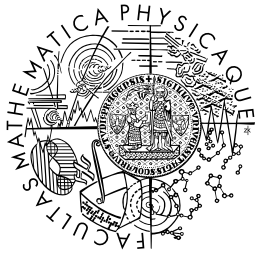

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Zuzana Gruberová

Simulation of noise in the Belle II strip vertex detector

Institute of Particle and Nuclear Physics

Supervisor of the bachelor thesis: RNDr. Peter Kvasnička
Study programme: Physics
Study branch: General Physics

Prague 2019

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

I wish to express my sincere thanks to my supervisor Peter Kvasnička for guiding my work, for his patience and support during our discussions, and for providing me with a valuable basis for my future projects. I would also like to thank Zdeněk Doležal for introducing me to the Belle II experiment and the Belle II Prague group.

I take this opportunity to also thank my family for their patience and support during my studies.

Title: Simulation of noise in the Belle II strip vertex detector

Author: Zuzana Gruberová

Institute: Institute of Particle and Nuclear Physics

Supervisor: RNDr. Peter Kvasnička, Institute of Particle and Nuclear Physics

Abstract: This bachelor thesis describes the development of strip noise simulation for the Belle II strip vertex detector (SVD). The overview part of the thesis describes the Belle II experiment and its detector system. The next part summarizes the basic principles of semiconductor detectors with a focus on electronic noise. This section is followed by a more detailed description of Belle II SVD, its noise characteristics and simulation methods. The methods part introduces basic concepts of noise simulation using machine learning methods, in particular, artificial neural networks. The experimental part describes the development and implementation of a production noise generator, and discusses the performance, precision, and alternative solutions.

Keywords: particle physics, Belle II, strip detector, noise, simulation, neural network

Contents

List of Figures

List of Tables

Introduction

Particle physics studies the elementary particles and forces that constitute the world around us, their properties and interactions. Current theory describing elementary particles and fundamental forces - the Standard Model - has predicted various properties and relations that have been confirmed, but it still leaves many observed phenomena unexplained.

One of the currently studied problems is finding the fundamental symmetry, that is, what kind of transformation applied to our universe would preserve all of its observable properties. The Standard Model introduces three possible natural symmetries - charge conjugation symmetry, parity symmetry and time reversal symmetry - none of which holds alone in our present universe. The search for the true symmetry continued by studying CP symmetry, that is, the combination of charge conjuration and parity symmetries. This combined symmetry was also shown to be broken by the discovery of CP violations in K and B meson decays. The symmetry that is still believed to hold is the CPT symmetry.

CP violation is an important discovery that promises more than just surpassing a wrong description of nature. Our universe exhibits great asymetry in abundance of matter and anti-matter and CP violation has been believed to provide at least some insight into this phenomenon. Understanding CP asymetry will also contribute to our knowledge about weak interactions responsible for particle decays.

High energy physicists study particle interactions predominantly in particle accelerator experiments. Accelerated particles collide at high energies and produce new particles that can exhibit rare decays, including those that break CP symmetry.

Several particle physics experiments were designed to study CP violation. Belle II is one of such experiments, developed at High Energy Accelerator Research Organisation (KEK) in Tsukuba, Japan. Its predecessor experiment, Belle, contributed important discoveries, completing its success by confirmation of the CP asymmetries predicted in B meson decays. The Belle II upgrade is expected to reveal more properties of particle decays and CP violation.

Particles are observed via large systems of detectors. Different types of detectors are required to track or capture different particles and their different properties. Silicon detectors are a widely used type of particle detectors, Belle II uses pixel and strip silicon vertex detectors as components of its tracking and vertexing system.

For precise reconstruction of physical processes, all tracking data must be measured with high accuracy. The accuracy depends on detector resolution and is adversely affected by electronic noise and beam background radiation.

Both accelerators and detectors consume large amount of energy spent on their operation and cooling. Detectors are also being damaged due to extreme radiation and need to be replaced every few years. Therefore, high energy experiments are very expensive, each component has to be precisely designed and carefully

tested, and prior to the start of the experiment, the measurements are simulated to obtain expectations of detector performance and to generate data with which physics reconstruction can be validated.

The simulations have to reflect every aspect of the real experiment including adverse effects like beam background and noise signals. By identifying unwanted signals, their rates and magnitudes, we are able to extract physics data and estimate their accuracy.

Computer simulations of this kind are huge computations that require large computing resources and take a lot of time. It is therefore essential to have efficient simulation methods that provide high accuracy with reasonable computing resources.

The subject of this bachelor thesis is the simulation of electronic noise in strips of the Belle II vertex detector. Electronic noise signals appear in strips of the SVD at different times and with different amplitudes. To simulate the noise, we have to be able to generate signals with rates and amplitudes governed by proper probability distributions. Our approach to this problem is to develop an artificial neural network with uniform random inputs to generate noise signal samples.

The review part of the thesis contains an overview of the Belle II experiment and its detector system and a description of semiconductor detectors with a focus on electronic noise origins and properties. It also describes simulation methods of the noise probability and introduces the concepts and properties of neural networks.

The central part describes the training data we used, the optimization of the neural network noise generator and also introduces some alternatives to neural network.

The attachments contain plots and tables illustrating the training process that did not fit well into the text flow of individual chapters.

1. The Belle II Experiment

Belle II is a particle physics experiment designed to study the properties of B mesons, heavy particles containing a bottom quark. On April 25, 2018 Belle II observed its first collisions at SuperKEKB, an e^+e^- asymmetric energy collider at the High Energy Accelerator Research Organization (KEK) in Tsukuba, Japan. Belle II is an upgrade of the previous programme at KEK - the Belle experiment.

This chapter briefly overviews the Belle II experiment and its detector systems. The last section of this chapter describes the beam background at SuperKEKB.

1.1 The Belle Experiment

The Belle experiment ran at the KEKB (KEK B-factory) accelerator between 1999 and 2010. Belle was designed for observation of CP violation in the B system by precisely measuring the differences between particles and antiparticles in certain decay channels of B mesons. Belle was able to collect 1000 fb^{-1} of data at various Υ resonances, setting a world record in integrated luminosity. Most of the luminosity was recorded near the $\Upsilon(4S)$ resonance, which is the optimal center of mass (CM) energy for the production of $B\bar{B}$ pairs used in B physics analysis.

CP asymmetries in B decays were predicted by the theoretical proposal of Kobayashi and Masakawa, who were awarded the Nobel Prize in Physics in 2008. Apart from CP violation measurements, Belle contributed several important discoveries in charm physics, tau lepton physics, hadron spectroscopy, and two-photon physics. [1]

1.2 Belle II: the Belle upgrade

CP violation is believed to be one of the reasons of the dominance of matter over anti-mater in our universe. However, Belle measurements were not sufficient to explain the observed asymmetry quantitatively. To reach a deeper understanding of this phenomenon, the 3 km long KEKB accelerator was upgraded to SuperKEKB, and provided with a new, Belle II, detector.

SuperKEKB is expected to reach a 40 times higher instantaneous luminosity of $8 \times 10^{35}\text{cm}^{-2}\text{s}^{-1}$ and total integrated luminosity of 50ab^{-1} in 2020's. The explorations of the new B-factory into the New Physics beyond the Standard Model will complement the Large Hadron Collider (LHC) experiments at CERN. While the LHC provides TeV mass scale, Belle II will focus on high-precision measurements of rare decays and CP violations at even higher mass scales through the effects of new particles in higher order processes.

Reaching higher luminosity requires both increasing the beam current and better focusing of the beams at the collision point. The energies of the electron and positron beams were changed to 7 and 4 GeV, respectively, the beam sizes at the

point of interaction were squeezed down to nanometer level.

The main concept of the new detector remains the same as in the Belle experiment. The detector is cylindrical with asymmetry respecting the forward direction of the interaction products (direction of the electron beam). Belle's silicon strip vertex detector is replaced by two layers of DEPFET pixel sensors and 4 layers of strip sensors to provide better tracking resolution. The new vertex detector of Belle II will cover a larger solid angle than in Belle. A completely new particle identification devices in the barrel and endcap regions are equipped with very fast read-out electronics, offering excellent separations of pions and kaons. The new data-acquisition system was designed to cope with considerably higher event rates. The electronics of the electromagnetic calorimeter will reduce the noise pile up, which is important for missing energy studies. [2] [3] [4]

1.3 Belle II Detector

SuperKEKB collides electrons with positrons, where positrons circulate in the Low Energy Ring (LER) of the accelerator at the energy of 4 GeV and electrons travel the opposite direction in the High Energy Ring (HER) at 7 GeV. When two bunches collide, the centre of mass moves in the direction of the electron beam. Therefore, most of the interaction products are detected in the forward direction and thus the layout of the detector is asymetric. 3D model of the Belle II detector is shown in Fig. 1.1.

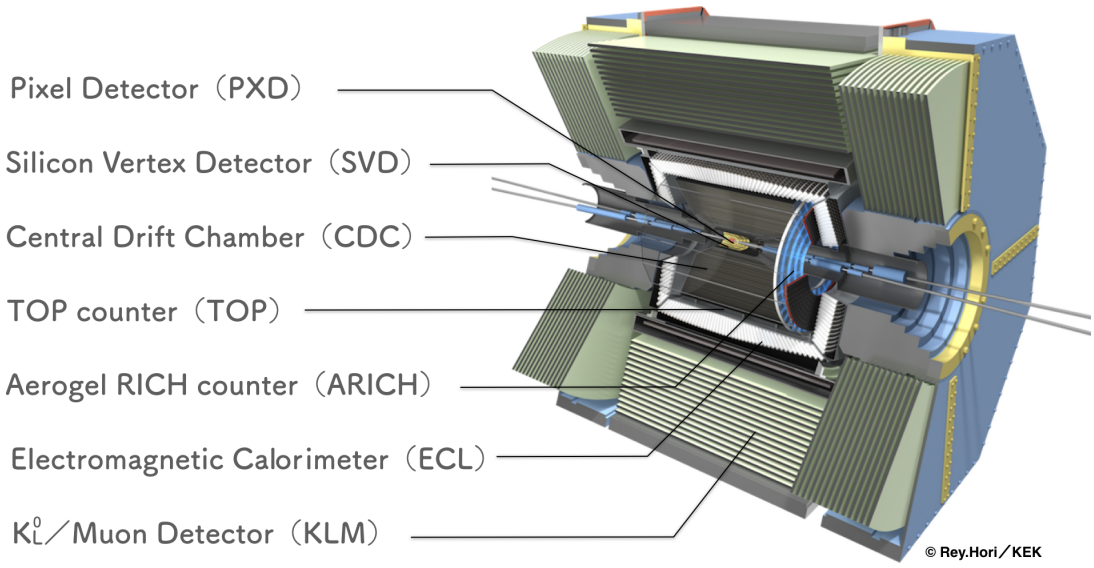

Figure 1.1: *3D model of the Belle II detector.* The figure shows major Belle II subdetectors. The tracking system comprises the PXD, SVD, and CDC; TOP and ARICH are responsible for particle identification. Crystals of ECL detect neutral particles and measure the energy of electromagnetic particles. The KLM surrounds the whole system and detects muons and kaons. The forward side of Belle II is to the right in the picture. [5]

The innermost sensitive part of the Belle II detector is placed 14 mm from the interaction point, which allows detection of low transverse momentum particles (from p_T as low as 6 MeV). Because of the extreme luminosity, large beam background is expected, therefore, high-granularity Pixel Detectors (PXD) are required in the innermost layers. The PXD sensors, based on the DEPleted Field Effect Transistor (DEPFET) technology, surround the beam pipe at 14 and 22 mm radii.

Four layers of the Strip Vertex Detectors (SVD) are placed between 38 and 140 mm from the beampipe. The SVD is built from double-sided silicon strip sensors. The barrel part consists of rectangular sensors in a windmill structure, the forward region is covered by trapezoidal-shaped sensors slated to form a lantern-shaped cap.

The Central Drift Chamber (CDC) is the largest tracking detector. The CDC occupies the volume between 160 and 1130 mm from the beampipe and its length is 2.4 m. It provides high precision track and momentum measurements that, combined with the data from the PXD and SVD detectors, allows precise reconstruction of charged particle tracks. CDC is also able to measure particles' energy losses and therefore contributes to particle identification. The CDC consists of sense and field wires parallel to the beam direction. The sense wires are made of tungsten and collect the signal from passing particles. The field wires made of aluminium surround the sense wires and create an accelerating electric field. The whole chamber is filled with He-H_2C_6 gas.

The Particle identification (PID) subsystem comprises the Time Of Propagation counters (TOP) in the barrel part and Aerogel Ring-Imaging Cherenkov detector counters (ARICH) covering the endcap. Cherenkov photons are created in aerogel inside the TOP by incoming charged particle and are guided into the quartz radiator and subsequently detected by photomultipliers at the end of crystal bars. Two spacial coordinates and very precise timing allow to reconstruct the Cherenkov ring image with required resolution. The ARICH counter is formed by aerogel layer, where the Cherenkov photons are created, an expansion volume to allow photons to form rings on the detector surface, and an array of position-sensitive photon detectors. These two particle identification detectors are designed to distinguish between kaons and pions and low energy pions, muons and electrons.

The Electromagnetic Calorimeter (ECL) is used to precisely measure the energy of electromagnetic particles, detect neutral particles and measure the luminosity. It consists of a 3 m long barrel section and annular endcaps, formed by CsI(Tl) crystals. Energy deposited by incoming particles is converted into photons and collected by photodiodes at the end of each crystal. The ECL also plays an important role in the triggering system.

The outermost detector is the K_L and muon detector (KLM), consisting of alternating layers of iron plates and sensitive detector elements and is located outside the superconducting solenoid. Charged particles are detected in the KLM by glass electrode Resistive Plate Chambers (RPC), which are interleaved between iron plates. RPCs collect the products of particle showers created in iron by muons and kaons.

Because of the large data flows required for the exceptionally high precision mea-

surements, fast readout electronics from every part of the Belle II detector is necessary. With the extreme luminosity, the trigger system needs to distinguish interesting physics events from among a huge number of other physical processes. The requirements for the trigger is high efficiency for hadronic events, maximum average rate of 30 kHz, timing precision better than 10 ns, and minimum two-event separation of 200 ns. Outputs from five sub-trigger systems are sent to the Global Decision Logic (GDL) system, where the final decision is made. There are several new information paths that were not present in the previous Belle trigger system. Data from the Data Acquisition Systems of subdetectors are merged, formatted to ROOT files and dispatched to server farms all over the world. [5]

1.4 Beam Background at SuperKEKB

The high design luminosity of the SuperKEKB result in challenging levels of beam-induced backgrounds around the interaction area. Proper study of these backgrounds is critical to the success of the Belle II experiment.

Five main beam background sources are present at SuperKEKB.

- The Touschek effect is a scattering process inside the bunch where, due to Coulomb scattering, energies of individual particles depart from the nominal energy of the bunch. This effect is enhanced because of the high compression of bunches in the nano-beam scheme.
- Beam-gas scattering is the scattering of beam particles on residual gas molecules in the beam pipe. It can occur either via Coulomb scattering changing the direction of the beam particle, or via bremsstrahlung scattering, which reduces its energy. The rate of beam-gas scattering is proportional to the beam current, which is nearly two times higher than at KEKB, and to vacuum pressure, which is the same.
- Synchrotron radiation (SR), emitted from the beam. SR power is proportional to beam energy squared and magnetic field strength squared; therefore, most of this background originates in the HER.
- Radiative Bhabha scattering decreases both electron and positron energies. The radiative Bhabha process produces photons which interact with the iron of the accelerator magnets and create gamma rays and neutrons. The number of these neutrons is proportional to luminosity. Neutrons create the main background of KLM measurements, low energy gamma rays are a significant source of background in the CDC.
- Low momentum e^-e^+ pairs produced via the two-photon process $e^-e^+ \to e^-e^-e^+e^+$ are an important luminosity-related background radiation. These pairs can spiral around the magnetic field lines and leave multiple hits in the inner (vertex) detectors. In addition, the primary particles that lose large amount of energy or scatter at large angles can be lost inside the detector.

Injection background appears when a charge is injected to the accelerator ring: circulating bunches get perturbed and a higher background rate is observed for a few milliseconds. A trigger veto is applied after each injection to avoid PXD readout saturation.

Background measurements at SuperKEKB are performed by dedicated beam background detectors, collectively known as the BEAST II subsystem. It consists of eight detector systems of different types and unique purposes, totalling in 116 sensors. Detailed description of the beam background measurement can be found in [6].

Beam background is generally detrimental to physics reconstruction - background signals compete for detection channels with signals from the physics processes we wish to study. The third player in the game is electronic noise - signals from the environment or from within the detection systems that are erroneously interpreted as particle signals. Both beam background and electronic noise limit the precision of physics measurements, and have to be studied thoroughly so that we can quantify, understand, and possibly eliminate their effect on physics studies.[7]

2. Strip Vertex Detector

Our work relates to the Belle II Strip Vertex Detector (SVD).

Strip detectors are semiconductor detectors with a finely spaced linear electrode structures - strips - that enables them to precisely sense positions of particle track intersections with the detector plane. We want to measure track points as close to the beam pipe as possible, because rare particles decay close to the interaction point and we need to detect the time and position of particle decays very accurately to be able to track decay products, which are detected and identified in further layers of the detector system, back to this point called the interaction vertex. Knowing what particles originated at a vertex, we can identify the type of interaction and subsequently the decaying particle (or particles).

The following sections describe basic principles of semiconductor detectors, focus on electronic noise of the measurements and add more details about the Belle II SVD.

2.1 Semiconductor Detectors

2.1.1 Sensor structure

Semiconductor detectors are basically solid-state ionization chambers. The simplest configuration consists of a volume of ionizable medium between a pair of electrodes with applied voltage, Fig. 2.1 (a). A passing particle ionizes the medium, creates charge pairs, which move under the influence of the applied field and induce an electrical current pulse in an external circuit.

To create position sensitivity, the electrodes of the sensor have to be segmented. Angled tracks will deposit charge on several strips and we can use the signals to get a more precise position estimate by interpolation. Subdividing both electrodes into strips to form an orthogonal lattice provides two-dimensional imaging, as shown in Fig. 2.1 (b). However, it is difficult to match strip signals to $x-$ and $y-$coordinates if several particles have passed the detector. To eliminate ghost hits, additional information is required. In some situations, strips subtending a small angle are used instead of perpendicular strips°. [8]

2.1.2 Sensor physics

Average signal charge produced by a particle passing through a detector is

$$Q_s = \frac{E}{E_i} e, \tag{2.1}$$

where E is the absorbed energy, E_i is the energy required to form a charge pair, and e is the electron charge. The absorbed energy must exceed the bandgap of the solid material. In Si the gap energy is 1.12 eV, so photons with wavelengths greater than 1.1 μm can not be detected. E_i for Si is 3.6 eV. A charged particle

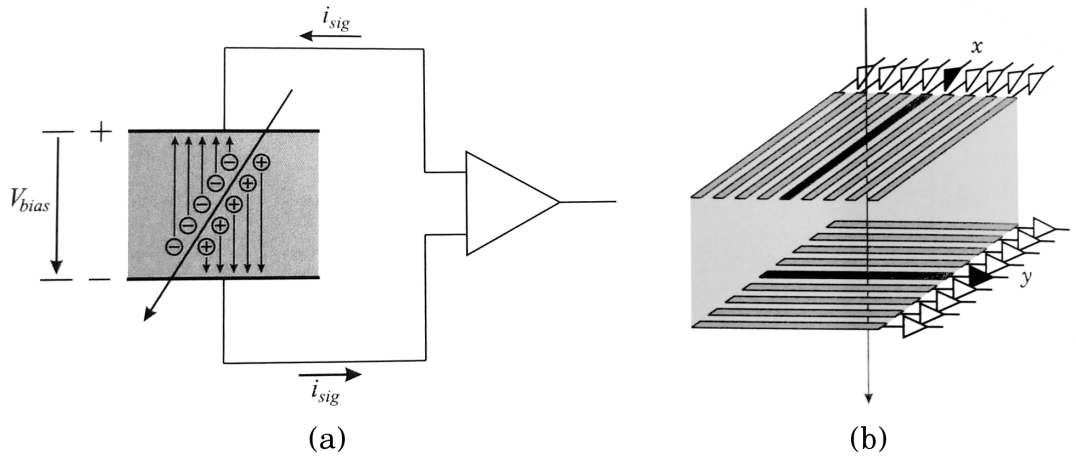

Figure 2.1: *Semiconductor detector principle.* A semiconductor detector is an ionization chamber with semiconductor as its absorber (a). A charged particle passing through the detector creates charge carriers, which move to the electrodes and induce electric current. Electrodes can be segmented into strips (b). Dividing electrodes on opposite sides of the detector into perpendicular strips provides two coordinates to determine the track position. [8]

traversing the sensor forms charge pairs along its track, the energy loss E increases with the sensor thickness. The velocity $v(x)$ of the charge carriers depends only on the local electric field $E(x)$

$$\vec{v}(x) = \mu\vec{E}(x), \tag{2.2}$$

where μ is the mobility of charge carriers. In Si the mobility is approximately 1350 V/cm·s^2 for electrons and 450 V/cm·s^2 for holes.

Most of detector volume is occupied by a depleted region around a *pn*-junction of differently doped Si crystals. The volume is reverse-biased so that the electrons in the *n*-type region and the holes in the *p*-type region are drawn away from the junction, creating a depleted area which behaves as an insulator in a linear electric field.

Fig. 2.2 shows the cross-section of a typical detector diode. The detector *pn*-junction is in the middle, similarly doped regions are to the left and right, forming a guard ring, which insulates the diode from the edge of the wafer. The guard ring captures edge currents and forms a well-defined electrical boundary for the diode. Electrical contact is provided by metallization layers, typically aluminium. The intermediate Si surface is protected by a layer of SiO_2.

Reverse bias voltage V_b yields a depleted region of width

$$w_d = \sqrt{\frac{2\varepsilon(V_b + V_{bi})}{Ne}}, \tag{2.3}$$

where V_{bi} is the built-in junction potential (typically 0.5 V), ε is the dielectric constant ($11.9\varepsilon_0$ for Si) and N is the dopant concentration in the bulk. The depleted junction forms a capacitor, for $V_b >> V_{bi}$

$$C \propto \frac{1}{\sqrt{V_b}}. \tag{2.4}$$

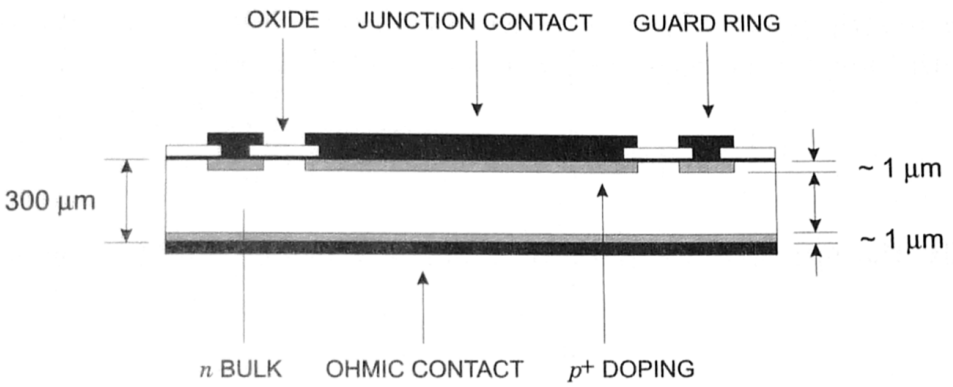

Figure 2.2: *A semiconductor detector diode.* The diode consists of a *pn*-junction between the electrodes. The guard ring around the diode captures edge currents and create a well-defined electrical boundary. The Si surface is protected by a layer of SiO_2. [8]

For a 100 μm Si diode it is about 1 pF/mm^2. In strip detectors, the fringing capacitance between neighbouring electrodes usually dominates, typical value is about 1 pF/cm.

The collection time is the time required for a carrier to traverse the full detector thickness. It can be estimated by using the average field $\bar{E} = V/d$ (V is the volume of the detector, d is the distance between electrodes)

$$t_c \approx \frac{d}{v} = \frac{d^2}{\mu V}. \tag{2.5}$$

The minimum signal which can be detected and the precision of the amplitude measurement are limited by fluctuations.

The relative resolution of the absorbed energy equals the relative resolution of the average number of signal quanta

$$\frac{\Delta E}{E} = \frac{\Delta N}{N} = \frac{\sqrt{FN}}{N} = \sqrt{\frac{FE_i}{E}}, \tag{2.6}$$

where E_i is the excitation energy and F is the Fano factor, which reflects the reduction of the error relative to a Poissonian error thanks to collective excitation mechanisms.

The position resolution is determined, to the first order, by the electrode geometry. Strips are usually $8-12\mu$m wide, placed on a pitch of $25-50\mu$m, and $6-12$ cm long. The root mean square (rms) resolution is proportional to the strip pitch p

$$\sigma^2 = \frac{p^2}{12}. \tag{2.7}$$

A better approximation counts in the thermal diffusion. Using the average field $\bar{E}$, the rms dispersion due to thermal diffusion is

$$\sigma_y \approx \sqrt{2\frac{kT}{e}\frac{d^2}{V_b}}, \tag{2.8}$$

and is the same for both electrons and holes. For $d = 300\mu$m, $T = 300$ K and $V_b = 100$ V it is approximately 7 μm.[8]

2.1.3 Electronic noise

Electronic noise imposes a lower bound on the detectable signal and determines the precision of signal level measurement. Noise signals are detected as random oscillations when no signal is present. Fig 2.3 (a) shows an example of noise signals. When an experimental signal arrives, it is added to the noise and we measure their superposition. Fig 2.3 (b) shows a noiseless signal profile, whereas (c) shows the same signal distorted by noise (pure signal is superimposed for comparison). Signals of this amplitude are invisible against this high noise background.

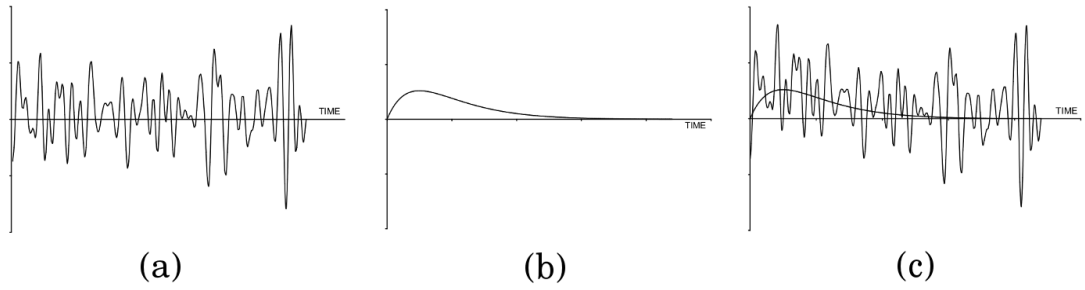

Figure 2.3: *Noise example.* (a) shows an example of electronic noise, (b) represents a noiseless signal profile, (c) shows the same signal distorted with noise, the noiseless signal is superimposed for comparison. [8]

In an optimized system, the time scale of the fluctuations is comparable to the signal peak time. This holds for signals with sufficiently large amplitude, more precisely, with sufficiently high signal-to-noise S/N ratio. Fig. 2.4 displays four measurements of the same signal taken at four different times with noise background at $S/N = 20$. Again, the noiseless signal is superimposed. We can recognize the signal peak; it is however clear that the noise affects both the observed arrival time and peak amplitude of the signal.

The current i induced by n carriers with charge e and velocity v moving through a sample of length l, is

$$i = \frac{nev}{l}, \tag{2.9}$$

at the ends of the sample. Fluctuation of this current is given by the total differential

$$\langle di \rangle^2 = \left(\frac{ne}{l} \langle dv \rangle \right)^2 + \left(\frac{ev}{l} \langle dn \rangle \right)^2, \tag{2.10}$$

where the two terms are statistically uncorrelated. Two mechanisms contribute to the total noise: velocity fluctuations (e.g. thermal noise) and number fluctuations (e.g. shot noise). Both of these mechanisms are "white" noise sources, which means that power is constant with respect to spectral density, $\frac{dP_{noise}}{df} = const.$

The noise voltage originates mainly in the amplifier. The noise current flows through the detector's capacitance and forms noise voltage that increases with

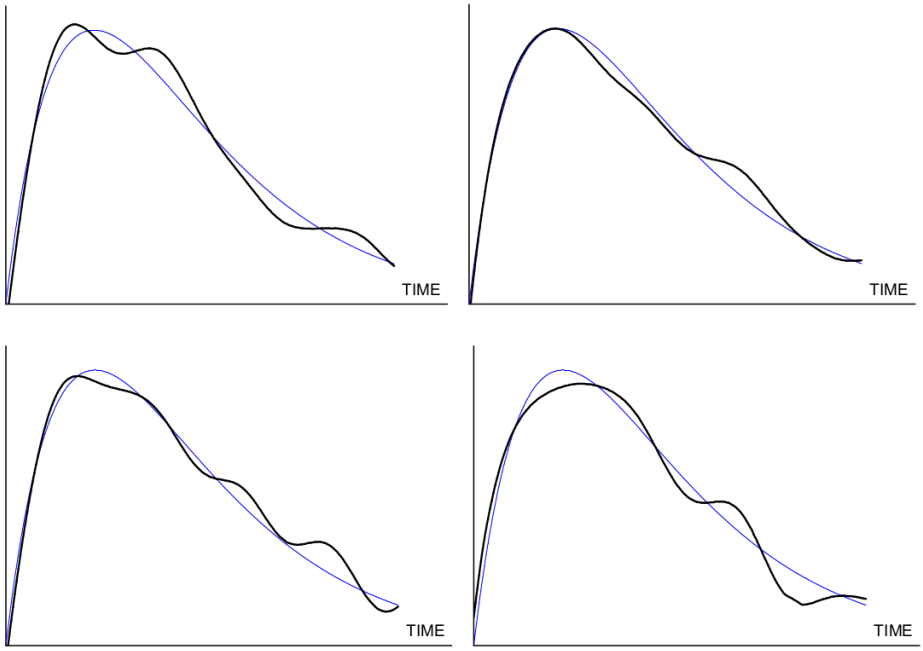

Figure 2.4: *Noise effects on measurement.* This figure shows four pictures of the same signal profile taken at different times. By comparison with the noiseless signal, we see that the noise changes both arrival time and peak amplitude of the original signal. [8]

decreasing frequency. The peak amplifier signal is inversely proportional to the total capacitance C at the input.

In general, optimum S/N is independent of what kind of signal is detected (voltage, current, or charge), but practical considerations can favour one configuration over the others. More details can be found in [8].

2.2 Belle II SVD

The main purpose of the Belle II SVD (Fig 2.5), together with the PXD and CDC is to measure the two B decay vertices. It also measures vertex information in other decay channels involving D and τ decays. The Belle II SVD provides data to extrapolate the tracks reconstructed in the CDC to the PXD with high efficiency. Together with PXD, it is able to reconstruct tracks with low transversal momentum which do not leave any hits in the CDC.

It is composed of four layer and the polar angular acceptance ranges from 17° to 150°, which is almost double that of the SVD in Belle.

The Bell II SVD operates efficiently and with low dead-time in the high beam background trigger rate environment of SuperKEKB. The background rate may be 30 times higher than in KEKB, and the expected maximum average trigger rate is 30 kHz.

To suppress beam background, a fast readout chip is indispensable. For this

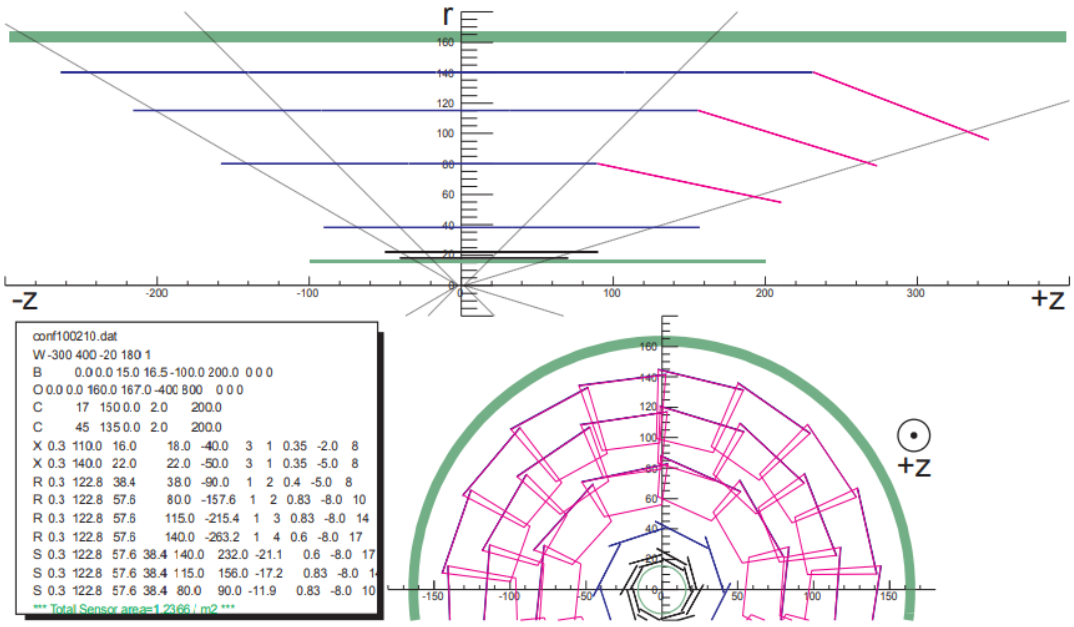

Figure 2.5: *The Belle II SVD.* The Belle II SVD design consists of four double-sided silicon strip layers (above the two inner layers of PXD) with slanted sensors in the forward region. All dimensions are in mm. [4]

purpose, the APV25 chip, originally developed for the CMS silicon tracker, was chosen. The APV25 consists of 128 channels of low-noise preamplifiers followed by a shaper stage. Each channel has a 192-cell deep analogue pipeline with FIFO index that can label up to 32 cells pending for subsequent readout, an analogue pulse-shaper processor and an analogue multiplex for the output. For more details we refer to [4].

3. Neural Networks

To reproduce strip noise signals in detector simulations, we need to be able to generate noise at the same rates and with the same time and amplitude distribution in the same way they occur in real data. For the production of random noise signals, arriving at certain times with certain amplitude, we decided to use the 2D distribution function inversion described at the end of the previous chapter and a neural network. Reasons for this approach and its alternatives will be discussed in the next chapter. The following sections give an introduction to the basic concepts of neural networks.

3.1 Network Structure

An artificial neural network is a computing structure which can be trained to learn complex functional relationships by analyzing a large number of examples without being given any task-specific algorithms or rules. The training examples are basically pairs of network inputs and correct outputs it is expected to provide. The network needs to identify the characteristic aspects of the input data with same output and develop the right connections in order to be able to predict the output for an input it has never seen before. Thanks to these newly developed connections and independence of any specific guidelines, artificial neural networks can resemble biological neural networks in animal brains, which create connections between neurons during learning.

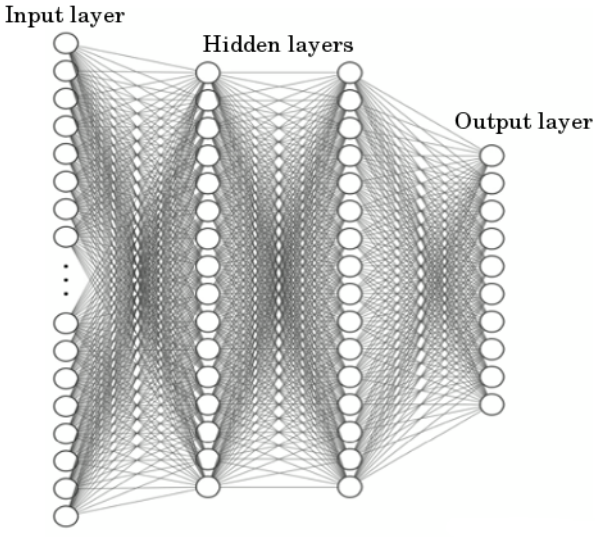

Figure 3.1: *A MLP neural network.* A Multi-Layer Perceptron neural network consists of an input layer, output layer, and several hidden layers in between. Neurons in each layer are connected with neurons in the next layer and are responsible for activation of the next layer. Activation in the input layer determines the activation in the output layer, the hidden layers help to recognize patterns in inputs and allow the network to learn more complex tasks. [9] (edited)

The basic structure of a neural network is the Multi-Layer Perceptron (MLP) layout (Fig. 3.1). Input signal can be represented as a layer of nodes, in analogy to neural system cells called neurons. The input layer usually consists of a large number of neurons, for example in image recognition, the neural network can be given one neuron for each pixel of the picture. The neurons are given an input quantity, for example some measure of pixel brightness. Similar set of quantities will characterize every input the neural network can be given. The layer representing outputs is usually smaller, sometimes only binary output is required (for example in classification problems).

Apart from these two layers, a neural network contains at least one hidden layer. These extra neurons allow the system to develop more complexity of interconnection between inputs and outputs. Every neuron in the input layer is connected with each of the neurons in the first hidden layer, these hidden neurons are connected to the following layer neurons and so on. The activation in the first layer determines the activation in the next layer via the developed connections. The hidden layers can encode features or structure identified during the learning process. Input parameters are analyzed and the information about recognized patterns is then put into the next layer neurons where they are analyzed again and processed further. For example, we can imagine it as recognizing lines and shapes in a picture, however, what the network really considers a characteristic structure in the data can be much more complex features. [9] [10]

3.2 Activation

A neural network receives some encoded data on its input neurons. Processing of the data consists of alternating computing and message-passing steps:

- In a computing step, each neuron receives its input, transforms it using its activation function, and outputs the result. For the network to be able to learn complex relationships, neuron activation has to be non-linear. Fig. 3.2 shows commonly used activation functions.
- In a message-passing step, output of each neuron in a layer, a_j^{i-1}, is redistributed to neurons in the following layer according to a matrix of weights $W_{j,k}^{i-1,i}$. Put simply, inputs of the next layer neurons are calculated from outputs of previous layer neurons by multiplication with a weight matrix.

With topology (number of layers, their sizes and activations) of a MLP determined, the network is parameterized by weight matrices between successive layers. Additionally, each layer has an adjustable bias b^i that adjusts the zero of neuron activation - it is added to the sum of inputs to the activation function. The whole formula of the k-th neuron activation is

$$a_k^{(i)} = f(b^i + \sum_l W_{lk}^{i-1,i} a_l^{(i-1)}). \tag{3.1}$$

or, in matrix form,

$$\mathbf{a}^i = f(\mathbf{W}^{i-1,i}\mathbf{a}^{i-1} + \mathbf{b}^i), \tag{3.2}$$

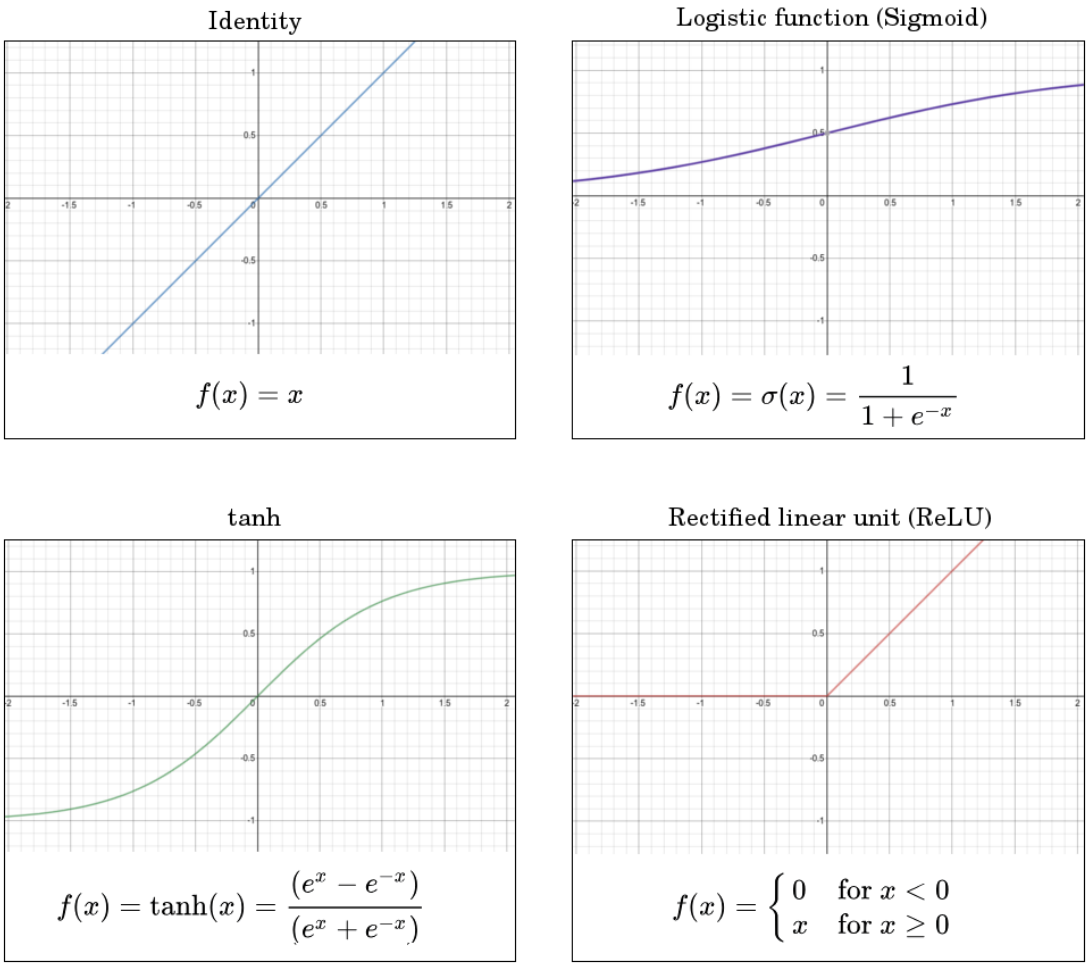

Figure 3.2: *Activation function examples.* Different applications require different activations. Simple functions as identity or ReLU take less time to compute and learn easier. When a restricted interval of values or less sensitivity to high input values is preferred, functions as logistic or tanh are used. Graphs were plotted in [11].

The whole neural network can be treated as a very complicated function receiving a number of input values, parameterized by typically hundreds of thousands weights and biases and giving the desired output. [9]

3.3 Training the neural network

Training is the process whereby the neural network determines its weights and parameters from training data by optimizing some measure of agreement between network outputs and known correct responses for given training inputs. At the beginning of a training, the weights and biases are usually chosen randomly. This untrained network analyzes the first dataset and produces predictions that are then compared to the correct outputs. The performance of the network can be evaluated by a loss function, measuring how far away network output is from the correct responses. This is usually the sum of squared errors for regression

problems and cross-entropy for classification. To train a better network we want to minimize this function by properly adjusting network weights and layer biases. From a random initial state, the network can move towards the minimal error state via computing the gradient of the cost function with respect to weights and biases. The gradient components adjust all the weights and biases towards the local minimum. After the next training step, the performance is better and when it reaches the optimum, the network in not able to improve any more.

The process of training is usually carried out using the backpropagation scheme. Comparing the results with the correct solution assigns the output neurons a correction proportional to the cost function gradient. The activation of the output neurons can not be changed directly, only via the biases and the weights that connect them with the previous layer. These changes then determine how the previous layer should perform in terms of activation and the process with gradient is applied recursively to the previous layer, from the output layer towards the input layer.

It takes a long time to calculate the gradient and we want to avoid ending in a local minimum. Therefore, stochastic gradient methods are typically used. This method divides the training data into smaller groups (batches) and computes the gradient just from this one batch. These steps are not precise but quicker and the network can perform effectively in certain cases. [12] [13]

4. Noise Simulation

We have a theoretical model of strip noise, and - at least in theory - we can use theoretical probability distribution(s) to generate random noise samples for strip detector simulation. However, practical considerations make the problem more challenging and impose further requirements on the desired noise generator.

In this chapter, we first describe the theoretical background of the SVD noise simulation methods and introduce the theoretical model. The following sections contain the description of noise generator implementations and their properties.

For a noise generator usable in production - that is, in real detector simulation - we need a sufficiently efficient implementation of the strip noise generator, that is, an implementation with reasonable memory and computing time footprint. We also need to take into account the specific parameterization of the generator. Importantly, the target implementation has to allow experimental data to be used for training or calibration of the noise generator.

We therefore start with the description of the exact implementation of the noise generator based on splines. We than explain why this is not suitable as a production noise generator and proceed to more efficient implementations based on machine learning methods. We explain pros and cons of individual methods, and explain the considerations that lead us in the search of the final implementation.

The Python code written for noise simulations is available at `https://github.com/zugru/SVDNoise`.

4.1 Noise in Belle II SVD

Noise in the SVD are the unwanted electronic signals that are indistinguishable from signals left by particles and pass the zero-suppression threshold. APV25 chips read out strip signals every 31.44 ns and store it in the pipeline. When the chip receives a trigger signal, it takes one sample recorded before trigger time and 5 samples after the trigger time and passes them further to the readout system. Strip signals are passed for data processing only if its amplitude exceeds the zero-suppression threshold, that is, when at least one of the six APV25 samples exceeds a fixed multiple (called zero-suppression threshold) of RMS strip noise, see Fig. 4.1.

To simulate strip noise, we need to know the probability that a noise signal passes the zero-suppression threshold. The probability depends on arrival time of the signal (t), signal amplitude (A), the zero-suppression threshold (T), the signal measurement noise (σ) and the shape of the pulse. For sufficiently high zero-suppression thresholds, we can model the generating noise signal as independent and identically distributed (iid) gaussian signals arriving at very small constant rate, so that the autocorrelation structure of the generating noise is not important. The gaussians are centered at zero and standard deviation is equal to RMS strip noise, which is regularly measured for each SVD strip. The noise expected in the measurement of APV samples is presumed to be gaussian and is a small fraction

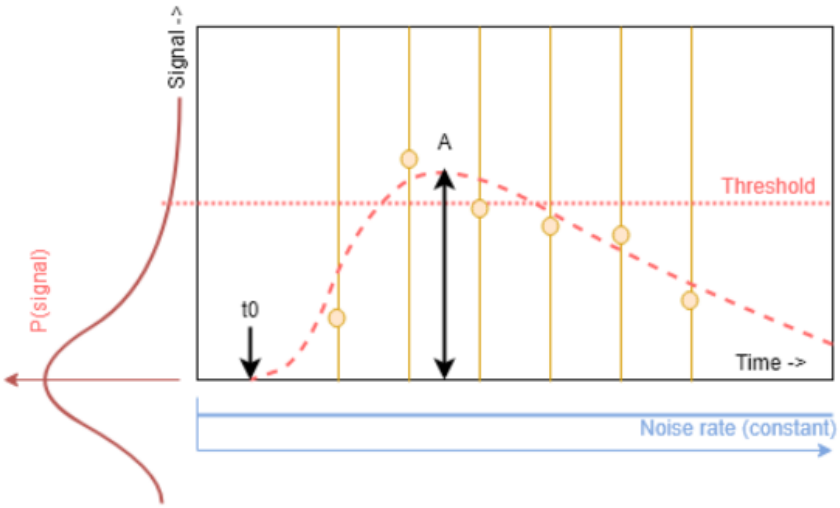

Figure 4.1: *Strip signal.* The strip signal passes the zero-suppression threshold whenever at least one of its APV25 samples is above threshold. [7]

of strip noise.

When a noise signal passes the zero-suppression threshold, it is passed on for processing. The mean number of noise signals collected during readout is called strip occupancy and it can be expressed as

$$n = \int_{-\infty}^{\infty} dt \int_{0}^{\infty} dA n(A,t) = \int_{-\infty}^{\infty} dt \int_{0}^{\infty} dA \lambda_0 P(A) P(s_i > T|A,t), \tag{4.1}$$

where λ_0 is the noise base rate, $P(A)$ is the gaussian distribution of the noise signal amplitude A and $P(s_i > T|A,t)$ is the probability that a signal with amplitude A arriving at time t will pass the threshold T with at least one APV25 sample s_i. For the simulation, we need to be able to generate random samples from the $n(A,t)$ 2D distribution.

We can calculate the probability of at least one sample passing the threshold T for a waveform with a given A and width w arriving at time t. From the strip noise distribution, we also know the probability of a noise signal with amplitude A. In simulation, we normalize the noise distribution to 1 and do a two-step sampling, first from the alternative (norm, 1-norm) distribution to decide if there is noise signal on the strip, and then from the normalized 2D distribution to generate the signal's amplitude and arrival time.

$$n(A,t) = \lambda_0 \lambda_{A,t} \Pi(s_i > T|A,t), \tag{4.2}$$

where $\lambda_0 \lambda_{A,t}$ together represent the $\sim e^{-CT^2}$ dependence of occupancy observed in the data [14]; the Π distribution reproduces the signal and time features of the noise.

To draw samples from a bivariate distribution, we need to know the correlation structure between A and t. Two samples from U(0,1) u, v are used to calculate t, A by inversion of the distribution function as follows:

$$t = F_t^{-1}(u), \tag{4.3}$$

$$v' = F^{-1}(v|u), A = F_A^{-1}(v'), \tag{4.4}$$

where $F_t(t), F_A(A)$ are marginal distribution functions and the conditional distribution function $F(v|u)$ - the "partial copula" - contains the correlation structure of A and t. [7]

4.2 Exact Generator

Using the noise probability distribution given by our theoretical model, we can directly construct a noise generator using numerical inversion of the 2D distribution function. The inversion is done using inverse interpolation of spline approximations to the distribution function, using the Python `NumPy` library [15]. The implementation is wrapped in the `NoiseGenerator` Python class. `NoiseGenerator` is initialized with $(t_0, amplitude)$ grid. Calling its `generate_pdf` method, providing values of threshold, sigma and width, creates structures needed for generation of random values:

- the method calculates the probability of a signal exceeding threshold on a user-defined grid of (t, A) values, then calculates the norm and normalizes the distribution, producing a probability density (pdf).
- integrates the pdf to get the cumulative distribution function (cdf),
- fits the cdf with a 2D spline,
- fits its margins with 1D splines,
- inverse-interpolates the t and A margins to invert their distribution functions,
- uses these to calculate the copula (uniform-margin cumulative distribution function),
- converts the copula to what we call "partial copula", a distribution which is pdf in the u (t quantile) direction and cdf in the v (A quantile) direction - that is the conditional distribution function used in random sampling from the distribution.

The inverted margin pdfs and the half-copula are then used to generate random samples from the original pdf using a call to the `NoiseGenerator` class's `random_transform` method. The method takes a vector of quantiles (that is, values between 0 and 1) u and v, and returns corresponding (t, A) pairs. The random generator functionality is easily achieved by providing vectors of uniformly distributed random values for u and v.

The `NoiseGenerator` class also provides some diagnostic plots - it plots the 2D probability density and marginal distributions (Fig. 4.2), 2D and marginal distributions of random samples (Fig. 4.3), and the half-copula distribution (Fig. 4.4)

used to calculate v' from u and v (see Eqn. (4.4)). The following figures were generated for $(T, \sigma, w) = (5.0, 0.1, 250)$, more examples can be found in Attachment 1.

`NoiseGenerator` is very precise and easy to use, and is a valuable tool to study the noise distribution and its parametric dependencies.

Nevertheless, it is unsuitable as a production noise sampler. The problem is that it has to calculate the complete pdf and do a complete 2D cdf inversion for each new (T, σ, w) triplet, which takes a few seconds. While the threshold T stays constant in most runs, σ and w are strip properties and vary over wide ranges of values. So, while we will usually be generating a few dozes random SVD signals per event, we will have to generate a complete pdf and do the cdf inversion for each of them separately.

It is also impractical to try to store the cdf inversions for a grid of (T, σ, w) triplets, since the structure would be quite large.

We therefore decided to find a better representation of the noise-generating structure using a machine-learning algorithm, and use the exact generator to produce enough training data.

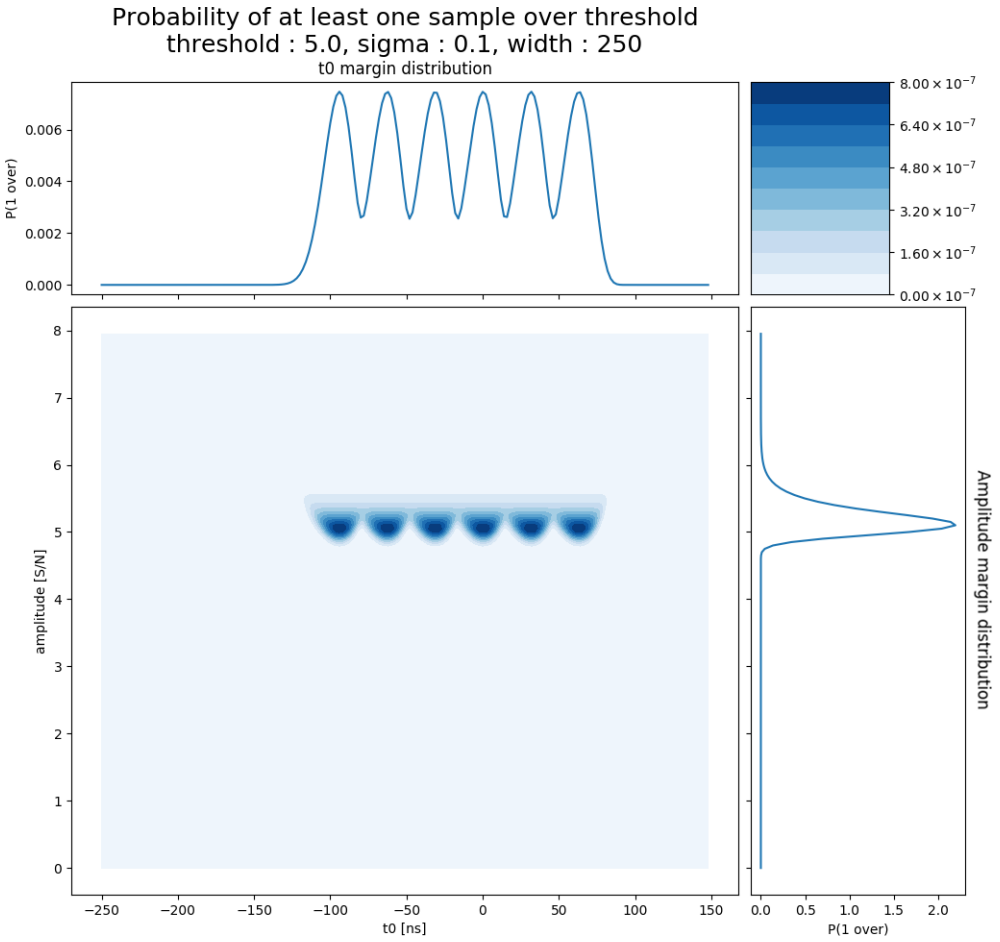

Figure 4.2: *2D probability density generated by* `NoiseGenerator`. The central plot shows the 2D probability density, amplitude margin is on the left, time margin is on top.

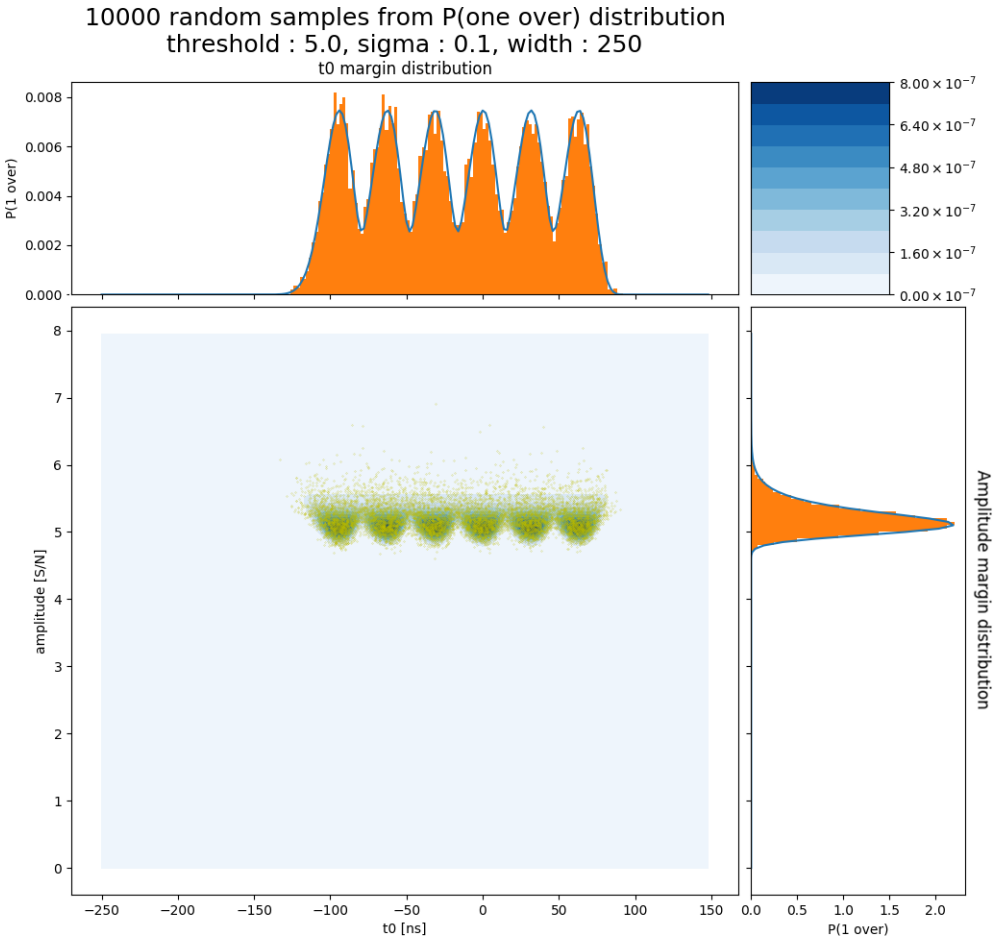

Figure 4.3: *2D probability density with random samples generated by* `NoiseGenerator`. The central plot shows the 2D probability density, amplitude margin is on the left side, time margin is on top. Orange bars in margin plots and yellow dots in the 2D plot are 10000 random samples.

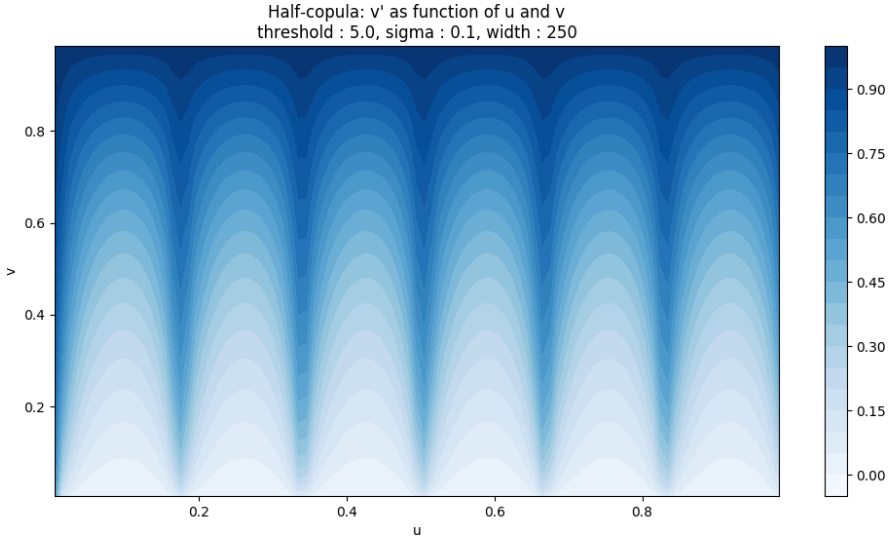

Figure 4.4: *Half-copula generated by* `NoiseGenerator`. This function is used to calculate v' from u and v.

4.3 Production Noise Generator

There are basically two possible routes to create a practical sampler:

- We can make a structure that learns the cdf inversions, $(u, v|T, \sigma, w) \rightarrow (t, A)$, such as a neural network or a boosted regression tree,
- or learn the probability distributions in a form that would make it easy to sample from, such as a gaussian mixture.

Along with signal sampling, we also have to calculate norms for each (T, σ, w) triplet. Here we have the same problem - we can easily calculate the norms, but it is the same computationally slow process used to generate the "exact" pdfs, so we have to learn the norms, too. This is easily done using standard (polynomial) linear regression.

4.3.1 Training data

Training data for signal sampling were generated using the `NoiseGenerator` Python class. For a typical training, we used 2 million samples with the following structure:

Input fields:

- T (threshold) random from a list [3, 4, ... 7]
- σ (signal measurement noise) random uniform from interval 0.1 to 1.0
- w (waveform width) random uniform from interval 200 to 360

(2000 triplets)

- u (time quantile) random uniform from interval 0 to 1
- v (amplitude quantile) random uniform from interval 0 to 1

(1000 pairs for each triplet)

Output fields, calculated for each set of input parameters:

- v' (corrected amplitude quantile)
- t (signal arrival time)
- A (amplitude)

Calculating the 2 million (t, A) pairs takes somewhat more than 8 minutes, which is fine for generating training data, but not for production use.

4.3.2 Training scheme

It took some time to design the training procedure so that it takes proper account of the additional (T, σ, w) parameterization. We decided to split the estimators only by threshold, and leave the (σ, w) parameterization inside. The reason is that in practical use, threshold is never changed within a run, and most runs are taken at the same threshold of 5.0. On the other hand, measurement noise and waveform width are strip properties varying over wide ranges and - for performance reasons - we want to sample random signals for them without switching estimators within an event.

This design eliminated an otherwise viable alternative approach of using estimators on a (T, σ, w) grid. With this method, we can train different networks for each desired threshold simultaneously.

4.3.3 Neural network sampler

We can train a neural network to generate samples from the right distribution. To create such network we used scikit-learn class `MLPRegressor` (scikit-learn is an open source Python tools package, available from [16]). The final code fits the neural network weights and biases to training data and returns the trained model which is then able to generate new random samples.

We trained separate networks to calculate signal arrival time t from (T, σ, w, u), and amplitude A from (T, σ, w, u, v). The reason for splitting the estimator into time and amplitude networks lies in different complexity of t and A: Even when properly scaled, t steals precision from A in a combined estimator. We split the 2 million training samples into 5 parts by discrete threshold values $(3, 4, ..., 7)$ and set apart 10% of the samples as test set.

We carried out the training of the 10 estimators in parallel and at first tried to determine the proper network topology and activation. By trial and error, and later by grid-searching the hyperparameter space, we arrived at the following settings:

time networks: $3 \times 100 \times 100 \times 1$ nodes, tanh activation

amplitude networks: $4 \times 250 \times 250 \times 1$ nodes, ReLU activation

Input nodes for time network represent (σ, w, u), for amplitude network it is (σ, w, u, v), T is a fixed parameter for each of the ten networks. Hidden layers enable the network to learn non-linear models but deeper networks are characteristic by non-convex loss functions with multiple local minima. Therefore, the resulting accuracy is sensitive to the random weight initialization and the network training can be non-reproducible. Time networks were difficult to train to enough accuracy. *tanh* activation makes the training longer, but in the end allows to reach the desired accuracy of below 0.15 ns. Amplitude networks train well with sufficiently large layers and the default ReLU activation, optimal accuracy is below 0.01.

Next we optimized hyperparameters of the training procedure using grid search to obtain highest network precision at reasonable training time. The grid searches were made using a single estimator. The results are shown in Tab 4.1.

Table 4.1: Parameters of the final time and amplitude networks

Parameter	Description	Default value	Optimized value for t-network	Optimized value for A-network
alpha	regularizes the solution by penalizing weights with large magnitudes	1.0e-4	1.0e-7	1.0e-7
tol	defined convergence criterion of the loss function	1.0e-4	1.0e-8	1.0e-8
batch_size	determines the size of training data chunks for stochastic gradient method	200	500	500
learning_rate_init	controls the steps of adjusting weights in backpropagation	1.0e-3	1.0e-4	3.0e-5
max_iter	determines how many times each data point will be used	200	1000	1000

The model is optimized by a stochastic gradient descent ("adam" optimizer) which is adequate for large datasets. In general, we switched off regularization (alpha) and slowed down the training by decreasing the initial learning rate, so that the networks have more time to organize itself before converging to a minimum. Other `MLPRegressor` parameters (not mentioned in the table) were left at their default values.

The following table shows χ^2 of the final networks. Attachment 2 contains some of the partial results recorded during optimization as well as some more detailed notes describing the process of optimization.

Table 4.2: χ^2 of the final set of networks

threshold	time χ^2	amplitude χ^2
3	0.058861	0.004075
4	0.062929	0.006370
5	0.078453	0.006543
6	0.081001	0.008162
7	0.081783	0.008672

Fig. 4.5 shows model 2D probability density and marginal distributions with random samples generated by the network, Fig. 4.6 shows contour lines of the half-copula and provides a sensitive comparison of probability distributions of the training data and samples generated by networks. We chose $(T, \sigma, w) = (5.0, 0.25, 250)$ as an example, plots for other parameter combinations can be found in Attachment 3.

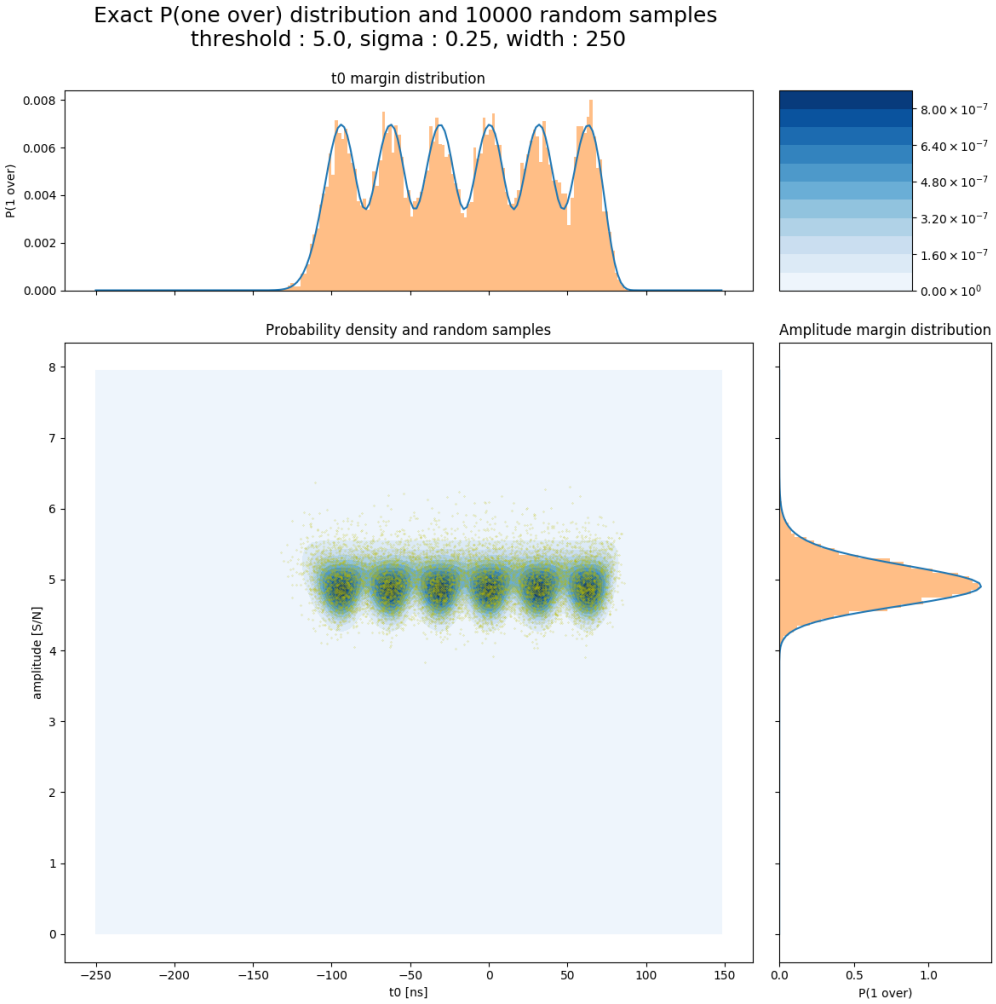

Figure 4.5: *2D probability density with random samples generated by neural networks.* The central plot shows 2D probability density, amplitude margin is on the left side, time margin is on the top. Orange bars in margin plots and yellow dots in the 2D plot are 10000 random samples, blue contours are the exact model.

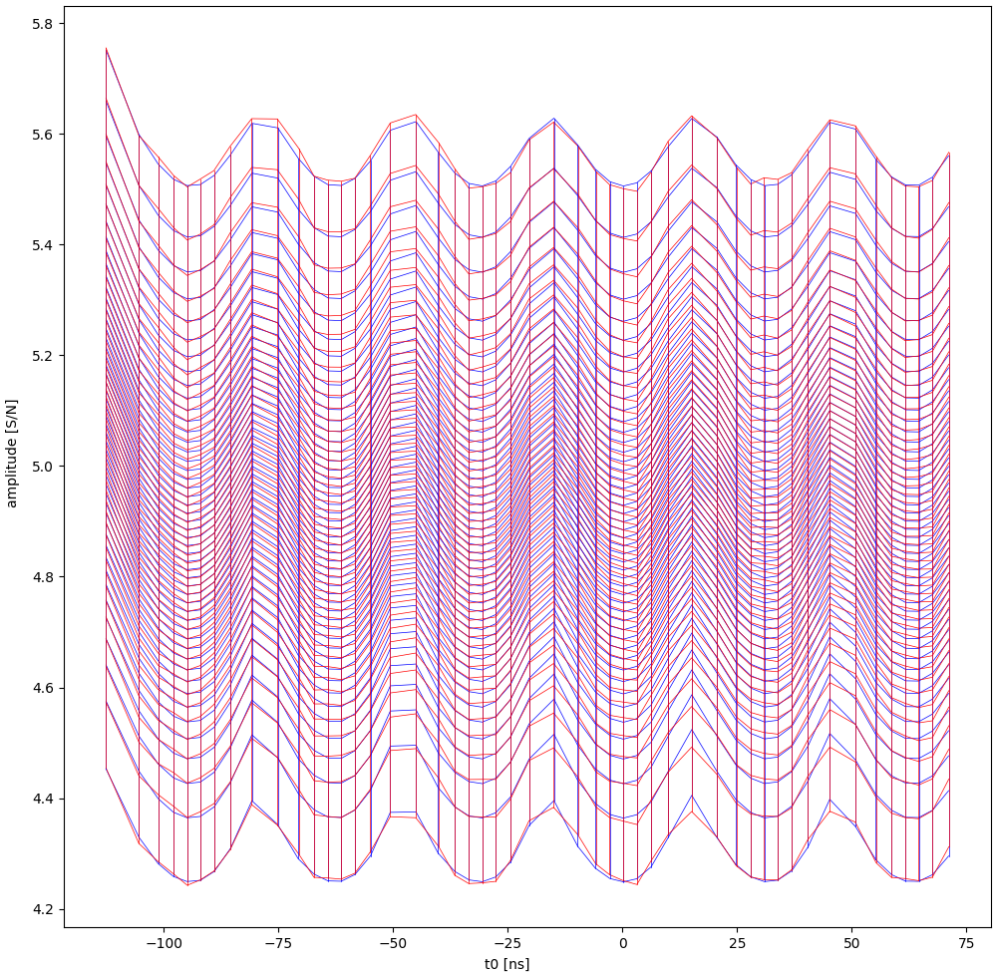

Figure 4.6: *Half-copula generated by neural network.* Blue lines are plotted from the training data, red lines shows the fit. This comparison is very sensitive and in fact we only look for general agreement between the patterns.

4.4 Normalization

The norm of the 2D probability distribution is proportional to strip noise rate, that is, to noise occupancy. At sufficiently high zero-suppression thresholds, noise rates are low and the norms are small. Our generators operate with distributions normalised to 1, and to create the production generator suitable for simulations, we need to know the actual noise rate.

The exact model calculates norms for given (T, σ, w), its predictions for some combinations are shown in Fig. 4.7, where the y-axis is the logarithm of the norm. We fitted a linear model to norms calculated by the exact model. The model comprised normalized polynomial components, generated from (T, σ, w) triplets up to 5th order. Fig. 4.8 shows the residuals - the agreement is very good. For small values of σ, the peaks of the probability distribution are very narrow and numerical errors of integration are larger.

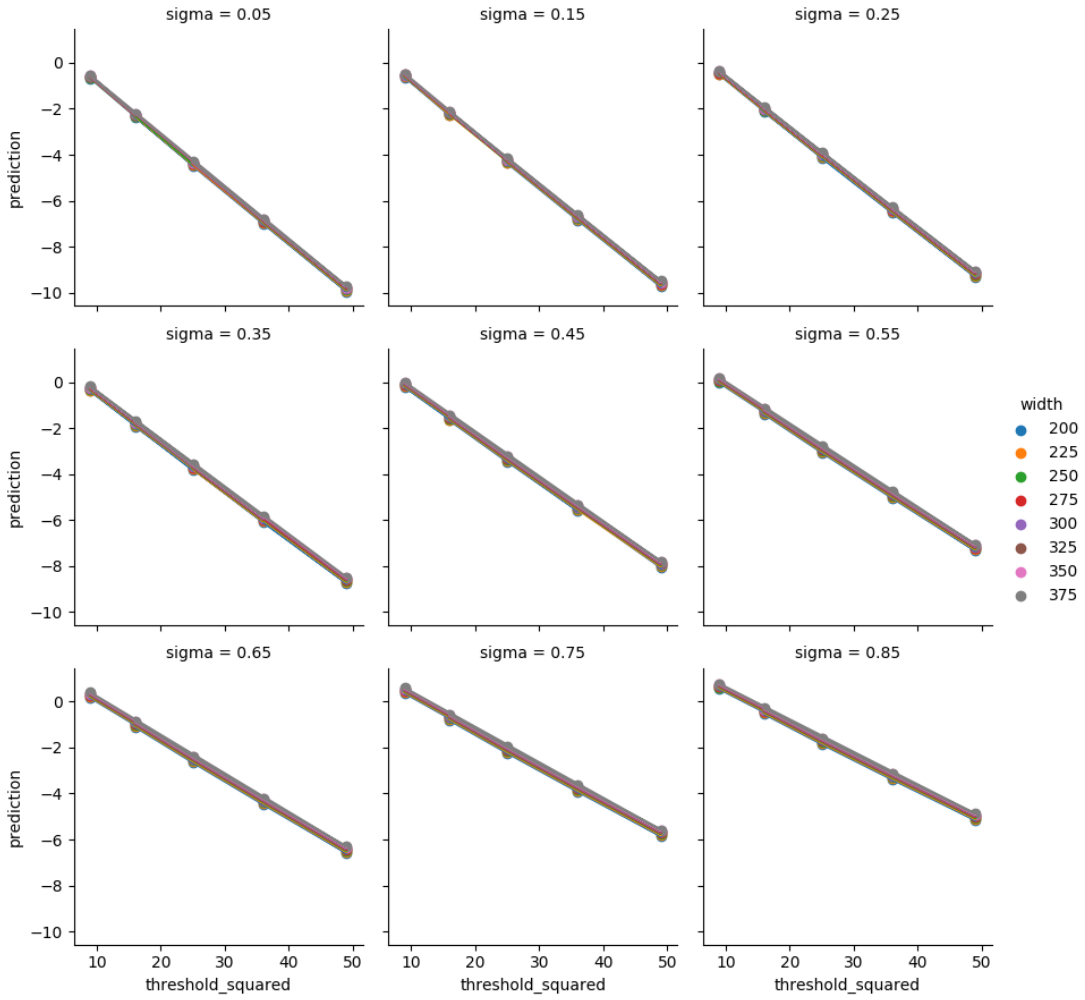

Figure 4.7: *Predicted normalizations.* Plots shows the logarithm of the norm as a function of threshold, sigma, and width.

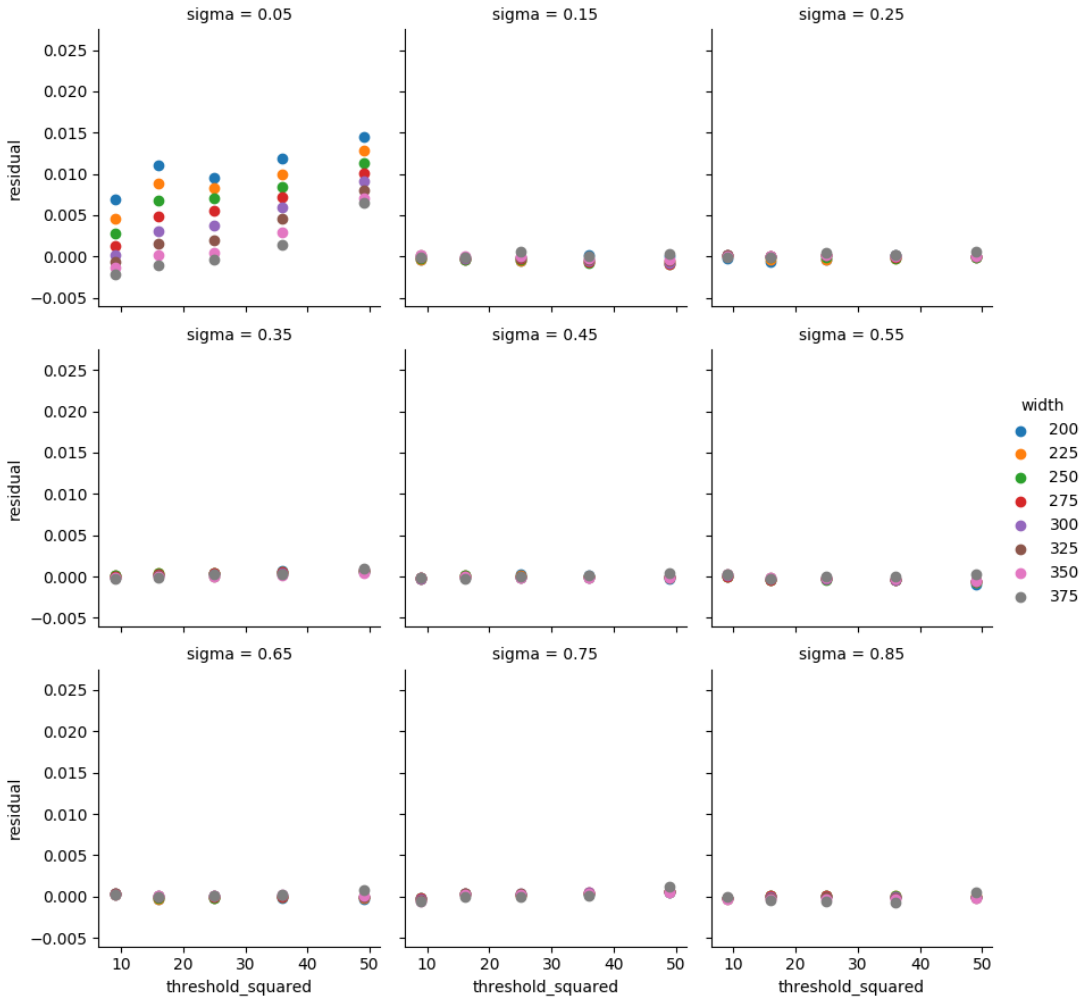

Figure 4.8: *Norm residuals.* Norms of network fits are in good agreement with linear model predictions.

4.4.1 Random forest sampler

Apart from neural network, we tried to fit boosted decision tree regressors to the data. Boosted trees provide a piecewise-constant approximation to the target functions. They learn much faster than neural networks, but generate more data to store. Performance of a decision tree depends on maximum allowed depth of a single tree and number of trees in a forest. Although we were able to train individual $u \to t$ and $(u, v) \to A$ estimators for (T, σ, w) triplets, once we included (σ, w) into the estimator, we were not able to obtain sufficiently precise estimators. The following table shows scores of a few random trees, especially amplitude scores are poor. A table showing the training process is in Attachment 4.

Table 4.3: Random trees score

T	σ	w	time score	amplitude score
4.0	0.25	200	0.99999988	0.99975581
5.0	0.50	300	0.99999971	0.99993959
4.0	0.50	200	0.99999978	0.99991203
4.0	0.25	300	0.99999654	0.99968532
4.0	0.05	300	0.99999979	0.99972972

For more details about decision trees and their implementation in Python, please refer to [17].

4.4.2 Gaussian mixture sampler

Another approach is to use a gaussian mixture model to directly fit the probability distribution. The idea is to represent the pdf by a linear combination of bivariate gaussians, and do the sampling by first selecting one of the gaussians and then taking a random sample from the gaussian. It is also more economical to store data about few tens of gaussians than parameters of the neural network. This method is implemented in Python as `BayesianGaussianMixture` class (see [18]) and works actually very well for two-parameter pdfs, that is, for pdfs calculated on a (T, σ, w) grid. The following pictures show an example of gaussian mixture fit, $(T, \sigma, w) = (5.0, 0.1, 300)$. Fig. 4.9 shows model 2D probability density and marginal distributions with random samples generated by gaussian mixture. Gaussian mixture components (including random samples) are shown in Fig. 4.10 and the total probability density produced by this mixture is plotted in Fig. 4.11. The plot in Fig 4.12 shows the distribution of mixture weights. More examples can be found in Attachment 5.

However, unlike in the cdf inversion approach, we have no easy way to make precise interpolation on this grid, the best we can do is sample from the nearest grid point.

An even more important problem is that the grid estimator is difficult to train on real data - we would have to discretize the data in σ and w to get enough

training data for each grid point. This is not a particularly elegant, but workable solution, which we, however, didn't pursue.

Instead, we tried a different approach using 4-parameter gaussian mixtures, parametrized by (σ, w, t, A). If we were able to fit the 4-parameter pdf with sufficient precision, we could extract the (t, A) distribution simply by conditioning on the given (σ, w) values, which is easily done for a combination of gaussians. However, by the time of writing this thesis, we had no working fitting procedure for the 4-parameter mixture - the training ends with very poor fits.

As the method has the merit of being directly usable with real data, it is worth putting some more effort into it.

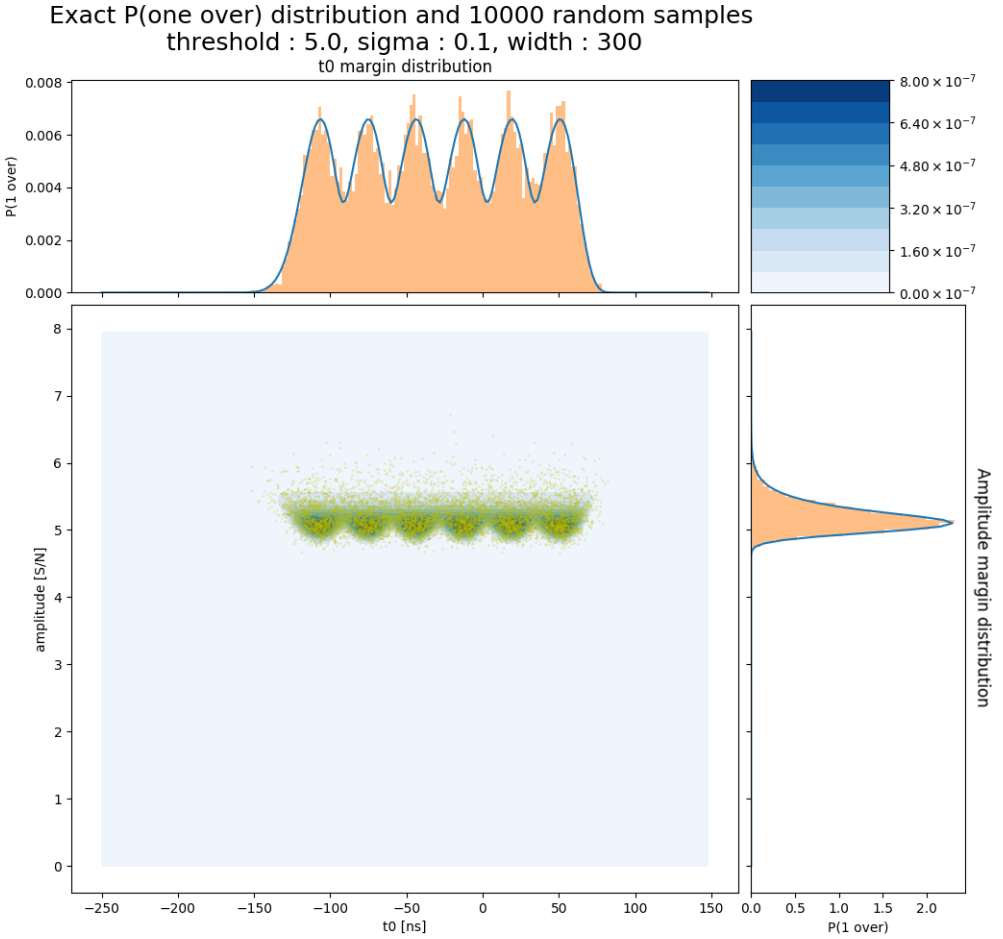

Figure 4.9: *2D probability density with random samples generated by gaussian mixture.* The central plot shows 2D probability density, amplitude margin is on the left side, time margin is on the top. Orange bars in margin plots and yellow dots in the 2D plot are 10000 random samples.

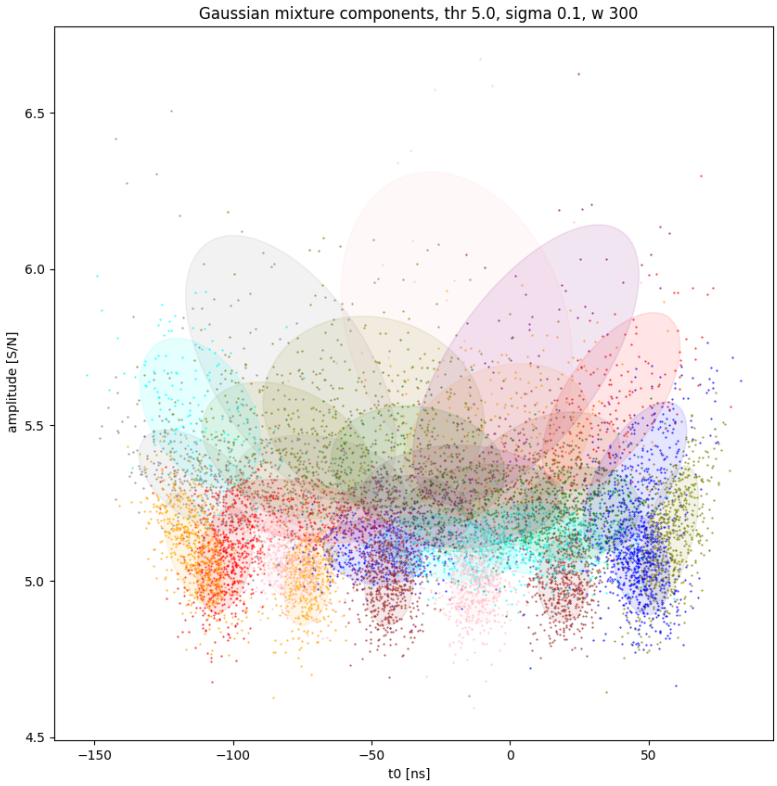

Figure 4.10: *Gaussian mixture components.* Coloured areas represent 2D gaussians, dots are random samples which share its colour with the appropriate gaussian.

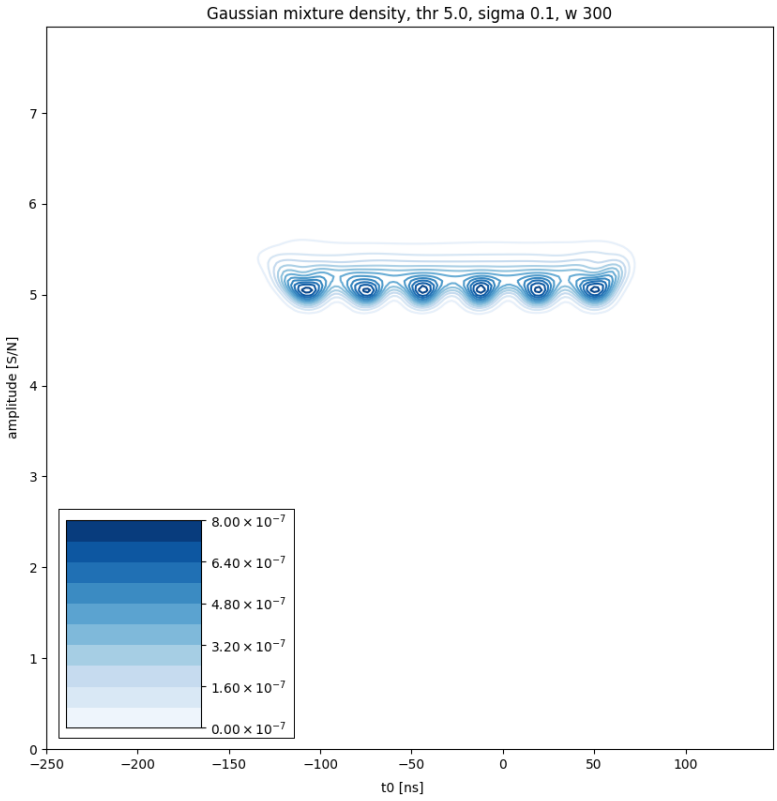

Figure 4.11: *Gaussian mixture total probability density.* The plot shows contour lines of 2D probability density composed from 2D gaussians.

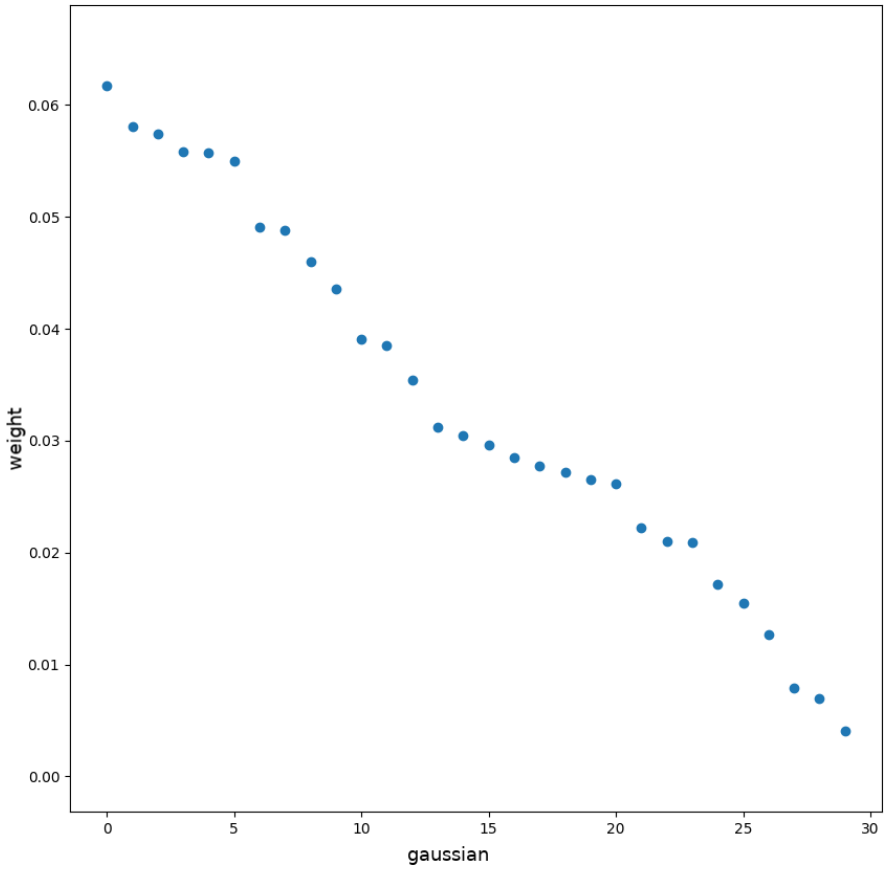

Figure 4.12: *Gaussian mixture component weights.* The mixture consist of 30 gaussians which are ordered according to their relative weights in the mixture.

4.5 Real Data Approach

To add noise to event simulations, experimental data from noise runs can be used. Noise runs are recorded before collisions to study background of the future measurements (e. g. determine the strip noise ranges). From this dataset we can chose random samples and add them to simulated event signals. This can be easily performed in basf2, Belle II software framework. The disadvantage of this method is the requirement of sufficiently large dataset to create independent random noise signals for large simulations.

4.6 Hybrid Approach

Noise run data are used to validate the performance of `NoiseGenerator`. Instead of building a generator it could be easier to use directly the experimentally measured noise as training data.

Neural network takes (T, σ, w) together with (u, v) and uses these inputs to compute the inverse distribution function and determine (t, A). Experimental data would have to be processed before passing it to the network which would be very complicated.

This approach may be possible with gaussian mixtures as this method only needs (t, A) pairs as inputs. As discussed above, the promising development of the gaussian mixture method is to operate with 4D distribution and using (σ, w) as parameters. If we manage to successfully train this kind of samplers, switching from generated data to experimental data would be a straightforward step.

Conclusion

The goal of this thesis was to develop a strip noise generator that would be applied in the production of simulations for the Belle II experiment.

We described the Belle II experiment and its detectors. A separate section is dedicated to the beam background sources to introduce the background signals competing for detector channels with signals from the studied physics processes.

The next part focuses on the SVD. We described semiconductor detectors, their structure and resolution, and characterized the electronic noise in silicon detector strips, which is another source of competitive signals. We also added more details about the Belle II SVD and its readout system.

We introduced the concept of artificial neural network and described its training process.

In the experimental part, we summarized our development process and results. First we described the theoretical background of the strip noise simulation. We introduced the theoretical model of the strip noise probability distribution and created an exact noise generator based on numerical inversion of the 2D distribution function. This generator was found unusable for production but we used it to generate training data for our machine-learning algorithms.

We constructed a system of neural networks that generates random noise samples for different threshold values. This design was enabled by the fact that threshold is never changed within a run and we could train multiple networks simultaneously. We managed to optimize the network parameters and training regimes and reached high accuracy of the sampler. We discussed the normalization of our models and showed that a linear model provides reasonable prediction of noise rates.

We also explored alternative approaches to creating a production generator. We tried to fit boosted decision tree regressors to the training data but the results have not reached the required accuracy. Another option we examined was a gaussian mixture. We managed to approximate the 2D probability distribution by a combination of 2D gaussians and generate samples from the mixture pdf. This generator worked well and would be easily trained on experimental data but we were not able to incorporate all necessary parametrizations with sufficiently accurate results.

The Appendix contains supplementary material - plots and tables illustrating the training process and showing results of different sampling methods and their parametrizations.

The Python code written for noise simulations is available at `https://github.com/zugru/SVDNoise`.

Bibliography

[1] Brodzicka, Jolanta, et al. *Physics achievements from the Belle experiment* [online]. Volume 2012, Issue 1, 2012, 04D001, https://doi.org/10.1093/ptep/pts072

[2] Belle II. *Super KEKB and Belle II* [online]. [cit. 26/03/19]. https://www.belle2.org/project/super_kekb_and_belle_ii

[3] High Energy Accelerator Research Organisation (KEK). *Belle II* [online]. [cit. 07/04/19]. https://www.kek.jp/en/Facility/IPNS/Belle2/

[4] Doležal, Zdeněk, Uno, S. et al. *Belle II Technical Design Report.* arXiv:1011.0352 [physics.ins-det], 2010

[5] Joint Institute for Nuclear Research. *Electrons and Positrons Collide for the first time in the SuperKEKB Accelerator* [online]. [cit. 26/04/19]. http://www.jinr.ru/posts/electrons-and-positrons-collide-for-the-first-time-in-the-superkekb-accelerator/

[6] Lewis, P.M., Jaegle, I., Nakayama, H. et al. *First Measurements of Beam Backgrounds at SuperKEKB.* arXiv:1802.01366 [physics.ins-det], 2018

[7] Kvasnička, Peter. *SVD strip noise simulation* [online]. [cit. 29/04/19]. https://github.com/PKvasnick/SVDNoise/blob/master/slides/PKvasnicka_SVDNoise_2018-09-26.pdf

[8] Spieler, Helmut. *Semiconductor Detector Systems.* Oxford University Press, 2005. ISBN 0-19-852784-5.

[9] Sanderson, Grant. *Deep learning, chapter 1: But what* is *a Neural Network?* [online]. [cit. 27/04/19]. https://www.youtube.com/watch?v=aircAruvnKk&t=96s

[10] Wikipedia, The Free Encyclopedia. *Artificial neural network* [online]. [cit. 27/04/19]. https://en.wikipedia.org/wiki/Artificial_neural_network

[11] Desmos. *Desmos - graphics calculator* [online]. 2019. https://www.desmos.com/calculator

[12] Sanderson, Grant. *Deep learning, chapter 2: Gradient descend, how neural networks learn* [online]. [cit. 27/04/19]. https://www.youtube.com/watch?v=IHZwWFHWa-w&t=1082s

[13] Sanderson, Grant. *Deep learning, chapter 3: What is backpropagation really doing?* [online]. [cit. 27/04/19]. https://www.youtube.com/watch?v=Ilg3gGewQ5U

[14] Spieler, Helmut. *Introduction to Radiation Detectors and Electronics: VIII.6. Rate of Noise Pulses in Threshold Discriminator Systems*. Lecture Notes - Physics 198, Spring Semester 1998 - UC Berkeley. Avaliable from http://www-physics.lbl.gov/~spieler/physics_198_notes/PDF/VIII-6-rate.pdf

[15] SciPy.org. *NumPy* [online]. 2019. https://www.numpy.org

[16] scikit-learn. *scikit-learn - Machine Learning in Python* [online]. 2019. https://scikit-learn.org/stable/

[17] scikit-learn. *1.10. Decision Trees* [online]. 2019. https://scikit-learn.org/stable/modules/tree.html#tree

[18] scikit-learn. *sklearn.mixture.BayesianGaussianMixture* [online]. 2019. https://scikit-learn.org/stable/modules/generated/sklearn.mixture.BayesianGaussianMixture.html#sklearn.mixture.BayesianGaussianMixture

A. Attachments

A.1 Exact generator

The following figures shows more examples of probability densities, marginal distributions, and half-copulas plotted by `NoiseGenerator`.

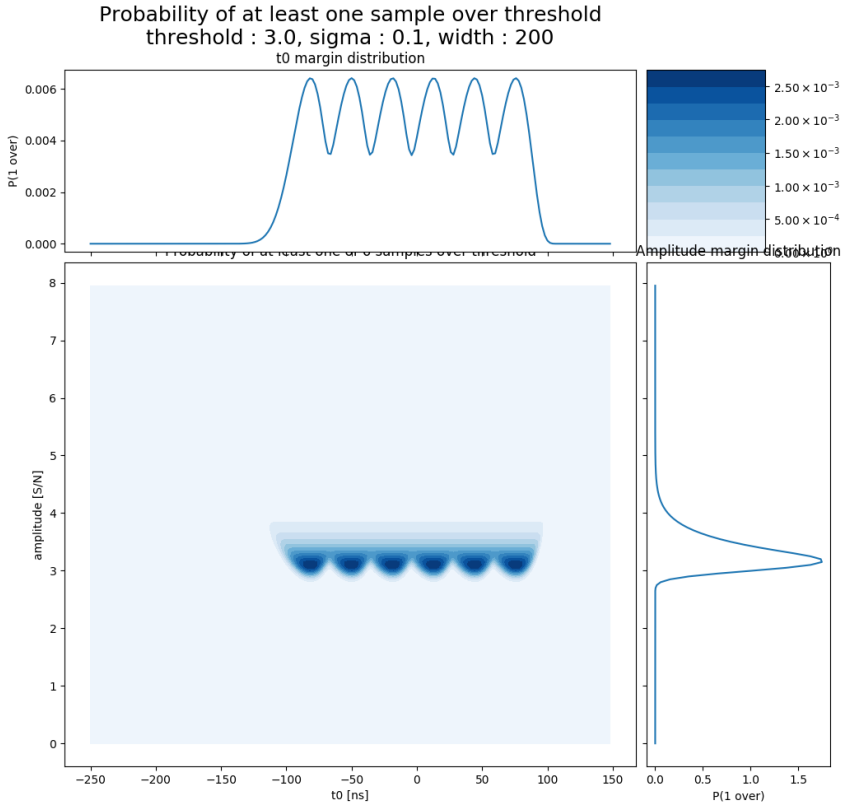

Figure A.1: *Training data example 1 (probability density).* This is a plot for $(T, \sigma, w) = (3.0, 0.1, 200)$. The central plot shows the 2D probability density, amplitude margin is on the left side, time margin is on top.

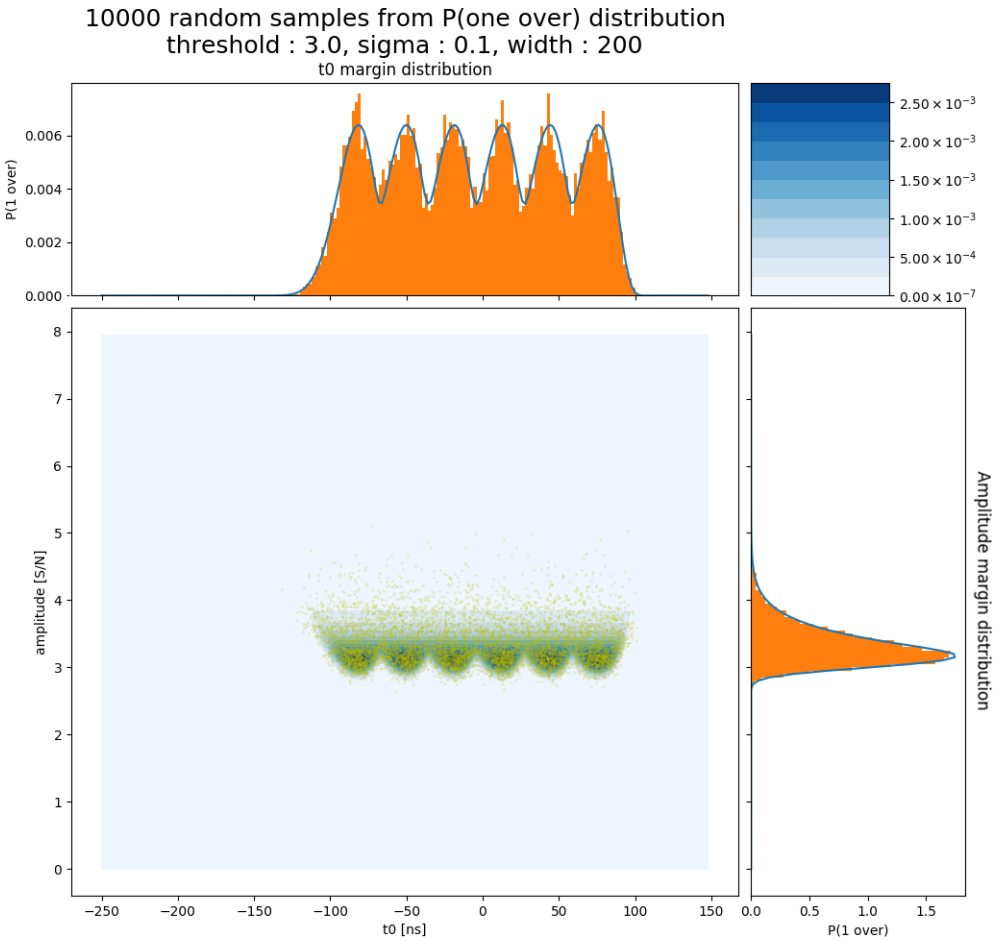

Figure A.2: *Training data example 1 (samples).* This is a plot for $(T, \sigma, w) = (3.0, 0.1, 200)$. The central plot shows the 2D probability density, amplitude margin is on the left side, time margin is on top. Orange bars in margin plots and yellow dots in the 2D plot are 10000 random samples, blue contours are the exact model.

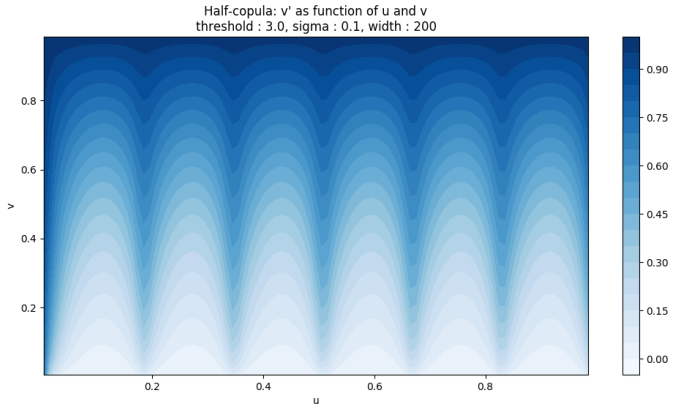

Figure A.3: *Training data example 1 (half-copula).* This is a plot for $(T, \sigma, w) = (3.0, 0.1, 200)$. Half-copula is used to calculate v' from u and v.

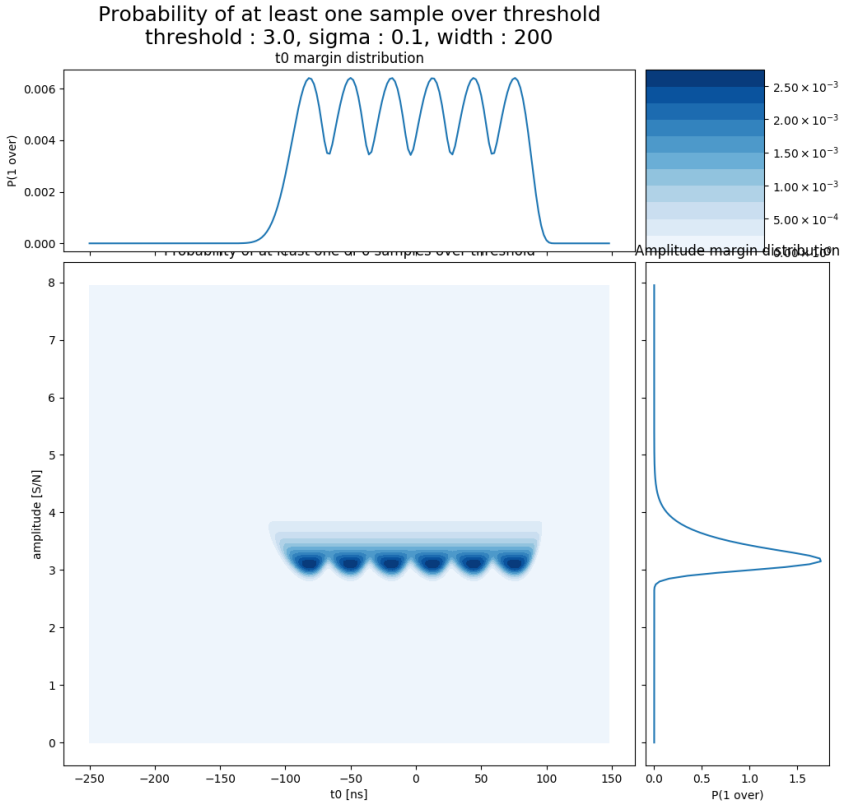

Figure A.4: *Training data example 2 (probability density).* This is a plot for $(T, \sigma, w) = (7.0, 0.5, 250)$. The central plot shows the 2D probability density, amplitude margin is on the left side, time margin is on top.

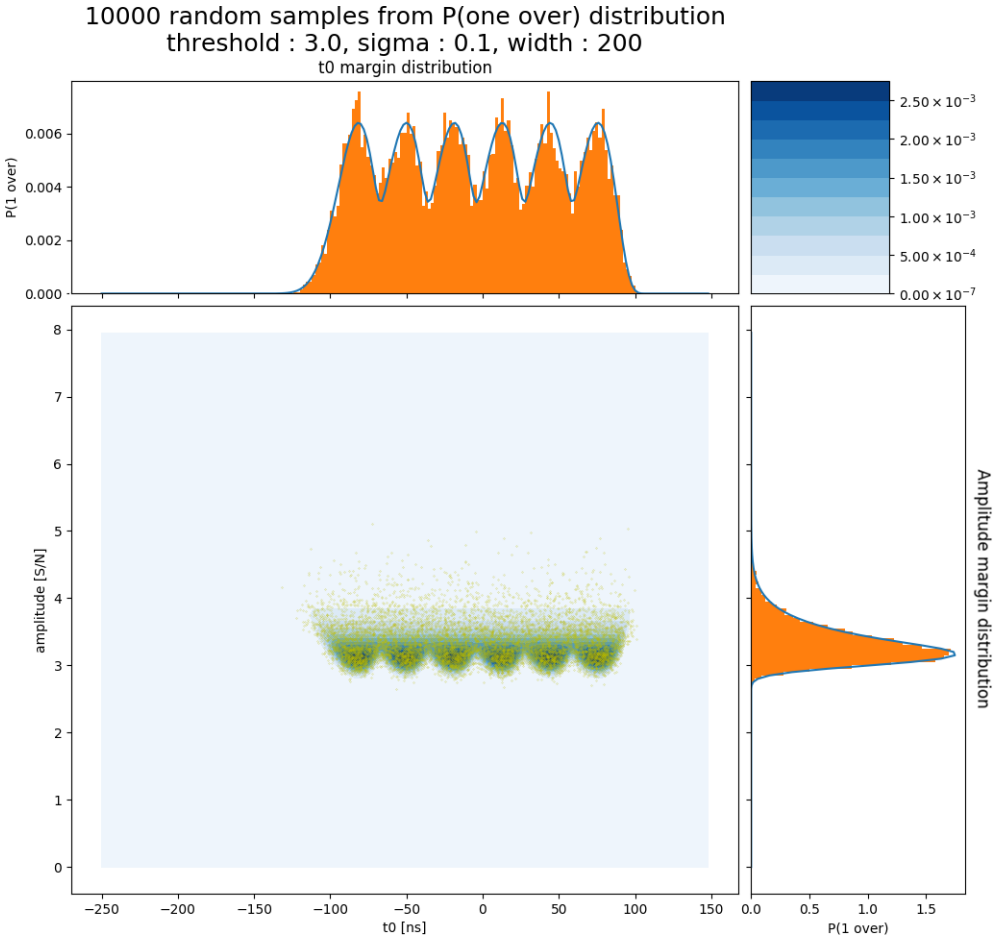

Figure A.5: *Training data example 2 (samples).* This is a plot for $(T, \sigma, w) = (7.0, 0.5, 250)$. The central plot shows the 2D probability density, amplitude margin is on the left side, time margin is on top. Orange bars in margin plots and yellow dots in the 2D plot are 10000 random samples, blue contours are the exact model.

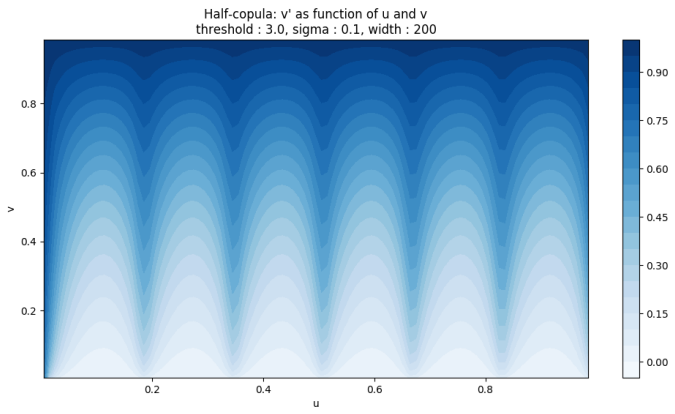

Figure A.6: *Training data example 2 (half-copula).* This is a plot for $(T, \sigma, w) = (7.0, 0.5, 250)$. Half-copula is used to calculate v' from u and v.

A.2 Neural network training

Attached tables shows some of the partial results and optimalization progress.

t0 activation: tanh nn: by-threshlod_multi

layers	batch_size	alpha	tol	max_iter	chi2 (t0)	chi2 (amp)	time (real)
100-100	500	1.00E-07	1.00E-08	500	0.099602	0.006998	129m45,974s
250-250	500	1.00E-07	1.00E-08	500	0.093489	0.006998	163m20,211s
100-100	500	1.00E-08	1.00E-08	500	0.099602	0.006998	135m10,778s
100-100	500	1.00E-07	1.00E-08	200	0.099602	0.006998	129m22,847s
100-100	200	1.00E-07	1.00E-08	500	0.097227	0.006998	116m9,659s
250-250	200	1.00E-07	1.00E-08	500	0.091305	0.008091	160m48,545s
+ amplitude batch_size = 200							
250-250	200	1.00E-07	1.00E-08	500	0.091305	0.008066	159m18,616s
+ amplitude layers = (300,300)							
300-300	200	1.00E-07	1.00E-08	500	0.086258	0.006998	142m39,468s
+ amplitude tol = 1e-9							
300-300	200	1.00E-07	1.00E-08	200	0.086258	0.007654	139m21,692s
+ amplitude max_iter = 200							
500-500	200	1.00E-07	1.00E-08	200	0.064978	0.006998	198m39,882s
300-300	200	1.00E-07	1.00E-08	200	0.086258	0.020309	135m11,363s
+ amplitude layers = (256,256,256)							
100-100	200	1.00E-07	1.00E-08	200	0.097227	0.006998	115m8,601s
+ amplitude max_iter = 300							
100-100	200	1.00E-07	1.00E-07	200	0.097227	0.006998	115m15,565s
+ amplitude max_iter = 300							
100-100	200	1.00E-07	1.00E-06	200	0.097227	0.024125	139m47,738s
+ amplitude max_iter = 300		+ amplitude tol = 1e-6					
100-100	200	1.00E-07	1.00E-05	200	0.097227	0.006998	148m36,797s
+ amplitude max_iter = 300							
100-100	200	1.00E-07	1.00E-09	200	0.097227	0.006998	158m47,443s
+ amplitude max_iter = 300		+ amplitude tol = 1e-9					
70-70	200	1.00E-07	1.00E-08	200	0.100565	0.006998	125m47,883s
+ amplitude max_iter = 300							
100-100	200	1.00E-07	1.00E-08	200	0.097227	0.006998	123m4,577s
+ amplitude max_iter = 300							
100-100	200	1.00E-07	1.00E-08	200	0.097227	0.006998	116m43,705s
+ amplitude max_iter = 300							
100-100	200	1.00E-07	1.00E-08	200	0.097227	0.022085	188m48,683s
+ amplitude max_iter = 200		+ amplitude layers = (200,200)					
100-100	200	1.00E-07	1.00E-08	200	0.097227	0.022085	132m10,700s
+ amplitude max_iter = 300		+ amplitude layers = (200,200)					
100-100	200	1.00E-07	1.00E-08	200	0.097227	0.006998	127m54,924s
+ amplitude max_iter = 300							
100-100	200	1.00E-07	1.00E-08	200	0.097227	0.008066	real 148m15,824s
+ amplitude max_iter = 300		+ amplitude layers = (300,300)					

Table A.1: *Neural network training process.* The table contains optimalization for time and amplitude network with $T = 0.5$. Cyan colour marks changed parameter, green and red colour mark improvement or deterioration, respectively.

	amplitude network	t0 network
layers	250-250	100-100
	chi2	chi2
thr = 3	0.011972	0.052715
thr = 4	0.006986	0.077846
thr = 5	0.022085	0.097227
thr = 6	0.010440	0.155683
thr = 7	0.012979	0.093256
time	132m10,700s	

	amplitude network	t0 network
layers	256-256	100-100
	chi2	chi2
thr = 3	0.011659	0.052715
thr = 4	0.007220	0.077846
thr = 5	0.006998	0.097227
thr = 6	0.012654	0.155683
thr = 7	0.011265	0.011265
time	127m54,924s	

	amplitude network	t0 network
layers	300-300	250-250
	chi2	chi2
thr = 3	0.009523	0.048456
thr = 4	0.005964	0.084373
thr = 5	0.008066	0.091305
thr = 6	0.008875	0.071106
thr = 7	0.010676	0.089439
time	159m18,616s	

Table A.2: *Influence of layers.* The table shows the influence of layers' sizes on the resultant accuracy. We focused on $T = 5.0$ but the accuracy of other networks is also important factor to track. Green and red colour mark whether the value is below, or above our desired precision.

A.3 Neural network generator

The following plots show our final neural network generator fits for different (T, σ, w) combinations.

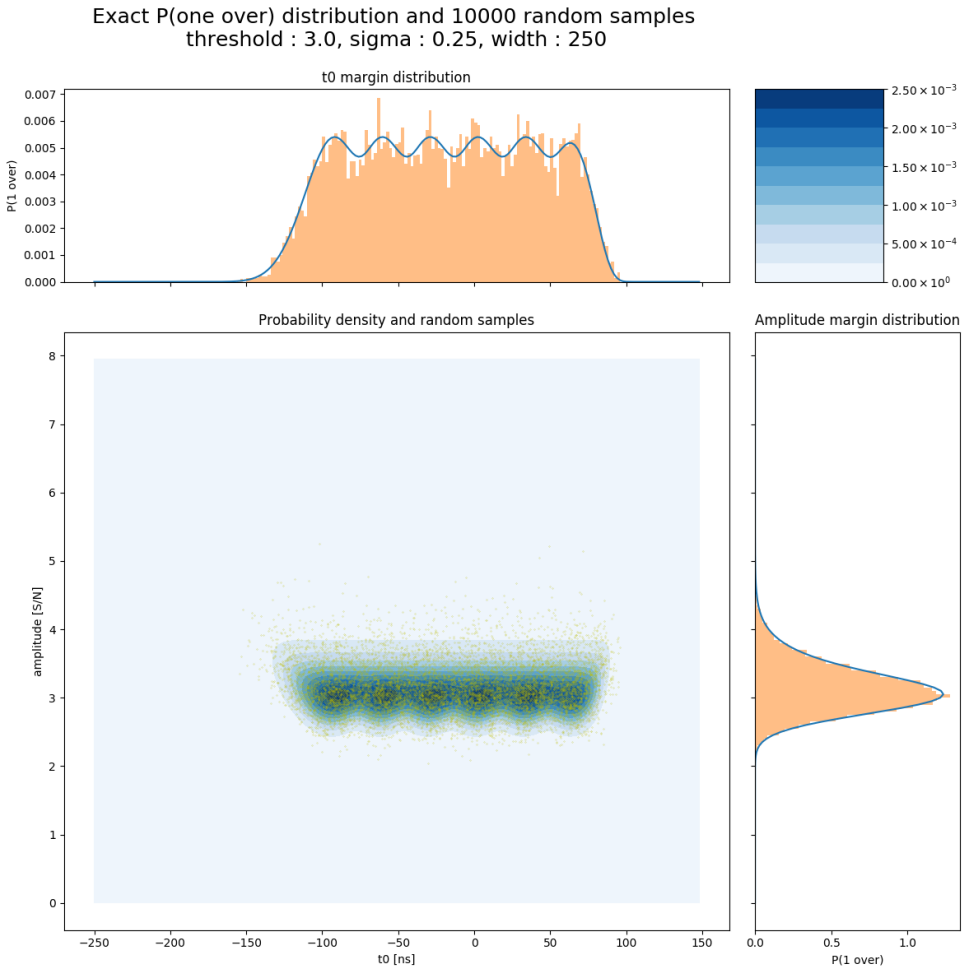

Figure A.7: *Network fit example 1 (samples).* This is a fit for $(T, \sigma, w) = (3.0, 0.25, 250)$. The central plot shows the 2D probability density, amplitude margin is on the left side, time margin is on top. Orange bars in margin plots and yellow dots in the 2D plot are 10000 random samples, blue contours are the exact model.

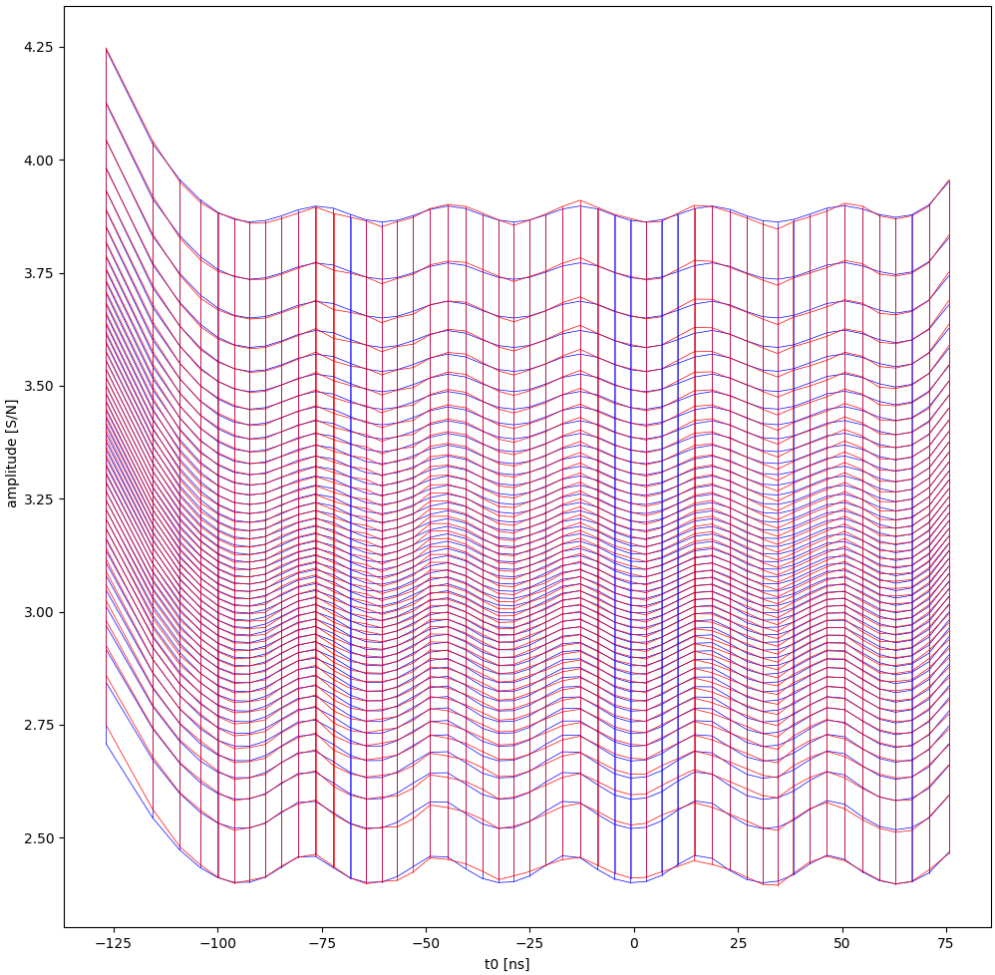

Figure A.8: *Network fit example 1 (half-copula).* Blue lines are plotted from the training data, red lines shows the fit. This comparison is very sensitive and in fact we only look for general agreement between the patterns.

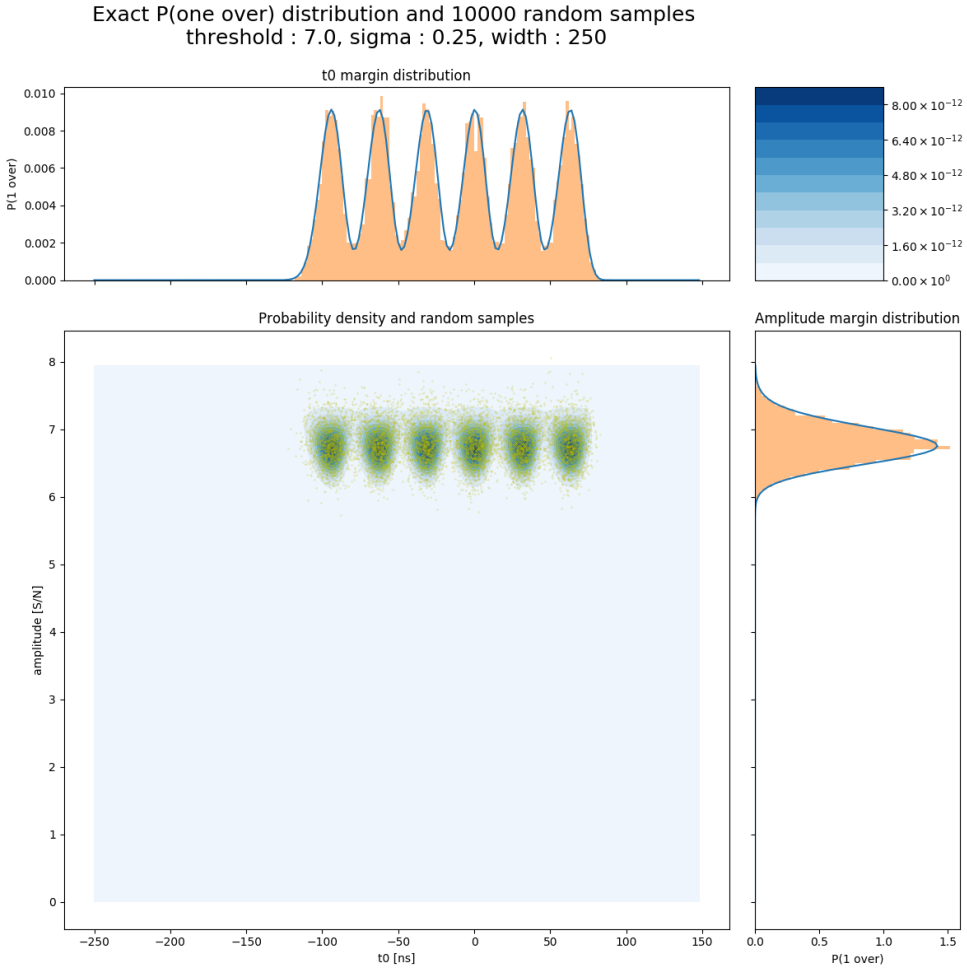

Figure A.9: *Network fit example 2 (samples).* This is a fit for $(T, \sigma, w) = (7.0, 0.25, 250)$. The central plot shows the 2D probability density, amplitude margin is on the left side, time margin is on top. Orange bars in margin plots and yellow dots in the 2D plot are 10000 random samples, blue contours are the exact model.

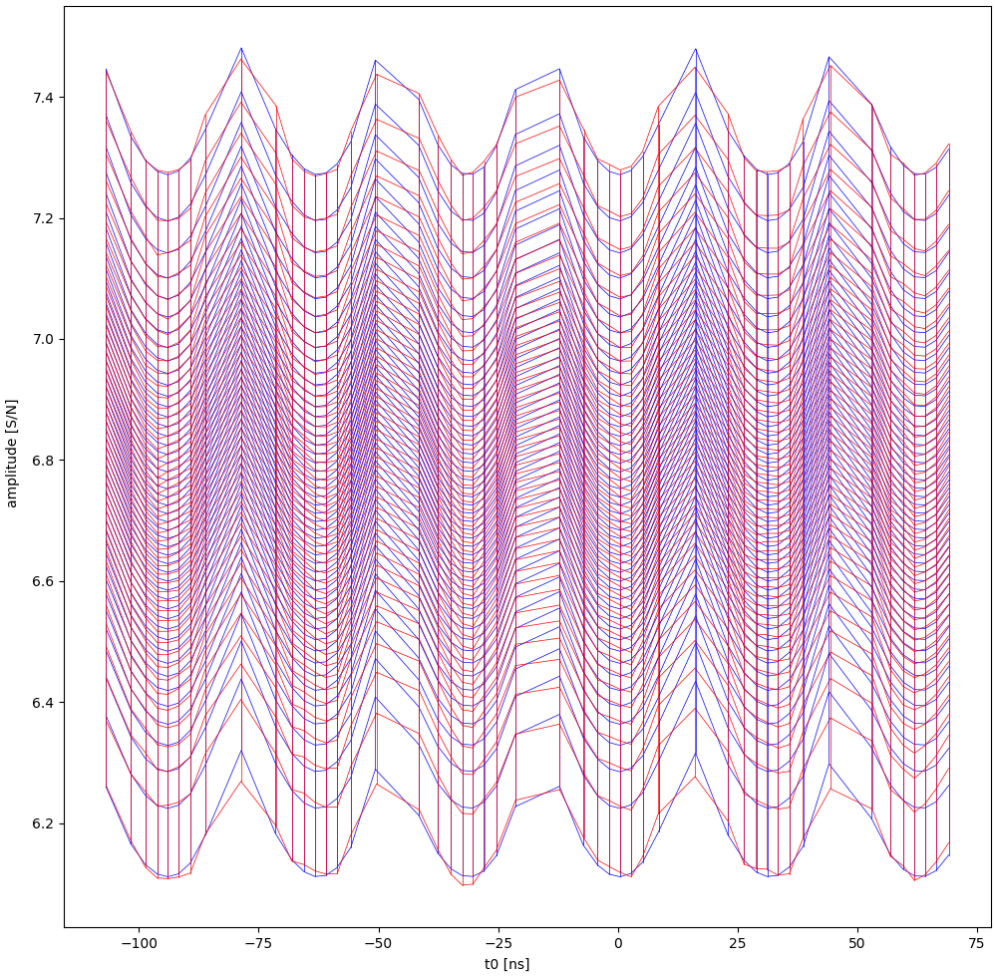

Figure A.10: *Network fit example 2 (half-copula).* Blue lines are plotted from the training data, red lines shows the fit. This comparison is very sensitive and in fact we only look for general agreement between the patterns.

A.4 Random forest generator

The following table shows an example of random tree training.

T = 4	S = 0.25	W = 200					
time		amplitude					
n_estimators	max_leaf_nodes	n_estimators	min_samples_split	max_depth	max_leaf_nodes	time score	amplitude score
---	500	100	---	---	2000	0.99999987	0.99907854
---	500	100	---	---	2000	0.99999985	0.99907841
---	500	100	---	---	2000	0.99999987	0.99907407
---	500	200	---	---	2000	0.99999989	0.99911023
---	500	200	---	---	2000	0.99999987	0.99911767
100	500	200	---	---	2000	0.99999994	0.99910216
100	500	200	---	---	2000	0.99999993	0.99910406
200	500	200	---	---	2000	0.99999997	0.99911473
100	500	250	---	---	2000	0.99999994	0.99911515
100	500	250	---	---	2000	0.99999994	0.99911255
100	500	300	---	---	2000	0.99999994	0.99912436
---	500	200	---	---	2000	0.99999987	0.99911358
---	500	200	10	---	2000	0.99999987	0.99911173
---	500	200	10	---	2000	0.99999987	0.99910459
---	500	200	25	---	2000	0.99999986	0.99911287
---	500	200	---	1000	2000	0.99999987	0.99909780
---	500	200	---	1000	2000	0.99999986	0.99910999
---	500	200	---	500	2000	0.99999988	0.99911687
---	500	200	---	500	2000	0.99999986	0.99910587
---	500	200	---	100	2000	0.99999988	0.99910957
---	500	200	---	50	2000	0.99999988	0.99911247
---	500	200	---	10	2000	0.99999988	0.99336751
---	500	200	---	---	1000	0.99999988	0.99772889
---	500	200	---	---	1500	0.99999988	0.99859400
---	500	200	---	---	2000	0.99999988	0.99911536
---	500	200	---	---	2500	0.99999987	0.99933822
---	500	200	---	---	3000	0.99999986	0.99945390
---	500	200	---	---	5000	0.99999989	0.99964065
---	200	200	---	---	5000	0.99999922	0.99964819
---	500	200	---	---	None	0.99999987	0.99974667
---	500	200	---	---	None	0.99999988	0.99975580

Table A.3: *Random forest training process.* The table contains optimalization for one (T, σ, w) gridpoint. Cyan colour marks parameter change compared to the previous row.

A.5 Gaussian mixture generator

Attached plots show more examples of gaussian mixture fits.

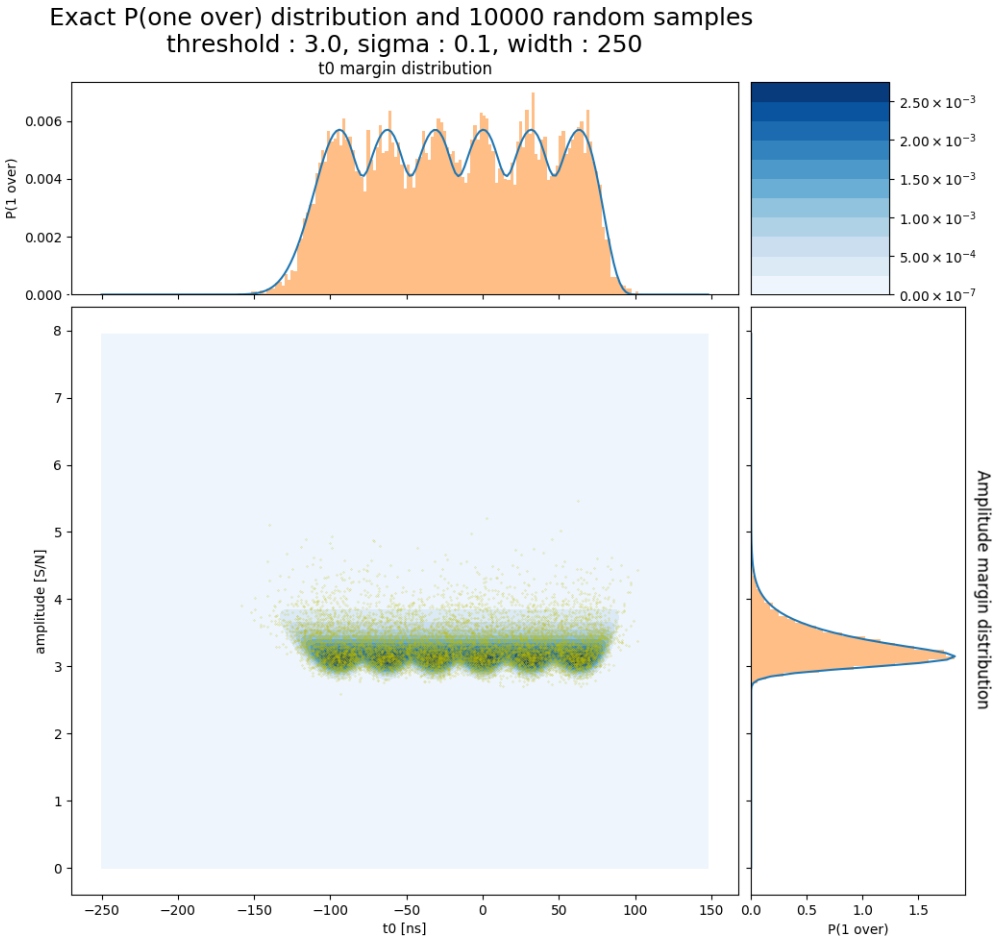

Figure A.11: *Gaussian mixture fit example 1 (samples).* This is a fit for $(T, \sigma, w) = (3.0, 0.1, 250)$. The central plot shows the 2D probability density, amplitude margin is on the left side, time margin is on top. Orange bars in margin plots and yellow dots in the 2D plot are 10000 random samples.

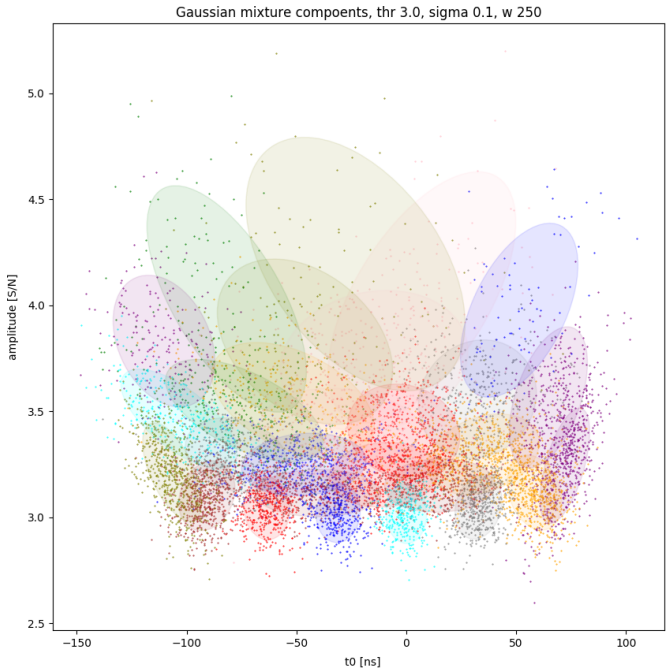

Figure A.12: *Gaussian mixture fit example 1 (components).* This is a fit for $(T, \sigma, w) = (3.0, 0.1, 250)$. Coloured areas represent 2D gaussians, dots are random samples which share its colour with the appropriate gaussian.

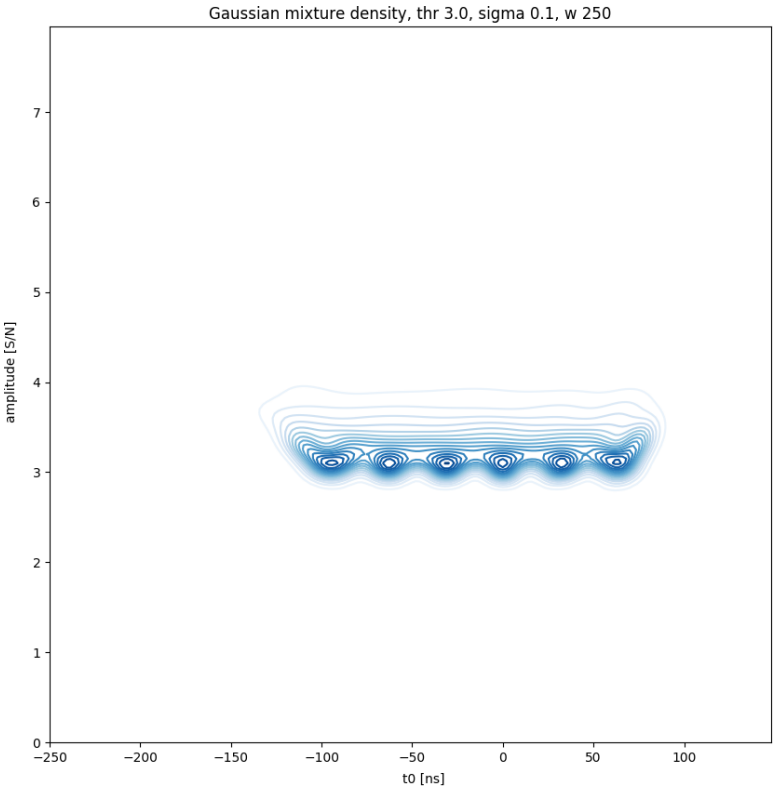

Figure A.13: *Gaussian mixture fit example 1 (total probability density).* This is a fit for $(T, \sigma, w) = (3.0, 0.1, 250)$. The plot shows contour lines of 2D probability density composed from 2D gaussians.

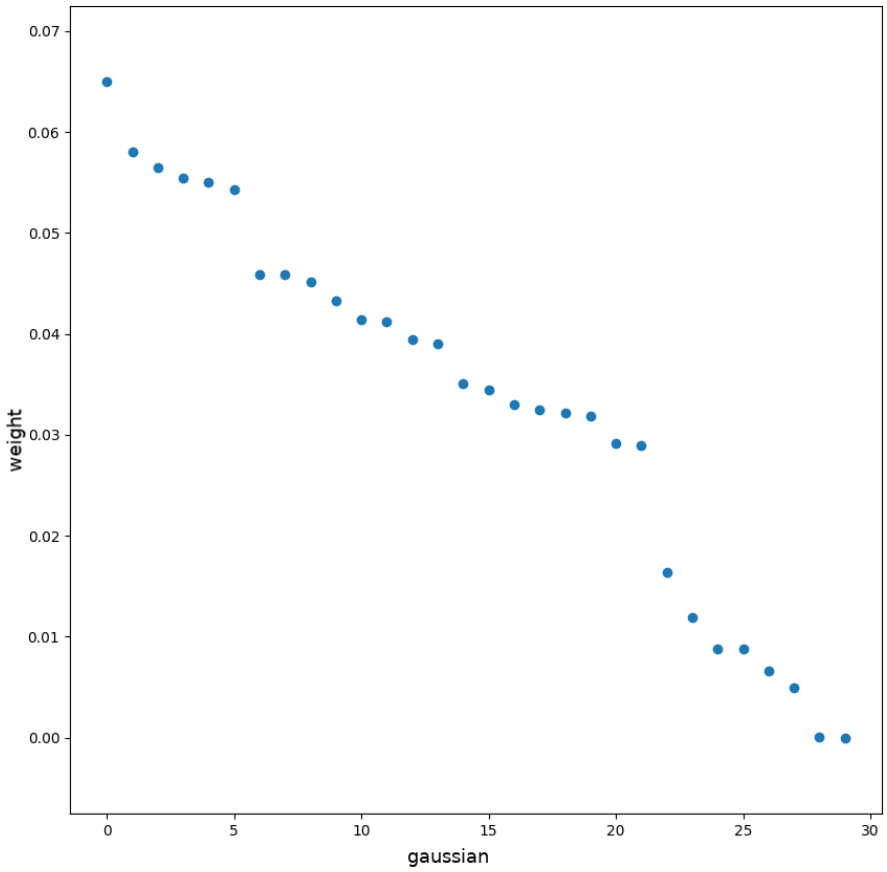

Figure A.14: *Gaussian mixture fit example 1 (weights).* This is a fit for $(T, \sigma, w) = (3.0, 0.1, 250)$. The mixture consist of 30 gaussians which are ordered according to their relative weights in the mixture.

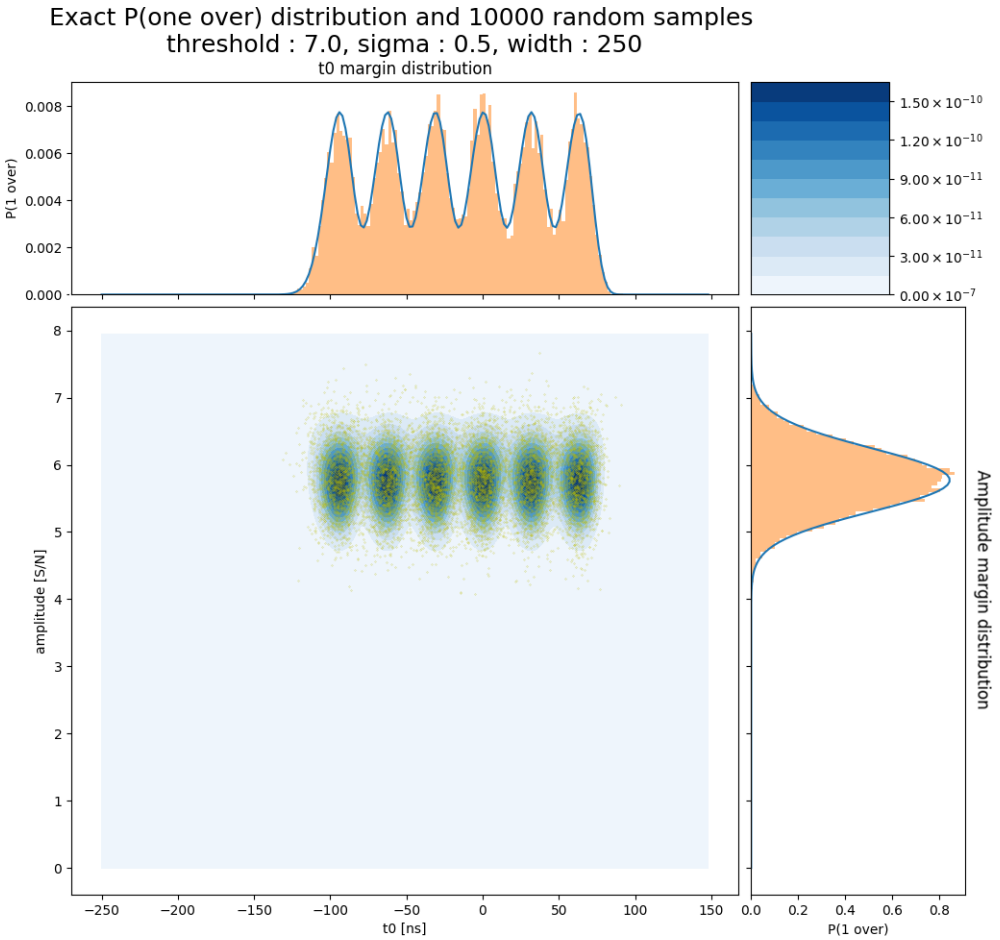

Figure A.15: *Gaussian mixture fit example 2 (samples).* This is a fit for $(T, \sigma, w) = (7.0, 0.5, 250)$. The central plot shows the 2D probability density, amplitude margin is on the left side, time margin is on top. Orange bars in margin plots and yellow dots in the 2D plot are 10000 random samples.

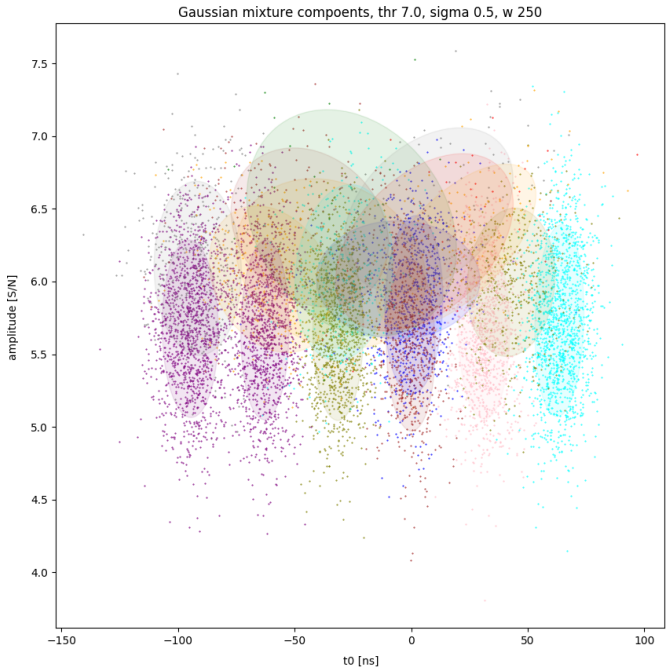

Figure A.16: *Gaussian mixture fit example 2 (components).* This is a fit for $(T, \sigma, w) = (7.0, 0.5, 250)$. Coloured areas represent 2D gaussians, dots are random samples which share its colour with the appropriate gaussian.

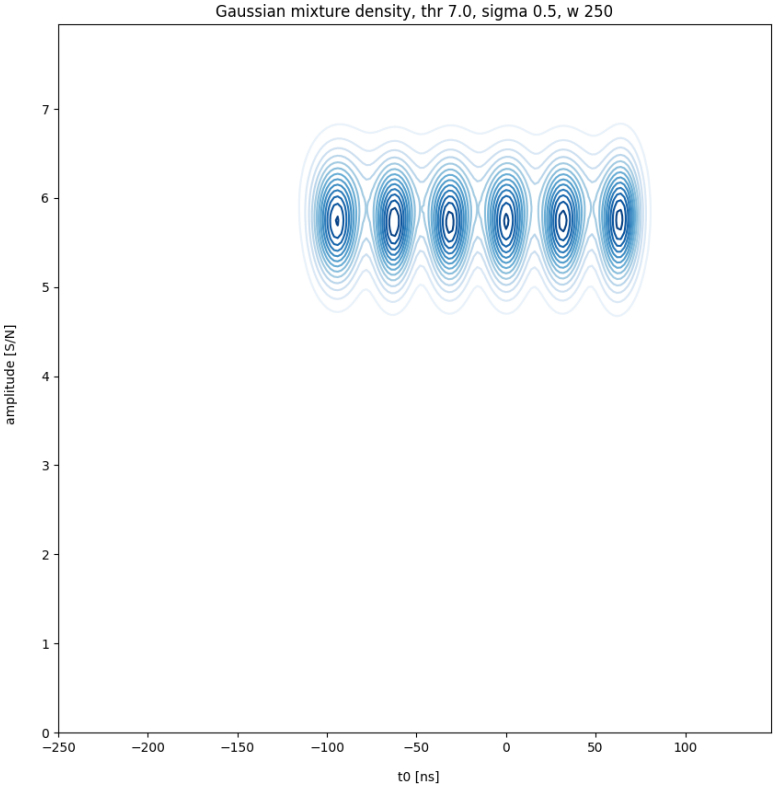

Figure A.17: *Gaussian mixture fit example 2 (total probability density).* This is a fit for $(T, \sigma, w) = (7.0, 0.5, 250)$. The plot shows contour lines of 2D probability density composed from 2D gaussians.

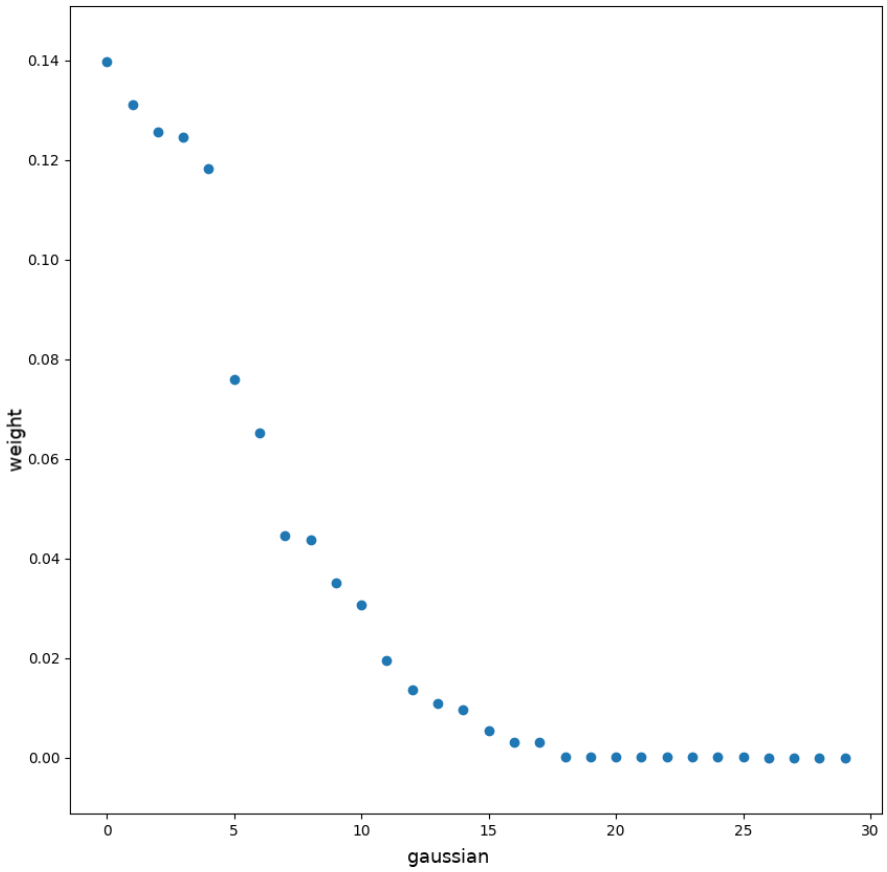

Figure A.18: *Gaussian mixture fit example 2 (weights).* This is a fit for $(T, \sigma, w) = (7.0, 0.5, 250)$. The mixture consist of 30 gaussians which are ordered according to their relative weights in the mixture.