1024 new time-domain data and executing an FFT for $1024 = 2^{10}$ data. Green: linear amplitude scale, blue: logarithmic power scale (10 dB/div); the latter trace shows that there are no spurious side-lobes at all, even at high resolution.

2.5 The Pseudo-Random Generator(s)

Nowadays all computer operating systems supply a pseudo-random generator. These are certainly good enough to roll dice or distribute cards in games of chance on a PC but the statistical quality in long runs and especially freedom from sequential correlations is not guaranteed a priori. To be sure we exclusively rely on probably the best pseudo-random generator presently available for *this* purpose – not for cryptography – the '*Mersenne Twister*' in its implementation MT19937 [4]. The period length is larger than 10^{6000} (2^{19937}–1 to be precise), but most important is the fact that a 623-dimensional equidistribution property is assured [4]. This means that at least any 623 sequential data are free of correlations[4]. As a comparison, the often-used *linear-congruential* pseudo-random generators do not have this property at all, a *very serious flaw* in the context of long simulations, especially if one 'event' (here the final position of the particle in phase space) depends on a sequential chain of random-calls (here the different noise-kicks). By avoiding multiplication and especially division MT19937 is even faster than most (lower quality) competitors.

The author of this paper has made a few small modifications in his implementation(s). First, the seed can be chosen either by (user defined) fixed data, allowing the same pseudo-random sequence for each new run to be repeated – important for debugging – or by a computer clock generated seed, different for each new run, important to study the scatter of the simulation results.

[4] in the 32 bit implementation [4] as used here

In the simulation the placing of the macro-particles in phase-space – rendering a pre-defined J or z distribution within statistical scatter – also needs a pseudo-random generator. For this purpose a second, completely *independent* implementation of MT19937[5] with different function names was added. This allows one to study the statistical scatter using different noise sets for the same bunch or vice-versa.

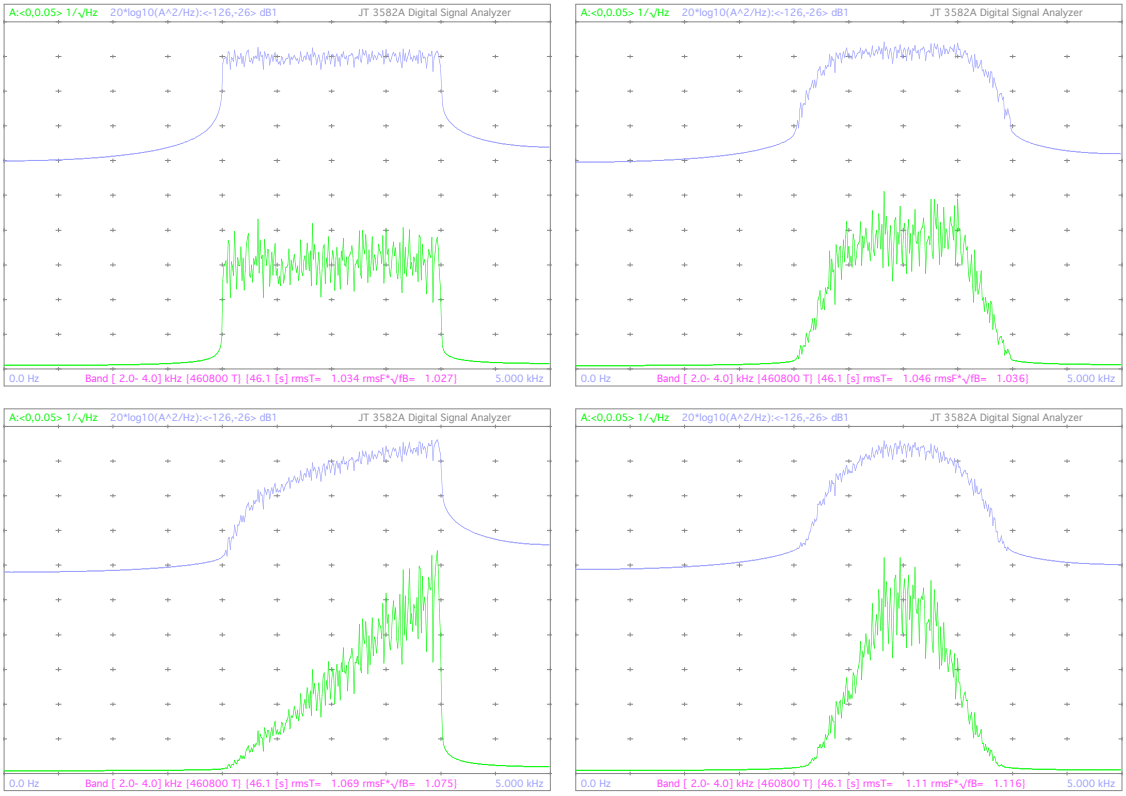


Fig. 4: Examples of numerically created signals with arbitrary (constant) noise spectra, all in a noise band from 2 to 4 kHz, display range of 0 – 5 kHz = $f_{Clock}/2$, clipping the upper mirror image spectrum, otherwise as Fig. 3. (top–left): rectangular as Fig 3, (top–right) trapezoidal, 25% rise and fall, 50% flat-top, (bottom–left): rising triangular as used in the SPS blow-up, (bottom–right) $\cos^2$ shaped.

2.6 The Applied Fast Fourier Transform

The FFT is generally understood to imply a number of data/channels that is a power of 2. This enforces a choice for the possible data length that may be somewhat coarse, especially for *large* powers of 2. The initial idea of the FFT can be generalized if the total number N of data/channels can be broken up as the product of factors which are as low as possible - the best would be prime factors. In the worst case N itself is prime leading to a classical 'slow' Fourier transform. A suitable function

```
PrimeFFT(iprim, nPrim, fr, fi, gr, gi)
```

existed in the author's software archive[6]. The 'nPrim' factors making up N are stored in the array 'iprim'; factors k^m enter as m times the entry k. f_r, f_i and g_r, g_i are arrays holding the real and imaginary part of the input (f) and output (g) arrays. A function that tries to split any N into (small) prime factors (below a given search limit) also exists; in the present program N may be automatically slightly increased to get a better splitting into smaller factors leading to faster execution despite the larger N.

[5] probably an overkill for, say, 10^4–10^6 macro-particles, but MT19937 is also very fast

[6] Developed for other but similar application

3. Variable Noise

3.1 Motivation

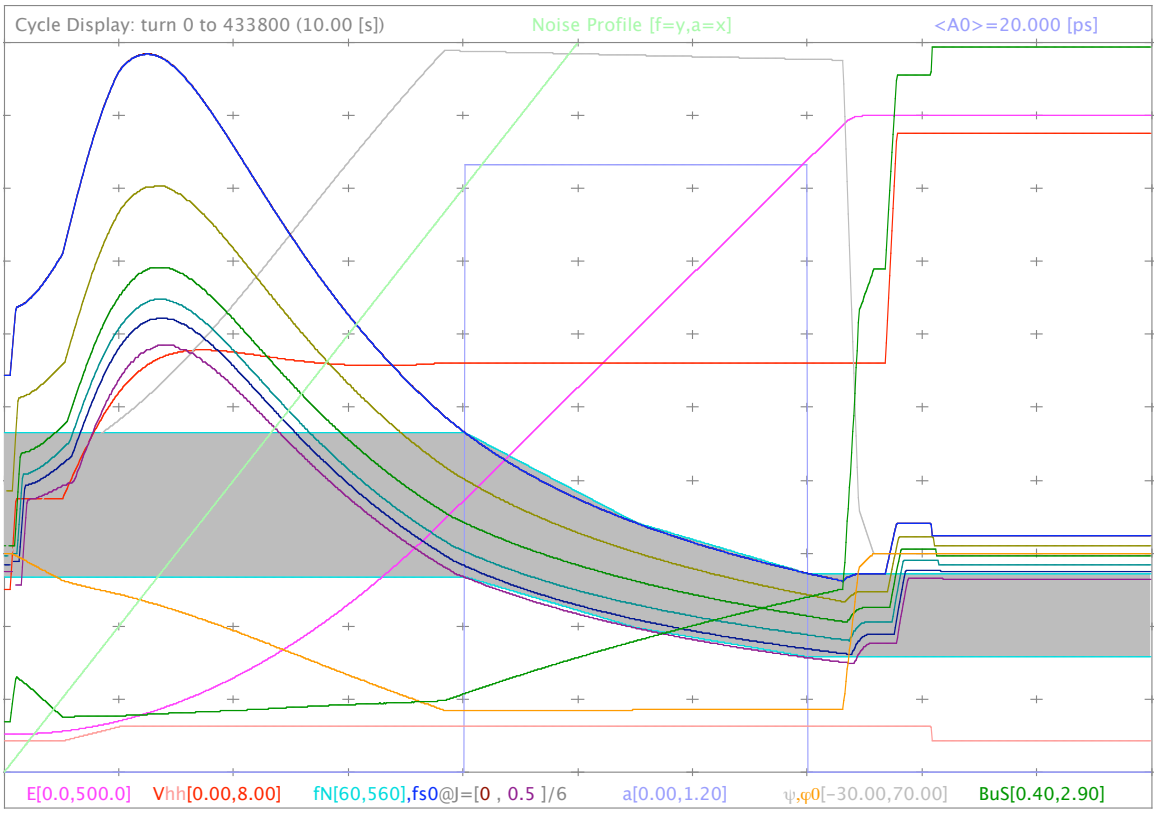


Fig. 5: Graphic[7] produced by 'NoisySync' [1] for a 10 s fraction of an SPS cycle starting with the last injection, here t=0. Shown are E_0 (magenta) 0–500 GeV, V_{200} (red), V_{800} (light red) both 0–8 MV and bucket size (dark green) 0.4–2.9eVs. [In reality the magnetic field on the flat-top is ramped down earlier (after ejection of the beam), in the present plot it is kept high for simplicity] The six lines having an upwards bump at the left represent the synchrotron frequencies [range 60 Hz - 560 Hz] of (unperturbed) particles corresponding to phase-space variables J=0 to 0.5 eVs, in steps of 0.1 eVs; the top line (dark blue) corresponds to f_{s0} with J=0. The grey strip depicts the foreseen excitation noise band but the noise is only switched on between t=4 s and t=7 s – as indicated by the lilac rectangular relative amplitude function. The noise band frequency limits follow those of particles at J=0 eVs (upper) and J=0.5 eVs (lower) to blow up the bunch to a limit of about 0.5 eVs with correspondingly reduced central population. In this example the function determining the band limits are simply two joined straight lines – obviously sufficient – but any, more sophisticated, smooth function could be used. The triangular noise profile is drawn in inverted coordinates with freq→y and amp→x (light green), to match the orientation of the depicted grey noise band (arbitrary units); strongest excitation on top for J=0 eVs (bunch centre) falling to zero excitation at J=0.5 eVs.

For our application we do *not* need *any* imaginable time dependence for the spectrum. It is sufficient that the initially designed spectral density, targeted onto a certain range of synchrotron frequencies with certain excitation amplitudes, can follow the frequency changes during the controlled slow machine changes such as acceleration. To do so it is sufficient that this spectrum follows the synchrotron frequencies of the particles, i.e. the higher end f_{up} follows the frequency of the central particle f_{s0} (according J_0=0) and the lower end f_{low} follows the frequency of an unperturbed particle encircling a certain emittance area J_1 in phase space, the intermediate frequencies in-between being scaled and shifted linearly with the end frequencies. This is illustrated in Fig. 5, (a part of) the SPS cycle with changing nominal machine momentum (bending field), main RF voltage at 200 MHz and a higher harmonic Landau system voltage at 800 MHz in bunch shortening mode. During the noise excitation the basic triangular noise profile – highest excitation at the bunch centre with J=0 and f=f_{s0} – is shifted and scaled (grey band) to keep excitation optimum. A practical execution for the SPS is

[7] The graphics output of 'NoisySync' only works on a Macintosh (OS X with PowerPC CPU) while the pure number-crunching part should work (almost) immediately under other systems by setting the pseudo-instruction #MACXXXX to undefined, thus excluding the MAC specific system calls.

sketched in Fig. 6. It seems evident that such scaling and shifting using arbitrary functions is sufficient for our purpose.

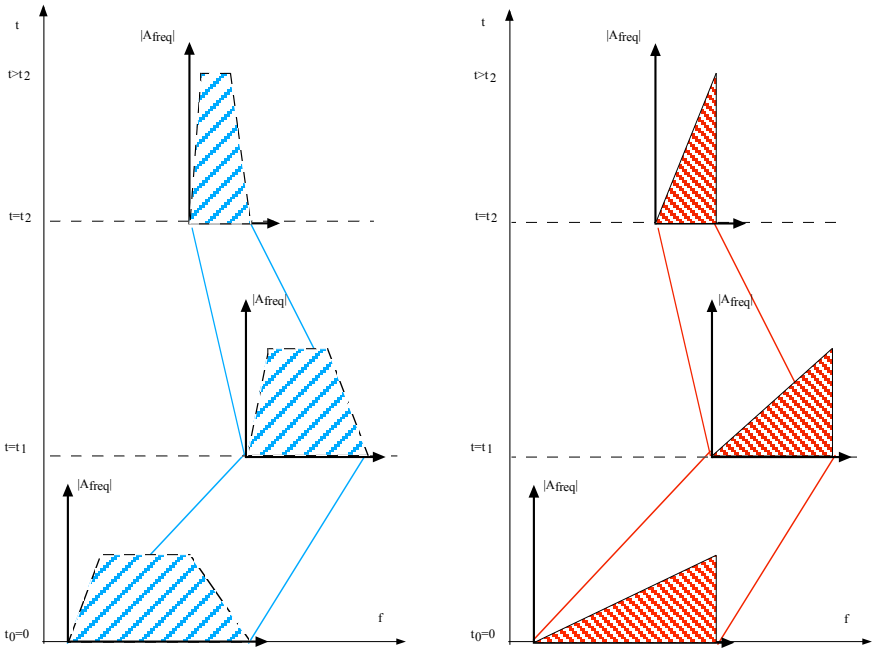


Fig. 6: Illustration of a variable spectrum with time running from bottom to top. The frequency limiting functions are both described by two adjacent straight lines (as applied in the SPS noise generator). Since, in this example, the spectrum contracts in width, the amplitude increases as the square root of the same factor (see below) to conserve the total noise 'power'. (left) trapezoidal basic spectrum; (right) triangular basic spectrum.

In principle it is of no importance if one first scales and then shifts the basic spectrum or vice versa. However for numerical reasons the first alternative is considerably better. As shown later, interpolation of the time-domain signal has to be applied for frequency scaling. Interpolation is much more precise if the function has only 'slow' variations as in the initial spectrum. The up-mixed signal is in general much 'faster' and needs much more careful interpolation to avoid spurious side-lobes in the output spectrum.

3.2 Frequency Scaling of the Basic Noise Signal

When music is recorded (analogue, not digital) on a gramophone record or magnetic tape and is played back faster or slower compared to the design speed, all frequencies appear scaled by the same ratio. The same effect is used here (in numerical realisation) to scale the basic frequency range. The basic spectrum is set up as if it would be used between 0 to f_{Clock}, hence to respect $f \le f_{Clock}/2$ (see above) the frequency-scaling factor is ≤1/2. The interpolation precision (see later) increases for lower scaling factors since intervals between data points get smaller.

The 'infinite' string of time-domain signal created above does not come as a smooth function but only in regularly distributed points, making good interpolation – except for very slow waves with $f \ll f_{Clock}$ – impossible. To increase the number of points for interpolation we increase the number of channels in the Fourier transformation. As depicted in Fig. 7 instead of starting with a frequency-domain signal of n_{Source} channels (sources), we create a spectral representation with $n_{Source} \cdot n_{Pnt}$ channels, n_{Pnt} being an integer, e.g. 8. Only the lowest channels, 0 to (n_{Source}-1), are defined with non-zero amplitude as used in the previous section, all other channels, n_{Source} to $n_{Source} \cdot n_{Pnt}$, have zero amplitude. In this way a (fast) Fourier transformation results in a time-domain signal with the highest frequency components being sampled by n_{Pnt} points per oscillation and the lower frequencies with proportionately more points. This allows interpolation at all frequencies up to the highest present with very good precision. The design play-back rate is f_{Clock} but if we would take one data per clock-tick now, the generated frequency would be n_{Pnt} times lower than before, i.e. the 'sampling time step' corresponding to the frequency f_{Clock} is n_{Pnt} times as wide as before. Evidently a step-width of $n_{Pnt}/2$ would produce the highest frequency signal at $f_{Clock}/2$. This is easily executed for even n_{Pnt} with integer

$n_{Pnt}/2$. However, now there are sufficient data points present to interpolate (6 point polynomial) smoothly between the regularly spaced data points and it is easy to define any non integer step-width as e.g. $0.168 \cdot n_{Pnt}$ (see e.g. Fig. 8), creating a signal with the lowest frequency at zero and the highest spectral frequency $\Delta f = 0.168 \cdot f_{Clock}$.

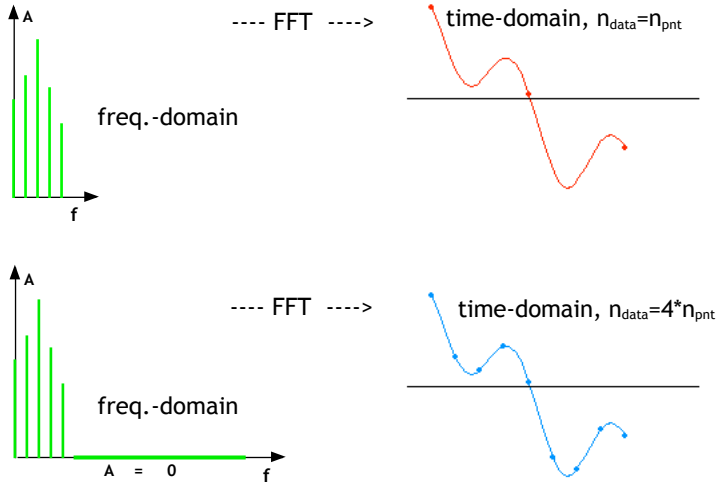


Fig. 7: FFT of n_{Source} amplitudes. (Top): standard transformation with (left) n_{Source} channels and (right) their FFT into time domain with n_{Sorce} channels/data points. Bottom: (left) with a factor n_{Pnt} (4 in this sketch) enlarged number of channels to $n_{Pnt} \cdot n_{Source}$ by adding the top $(n_{Pnt}-1) \cdot n_{Source}$ channels with zero amplitudes and (right) the resulting FFT with $n_{Pnt} \cdot n_{Source}$ data, hence n_{Pnt} times the density of points in time-domain. This allows precise interpolation for any arbitrary time-step (in reality $n_{Pnt}=8$ is used).

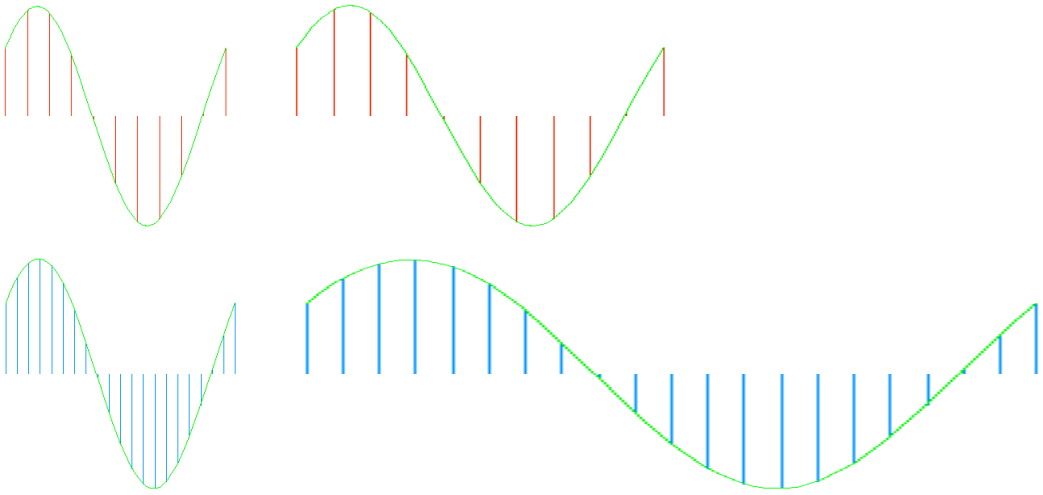


Fig. 8: Smoothly interpolated steps in 'time' domain of the data delivered by the FFT in Fig. 7. Left: sampling (interpolating) by step-width of (top–left) e.g. '0.336' and (bottom–left) with half the previous step-width, i.e. '0.168'. Right: *Invariant* playback with *one data per clock-tick* $1/f_{Clock}$ with (top–right) twice the frequency, $0.336 \cdot f_{Clock}$, as (bottom–right) $0.168 \cdot f_{Clock}$.

To be on the safe side we apply $n_{Pnt}=8$ so that the highest created frequency has enough interpolation points per oscillation to give very good interpolation results. In fact, on Fig. 3 it can be seen that even in logarithmic display – effectively magnifying small signal levels – side-lobes or perturbing lines do not show up at all.

In practice this means that the FFT has to be done over a range n_{Pnt} (8) times longer than for the simple fixed noise initially envisaged. But even for large number of points the FFT is very fast.

At the end of this step one gets an 'infinite' stream of time-domain data – the sliding merger of two arrays of $n_{Source} \cdot n_{Pnt}$ data $\{r_k\}$ and $\{s_k\}$ resulting in $\{u_k\}$ as shown above is always applied to get an quasi unlimited stream – now having a spectrum of pre-defined shape between 0 and the chosen Δf, by using the step width $\Delta f / f_{Clock} \cdot n_{Pnt}$. Since we want to up-mix this signal without creating mirrored lobes we have to retain only positive frequencies. To do so we preserve not only the real but also the

imaginary part of the interpolated time-domain signal, i.e. we keep a complex valued oscillation for the time being.

3.3 Frequency Shift of the Scaled Noise Signal

Shifting a spectrum can be done by mixing the time-domain signal with a local oscillator (LO). In a hardware implementation with a standard mixer/multiplier upper and lower sidebands appear, i.e. not only an upper spectrum just above the LO frequency appears but also its mirror image just below LO, the initial spectrum starting at zero. This effect was a limitation in the initial hardware-based set-up. By numerical treatment it is easy to create the signals not as real but as *complex* numbers (see above) with positive frequencies exclusively. When the LO signal is also defined as complex with a (unique) positive frequency, mixing/multiplication produces no negative frequencies and there are no side-lobes. The resulting output time-domain signal has a clean spectrum without mirror images or spurious residuals just below the LO frequency. It is evident that for our case the LO frequency should correspond to the lower band end frequency f_{low}, shifting the initial spectrum from $[0, \Delta f]$ to $[f_{low}, f_{low}+\Delta f] = [f_{low}, f_{up}]$.

Practically a *complex* number e with unity absolute value represents the LO status variable. Since one noise data has to be produced for each clock tick T_{clock}, this complex status variable turns in the complex plane at each clock tick as

$$e_k \cdot \exp\left(2\pi \cdot i \cdot f_{low,k} \cdot T_{Clock}\right) \rightarrow e_{k+1}$$

with the (possibly varying) lower band limit frequency $f_{low,k}$ at the 'instant' k. Evidently the absolute value of e is conserved. The complex time domain signal $s_{scsh,k}$, scaled and shifted, is then the complex product

$$s_{scsh,k} = e_k \cdot s_{sc,k}$$

where $s_{sc,k}$ is the scaled signal as obtained in the last section. The noise value $n_{out,k}$ really used is then the real part of $s_{scsh,k}$. (The imaginary part would also work. However both should not be used as they are correlated).

$$n_{out,k} = \mathrm{Re}\left[s_{scsh,k}\right]$$

To avoid that the absolute value of e diverges numerically from unity during the many steps, it is re-normalized 'from time to time'.

Fig. 6 shows an example of variable noise with scaling and shifting where the functions for f_{up} and f_{low} are defined by two consecutive straight lines each sufficient for the frequency variation encountered in the SPS.

3.4 Amplitude Normalization of the Variable Noise Signal

Evidently the frequency shift of a given spectrum does not change the value of the local 'noise-power' in [1/Hz], only the corresponding frequency-location is shifted. But when a spectrum is scaled by a frequency factor of e.g. 1/4 in width, the *local* noise 'power' increases by a factor $\sqrt{4}=2$, the total 'noise-power' remaining constant. This relation might be desired or not, in any case it is *not* modified. If the user wants to keep the local power constant – or changing by some other law – the relative amplitude function can be updated by `SetBandRelAmp( arel )` or together with the frequencies as well by `SetBandPosAndRelAmp( , , arel )` (for more details see Appendix).

4. Monochromatic Sharp Lines

Perturbations to the beam often arise from induced signals in the RF at the power grid frequency and its multiples, e.g. those generated in power converters. Describing such lines by a 'smooth' spectrum is clumsy and not very precise. Therefore isolated monochromatic lines with an arbitrarily sharp frequency $f^{(m)}$, amplitude $a^{(m)}$ and start phase $\psi^{(m)}$ for line m can be defined. The lines are created

precisely at the desired frequency using a complex status variable as shown for the local oscillator signal in section 3.3., hence for line m with the frequency $f^{(m)}(t)$

$$e_k^{(m)} \cdot \exp\left(2\pi \cdot i \cdot f_k^{(m)} \cdot T_{Clock}\right) \rightarrow e_{k+1}^{(m)} \quad \text{starting with} \quad e_0^{(m)} = \exp\left(i \cdot \psi^{(m)}\right)$$

These lines may be absent (no initialization, e.g. Fig. 3), uniquely present without smooth noise (no smooth noise initialization, Fig. 9a) or with both lines and noise superimposed (both initialized, Fig 9b). Output n_k at step k is the real part of the *sum of all initialized lines*, i.e.

$$n_k = \mathrm{Re}\left[\sum_{lines} a^{(m)} \cdot e_k^{(m)}\right]$$

Sometimes sharp noise lines are part of a variable smooth noise spectrum or one wants to simulate a line 'brushing' through a bunch. Therefore a Boolean variable exists for each individual line telling whether the line slides with the spectrum (true, applying the same frequency scaling law as for the smooth noise, i.e. lines keep their position with respect to the varying smooth noise) or remains fixed at its frequency (false).

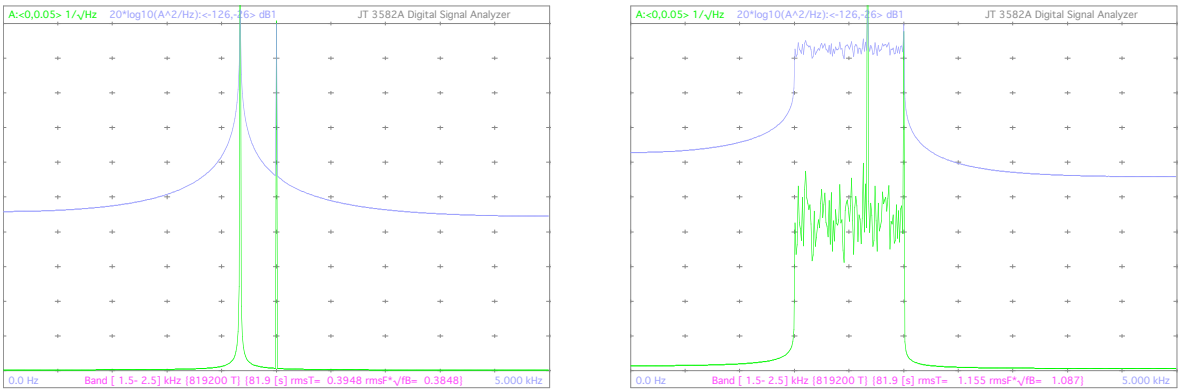


Fig. 9: (a, left): two lines without smooth noise; (b, right) two lines sliding with the smooth noise spectrum.

5. Conclusions

The generation of noise signals with arbitrary spectral density was successfully achieved and applied in simulations for the PS, SPS and LHC. The monochromatic line option was applied to check the influence of the induced signals at 50 Hz multiples of the power-grid as measured on the cavity RF phase in LHC [5].

For the SPS as LHC test-bed the initial simple hardware set-up with large tails and mirror spectrum was successfully replaced by the digital approach described above. This has no mirror-spectrum and is spectrally very clean. Also modifications to the spectral parameters could be easily done by simply changing some numbers on a graphical user-interface (GUI).

This successful test leads us to hope that a similar technique can be used in the LHC for the longitudinal blow-up required between injection and top energy to avoid large intra beam scattering in coast. The implementation would then run on a single computer in conjunction with a fast DAC – to avoid the slow uploading and the cost of an AWG – that would 'in parallel' calculate chunks of noise data and send them in analogue form to the main RF system phase loop.

Appendix A: Some Practical Considerations

A.1 Smooth Noise Spectra

The basic spectral *amplitude shape distribution* $\rho(x)$ (proportional to the square root of the spectral density) is defined once and for all by a generic (double) function `RelShape(x)`, with the normalized frequency co-ordinate $0 \leq x \leq 1$. In the application later '0' is always scaled and shifted to the actual f_{low}, '1' to the actual f_{up}. The function used is relative, i.e. absolute scaling of `RelShape` is of no importance; the true user function can have any (other) name.

Before using the noise-package, an initialisation call to define all fixed parameters has to be placed to the void function

```
InitializeNoise(fClock,rms,RelShape,nSourceMin,nPntMin,iRand).
```

Here "f_{Clock}" is the intended play-back clock rate [Hz] giving the correct absolute frequency calibration, "rms" the rms expectation-value of the output signal (for variable relative amplitude 1), "`RelShape`" the function pointer to the spectral amplitude shape function and "$n_{Source,Min}$" the minimum number of elementary noise sources in a basic data set which the set-up procedure may slightly increase to speed up later execution. The number of elementary noise sources should be large enough – i.e. tight enough – so that they cannot be clearly resolved within the 'running time' of the data set. "$n_{Pnt,Min}$" is the minimum number of interpolation points for the fastest elementary oscillator (at least 8 recommended, prime numbers to be avoided).

The variable "iRand" is a (long) random seed value and flag at the same time. If it is negative then a different, automatically generated, computer clock seed is applied at each run, forcing a different random set each time. If it is larger/equal to zero then the input data iRand is directly used to create the seed, hence always the same random data are created.

`InitializeNoise` also allocates the needed memory and keeps track of it. This function can hence be called many times, even for different parameters, and the memory is adjusted correspondingly. To restart with the *same* conditions, `RestartNoise( )` is easier to use. If a fixed seed was chosen at initialisation, exactly the same noise as before is reproduced (e.g. to apply the same noise to different applications), otherwise a new computer clock generated seed defines the following noise output.

Just after set-up, the three 'spectral parameters' f_{low}, f_{up} and a_{rel} have to be defined at least once by a function call to `SetBandPosAndRelAmp(flow,fup,arel)`. From then on any call to the (double) function `NextVNoise()` delivers the next noise data with the actually valid spectral parameters. To avoid running with different software (or extensive switching) we only keep the most versatile version, i.e. the one with variable noise capability, and use it also for *stable* noise spectra, leaving `flow, fup` and `arel` as initially defined. The overhead compared to an 'only fixed noise' version is very small and e.g. does not hinder real-time noise creation with high f_{Clock}.

To modify the spectral settings, `SetBandPosAndRelAmp(flow,fup,arel)`, `SetBandPos(flow,fup)` or `SetBandRelAmp(arel)` can be called for each change of parameter(s). For a continuous change it is easier to call directly one of the overloaded[8] instances of `NextVNoise():` `NextVNoise(flow,fup)` or `NextVNoise(flow,fup,ramp)`.

It is up to the user to construct functions such as `flow=GetFlow( turn )` to define the spectral parameters as a function of turn (or time) if this is required (variable spectrum). In his simulation program NoisySync the author has data pairs for each turn-number and e.g. f_{low} stored in a C-structure. There is also a function that linearly interpolates between these (not necessarily regularly spaced) data points. Standard SPS files with e.g. V_{200} as a function of cycle time [ms] can be read in as they exist and are then transferred onto an appropriate C-structure for use by the linear interpolation function.

---

[8] a function is called overloaded when it has for the same name different versions distinguished by differing number or type of arguments.

Using NoisySync the synchrotron frequencies of the different particles can also be plotted and/or written in text form to determine f_{low} and f_{up} for changing cycle data, such as V_{200}. These data can then be stored in a C-structure and are directly used for the simulation.

A.2 Monochromatic Sharp Lines

By calling the following set-up function monochromatic lines are initialized:

```
InitializeGenMonoLines(nMono,doSlide,freq,amp,psiDeg)
```

"nMono" is the total number of lines. The other arguments are arrays (pointers) with one entry per line each: "freq" [Hz], "amp" [1] the absolute amplitude, "psiDeg" [º] the starting angle (all double). "doSlide" is a Boolean flag determining if the line should stay constant in frequency even when the smooth part moves (false) or should slide as if it would be part of the smooth noise (true). The initialization function also allocates memory and can be called many times with differing parameters; memory and constants are reset each call. To restart with the same conditions, a call for `RestartGenMonoLines( )` is sufficient.

The total signal-level from all lines is calculated in a call to `NextMonoLines( )`, a double function. Frequencies and amplitudes of lines declared fixed, i.e. which do not slide with the smooth range, can also be displaced or allowed to slide *manually* independent of the smooth spectrum's sliding.

A single line, number "iLine" (counting starts at 0!), can be changed in frequency by `ChangeMonoLineFreq(iLine,newF)` or in amplitude by calling the function `ChangeMonoLineAmp(iLine, newA )`. All lines can be changed together using one call, faster than several individual line calls, by `ChangeMonoLineFreq(newFs)` and `ChangeMonoLineAmp(newAs)` where "newFs" and "newAs" are arrays containing "nMono" entries with the new settings (overloaded functions).

Appendix B: Transformation of random number sets

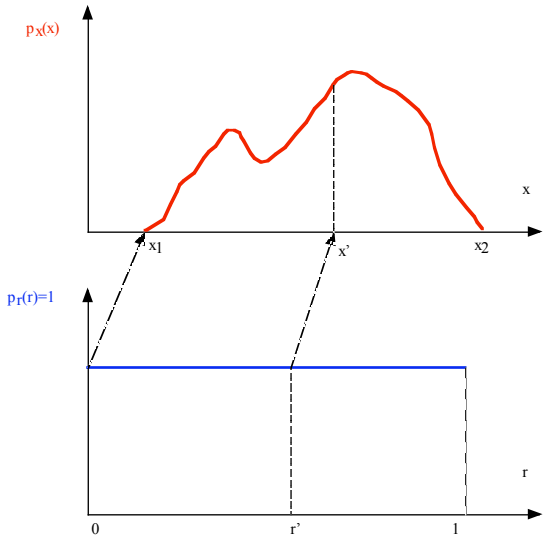


Fig. B1: Two linked probability density functions $p_x(x)$ and $p_r(r)$.

To create a sequence of random numbers x_k with an arbitrarily chosen density distribution $p_x(x)$ in $[x_1,x_2]$ (including $\pm\infty$) based on another sequence r_k with the known distribution $p_r(r)$, is a common problem in simulations, i.e. a non-unique corresponding transformation $x_k=F(r_k)$ is to be searched for. Generally $p_r(r)$ is the standard equidistribution in [0,1] with $p_r(r)=1$; we will treat only this special case here, but the generalization is straightforward.

The number of events with $r_k \le r'$ has to be identical to the number of events with $x_k \le x' = F(r')$, i.e. one has (see also Fig. B1)

$$H(x') \;=\; \int_{x_1}^{x'} p_x(x)\; dx \;=\; \int_{r_1}^{r'} p_r(r)\; dr \;=\; r' - r_1 \;=\; r' .$$

Since probability densities are always larger/equal to zero, H is monotonic and can hence be inverted[9] and one concludes immediately that F is the inverse function of H, i.e.

$$F(r) \;=\; H^{-1}(r) \quad \Rightarrow \quad x_k \;=\; H^{-1}(r_k)$$

In principal this method always works but if H(x), the integral over $p_x(x)$, or its inversion are not simple, necessary for fast execution, other methods might be more efficient. This is the case for a Gaussian probability distribution for unit σ with $p_x(x)$ proportional to $\exp(-x^2/2)$. But there exists another algorithm[10], inspired from the determination of the definite integral over the Gaussian function from $-\infty$ to $+\infty$: the square of the desired integral is expressed by a 2-dimensional integral that can be transformed to polar co-ordinates with 'easy' integrals.

In fact, let us imagine that we can generate a 2-dimensional probability density distribution in $0 \le \varphi \le 2\pi$ and $0 \le \rho < \infty$ as

(B1) $$p_{\rho,\varphi}(\rho,\varphi)\cdot d\rho \cdot d\varphi \quad \propto \quad \rho \cdot \exp(-\rho^2/2)\cdot d\rho \cdot d\varphi$$

This distribution in polar co-ordinates can also be expressed in Cartesian co-ordinates by

$$x \;=\; \rho\cdot\cos(\varphi); \quad y \;=\; \rho\cdot\sin(\varphi); \quad dx\cdot dy \;=\; \rho\cdot d\rho\cdot d\varphi$$

yielding

$$p_{x,y}(x,y)\cdot dx\cdot dy \;\propto\; \exp(-(x^2+y^2)/2)\cdot dx\cdot dy \;=\; \exp(-x^2/2)\cdot\exp(-y^2/2)\cdot dx\cdot dy$$

Now one need only look for the distribution in x with y becoming irrelevant, i.e. one has to integrate over the whole range of y yielding the desired relation

$$p_x(x)\cdot dx \;=\; dx\cdot\int_{-\infty}^{+\infty} p_{x,y}(x,y)\cdot dy \;\propto\; dx\cdot\exp(-x^2/2)\cdot\int_{-\infty}^{+\infty}\exp(-y^2/2)\cdot dy \;\propto\; dx\cdot\exp(-x^2/2)$$

i.e. once the 2-dimensional distribution (B1) can be generated, the Gaussian distribution $p_x(x)$ can be obtained by neglecting y and only considering x. Evidently (B1) as a 2-dimensional distribution needs two *statistically independent* random numbers r_a and r_b per data x.

To obtain data distributed as (B1) one can use the standard method described above. One splits (B1) into two independent factors $p_\varphi(\varphi)d\varphi$ and $p_\rho(\rho)d\rho$ and applies the method for each of them, i.e.

$$p_\varphi(\varphi)\cdot d\varphi \;\propto\; 1\cdot d\varphi$$

$$p_\rho(\rho)\cdot d\rho \;\propto\; \rho\cdot\exp(-\rho^2/2)\cdot d\rho$$

The absolute normalization constants have to be determined by the condition that the total probability for any event – i.e. the integrals over $p_\rho(\rho)$ and $p_\varphi(\varphi)$ for the corresponding whole range – has to be one. This yields finally

$$p_\varphi(\varphi)\cdot d\varphi \;=\; d\varphi/(2\pi) \quad\Rightarrow\quad H_\varphi(\varphi') \;=\; \int_0^{\varphi'} d\varphi/(2\pi) \;=\; \varphi'/(2\pi)$$

$$p_\rho(\rho)\cdot d\rho \;=\; \rho\cdot\exp(-\rho^2/2)\cdot d\rho \quad\Rightarrow\quad H_\rho(\rho') \;=\; \int_0^{\rho'}\rho\cdot\exp(-\rho^2/2)\cdot d\rho \;=\; 1-\exp(-\rho'^2/2)$$

Inverting H yields F with

$$\varphi_k \;=\; 2\pi\cdot r_{a,k} \qquad \text{and} \qquad \rho_k \;=\; \sqrt{-2\cdot\ln(1-r'_{b,k})} \;\rightarrow\; \sqrt{-2\cdot\ln(r_{b,k})}$$

In the latter expression we can write $r_{b,k}$ instead of $1-r'_{b,k}$ when considering that for a random equi-distribution r' in [0,1], r=1-r' is also an equivalent random distribution with the same specifications. This means then for x (and y) in Cartesian co-ordinates

$$x_k \;=\; \cos(2\pi\; r_{2k})\;\sqrt{-2\cdot\ln(r_{2k+1})}\,; \qquad y_k \;=\; \sin(2\pi\; r_{2k})\;\sqrt{-2\cdot\ln(r_{2k+1})}$$

---

[9] at least piecewise for regions where $p_x>0$; macroscopic regions $p_x=0$ lead to a discontinuous inverse H^{-1}

[10] also fast approximations for the inverted Gauss integral are applied.

where we have assumed, as is usually done, that $r_a=r_{2k}$ and $r_b=r_{2k+1}$ are two sequential calls for the same standard pseudo-ransom generator; evidently the latter has to produce statistically independent sequences, hence should be a high-quality generator such as the Mersenne Twister [4].

Here x and y both have Gaussian distributions, hence both sequences might be used. However, one should not be tempted (for efficiency reasons) to use both as two Gaussian variables as they are correlated, one series has to be 'thrown away'.

For primary complex noise, as used in this paper, x and y are just the real and imaginary part for further transformations and the final real part is the true output.

To avoid numerical problems in the logarithm if r would become (very close to) zero, one should replace

$$\sqrt{-2\cdot\ln(r)} \quad \rightarrow \quad \sqrt{-2\cdot\ln(\varepsilon + (1-\varepsilon)\cdot r)}$$

with a well-chosen very small constant ε

Acknowledgement

The author would like to thank Elena Shaposhnikova, Trevor Linnecar and Thomas Bohl for helpful discussions and the latter for supplying the SPS cycle data as (E, V_{200}, V_{800}) as well as Urs Wehrle for his contributions to the AWG tests and the LabView® interface.

References

[1] J. Tückmantel, 'NoisySync', a simulation program for RF noise in synchrotrons, CERN, unpublished

[2] T. Bohl et al., Controlled Longitudinal Emittance Blow-up in the SPS as LHC Test-Bed and Injector, CERN AB-Note-2003-84 MD

[3] Urs Wehrle et al., priv. Comm.

[4] Makoto Matsumoto and Takuji Nishimura, Copyright (C) 1997 - 2002, all rights reserved.

http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html

email: m-mat @ math.sci.hiroshima-u.ac.jp (remove 'anti-spam' space)

[5] J. Tückmantel, Simulation of LHC Bunches under Influence of 50-Hz-multiple Lines on the Cavity Field, LHC Project Note-404 (2007)