Noise Generation for Continuous System Simulation

Andreas Klöckner and Franciscus L. J. van der Linden and Dirk Zimmer

Adding random disturbances to Modelica models is necessary to represent stochastic fluctuations like sensor noise, air gusts and road irregularities. In this paper, we present a library to specify a pseudo random noise for continuous-time simulations. The random number generator, a probability density function and a frequency spectrum can be defined independently. A new random number generator is proposed to generate a continuous random signal, which is proven to be highly suitable for continuous models. The performance of the noise models is tested in two benchmarks using an academic as well as a realistic model both showing a remarkable increase in simulation speed.

Keywords: Noise, Stochastic Models, Random Number Generator

Citation Notice

[1] Andreas Klöckner, Franciscus L. J. van der Linden, and Dirk Zimmer. Noise Generation for Continuous System Simulation. In Hubertus Tummescheit and Karl-Erik Årzén, editors, *Proceedings of the 10th International Modelica Conference*, number 96 in Linköping Electronic Conference Proceedings, pages 837–846, Lund, Sweden, March 10-12 2014. Modelica Association and Linköping University Electronic Press. ISBN: 978-91-7519-380-9. ISSN: 1650-3686. eISSN: 1650-3740. doi:10.3384/ECP14096837.

```
@INPROCEEDINGS{kloeckner2014noise,
  author = {Andreas Kl{\"o}ckner and Franciscus L. J. van der Linden and Dirk
       Zimmer},
  title = {{Noise Generation for Continuous System Simulation}},
  booktitle = {Proceedings of the 10th International Modelica Conference},
  year = {2014},
  editor = {Hubertus Tummescheit and Karl-Erik {\AA}rz{\'e}n},
  number = {96},
  series = {Link{\"o}ping Electronic Conference Proceedings},
  pages = {837-846},
  address = {Lund, Sweden},
  month = {March 10-12},
  publisher = {Modelica Association and Link{\"o}ping University Electronic Press},
  note = {ISBN: 978-91-7519-380-9. ISSN: 1650-3686. eISSN: 1650-3740.},
  abstract = {Adding random disturbances to Modelica models is necessary to represent
       stochastic fluctuations like sensor noise, air gusts and road irregularities.
       In this paper, we present a library to specify a pseudo random noise
       for continuous-time simulations. The random number generator, a probability
       density function and a frequency spectrum can be defined independently.
       A new random number generator is proposed to generate a continuous
       random signal, which is proven to be highly suitable for continuous
       models. The performance of the noise models is tested in two benchmarks
       using an academic as well as a realistic model both showing a remarkable
       increase in simulation speed.},
  doi = {10.3384/ECP14096837},
  keywords = {Noise, Stochastic Models, Random Number Generator},
  owner = {kloe_ad},
  timestamp = {2014.03.21}
}
```

Noise Generation for Continuous System Simulation

Andreas Klöckner
andreas.kloeckner@dlr.de

Franciscus L. J. van der Linden
franciscus.vanderlinden@dlr.de

Dirk Zimmer
dirk.zimmer@dlr.de

German Aerospace Center (DLR), Institute of System Dynamics and Control,
82234 Weßling, Germany

Abstract

Adding random disturbances to Modelica models is necessary to represent stochastic fluctuations like sensor noise, air gusts and road irregularities. In this paper, we present a library to specify a pseudo random noise for continuous-time simulations. The random number generator, a probability density function and a frequency spectrum can be defined independently. A new random number generator is proposed to generate a continuous random signal, which is proven to be highly suitable for continuous models. The performance of the noise models is tested in two benchmarks using an academic as well as a realistic model both showing a remarkable increase in simulation speed.

Keywords: Noise, Stochastic Models, Random Number Generator

1 Motivation

When simulating real-world systems, the problem of introducing disturbances to the nominal system eventually becomes an issue. Especially, when dealing with controlled systems, important tasks are to check whether the controller is able to reject realistic disturbances, and to assess the performance of the system including noise. The problem is not limited to the field of control design, but is also of interest in e.g. specification of aircraft airworthiness requirements with respect to turbulence, estimation of the power outcome of wind energy farms or when interpreting contaminated sensor readings of experiments.

These kinds of distortions are typically taken into account by additive injection of a noise signal into the system as shown in Fig. 1. The noise signal can have a strong impact on the system's performance and must thus be specified carefully. However, there are no convenient means of specifying noise properties in Modelica, such that typical approaches implement ad-hoc modifications of a simple random signal.

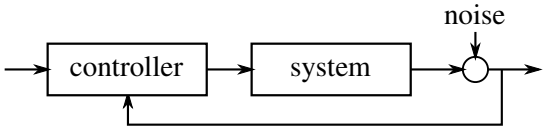


Figure 1: Noise is typically introduced additively into a controlled system.

Additionally, injecting noise into a continuous system typically decreases the simulation speed drastically, because standard noise generators are sampled systems, which generate time events by definition. These time events lead in most cases to integrator restarts, imposing a big penalty on simulation performance.

In this work, we present ways to solve the two issues outlined above in an integrated library:

1. We describe a general procedure to specify a suitable noise signal by means of selecting a high-quality random number generator, a probability distribution and a power spectrum (see Sec. 3).

2. By providing a continuous noise signal formulation using the sample-free generators introduced in Sec. 3.1 and a smooth interpolation (Sec. 3.3), we provide means for continuous noise generation. Avoiding events and using a smoothly filtered signal speeds up the simulation as compared to standard methods (see Sec. 5).

3. The methods and processes are integrated in a library with convenient user interfaces (see Sec. 4). This enables a user to easily specify a desired noise signal and to use it in complex simulation models (see Sec. 6).

2 Theoretical Background of Noise

Noise is omnipresent in technical systems. However, it is not usually a physical process per se. The term noise is rather used to describe influences in a system, which are not covered by the model of the system itself.

These influences can e.g. include rough road conditions in vehicle dynamics, wind in aerodynamics or electrical radiation compromising sensor readings. In any case, a suitable model for such influences must be found in order to account for them in simulation, control and signal processing applications.

A common model for noise is *white noise* (see e.g. [1, p. 19]). It is described by a stream of random variables $w(t)$ dependent on the time t. The main property of such white noise is that it has a flat power spectrum, i.e. that all frequencies contribute equally to the noise signal. In a statistical sense, this means that all instances of the random signal $w(t_1)$ and $w(t_2)$ are uncorrelated:

$$Cov(w(t_1), w(t_2)) \stackrel{!}{=} 0 \quad \forall t_1 \neq t_2. \tag{1}$$

The second property of white noise is that the probability distribution W of the signal's values is equal for all time instants:

$$W(t_1) \sim W(t_2) \quad \forall t_1, t_2. \tag{2}$$

However, the actual distribution W is not specified by the term white noise and must be specified additionally. Gaussian white noise e.g. refers to the case that the signal is normally distributed.

In actual applications, noise is very rarely white but can be specified with a given power spectrum. In aerodynamics e.g. the von Kármán spectrum is used to model turbulence and in signal processing the noise is usually low-pass filtered and sampled. Such signals are referred to as colored noise.

Especially for numerical simulations, this band-limitation of natural noise is very convenient, because signals with infinite frequency contributions cannot be simulated. Using the above observation, we can correctly reflect the influence of natural noise on a simulated system by specifying a distribution of the random process and a suitable filter. The sampling rate of the raw noise can then be chosen just high enough to generate all contributions to the desired spectrum.

Figure 2 illustrates the properties of natural noise from a mechanical test-rig entering a continuous system. The noise is sampled with 6 kHz and passed on as a continuous-time sample-and-hold signal. The power spectral density (PSD) is estimated by oversampling the signal at 50 kHz and using Welch's method as implemented in the Matlab signal processing toolbox.

The noise appears approximately normally distributed. We can see that the noise does not have the flat power spectrum specified for white noise. The power spectrum shows a strong influence of the initial sampling with 6 kHz. In the following sections, we will discuss how to numerically generate a representative signal of such nature.

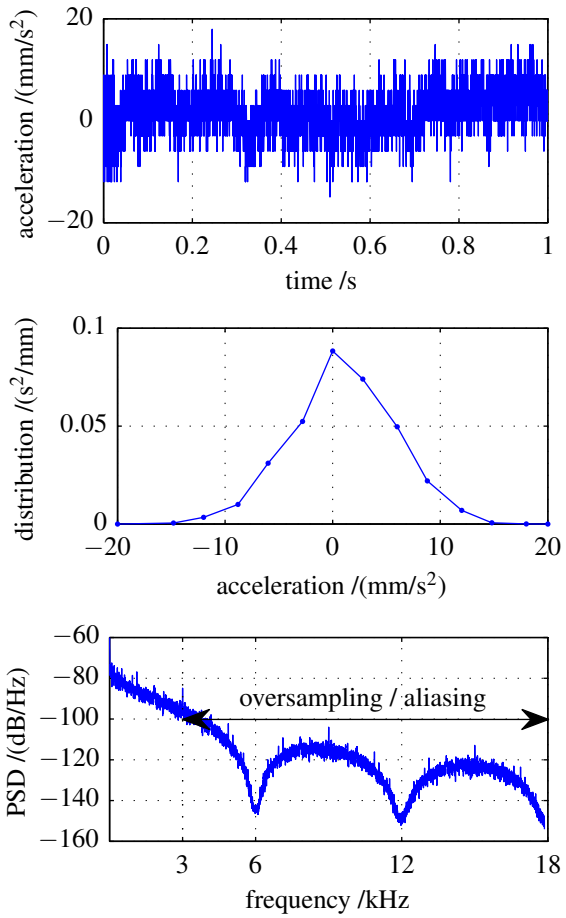


Figure 2: A noise sample from a static mechanical acceleration sensor is approximately normally distributed and has a characteristic power spectrum.

3 Noise Generation

In order to conveniently generate a realistic noise for use in continuous system simulations, a couple of criteria must be met:

a. The noise should be *realistic* with respect to the specifications given in Section 2: The signal should be uncorrelated, match a specified distribution and also match the specified frequency content.

b. A *clear and modular* approach should provide support in independently specifying these properties of the desired noise signal.

c. It should be possible to evaluate the noise signal *continuously* without the need to incrementally increase the internal states as is the case with conventional noise generators.

DOI 10.3384/ECP14096837

To comply to the specifications of above, the process is split in three steps:

1. A *random number generator* (RNG) implements an algorithm to generate a sequence of uncorrelated, uniformly distributed random numbers U_i.
2. These are transformed according to the same *probability density function* (PDF) for each discrete time instance separately to yield random numbers X_i with a specified distribution.
3. The random numbers are filtered to match a *power spectral density* (PSD) specified in the frequency domain. This results in a continuous-time noise signal $r(t)$ with the desired characteristics.

3.1 Uniform Random Number Generators

Truly random numbers are difficult to generate in numerical simulations. Fortunately, they are typically not desired to be truly random, because simulations should be repeatable and thus deterministic. Pseudo-random numbers are thus usually used, which are deterministic but appear random to the simulated system.

Pseudo-random numbers are usually generated computationally using recursive arithmetic generators. These generate random numbers Y_i recursively based in the last random number Y_{i-1}. The initial value Y_0 is known as the seed of the recursive generator. This discrete state Y_i of the random number generator must be advanced incrementally, which limits the time steps taken by an integrator to be smaller than the generator's update period.

One of the simplest recursive arithmetic generators is the linear congruential generator (LCG). It uses the parameters a, b and m to generate random integers Y_i in the interval $[0, m-1]$ and uniformly distributed numbers U_i according to the following formulas:

$$Y_i = (a \cdot Y_{i-1} + b) \mod m, \tag{3}$$

$$U_i = Y_i / m. \tag{4}$$

Better implementations of recursive arithmetic generators are e.g. the algorithms of the WELL family (Well Equidistributed Long-period Linear) [2], which feature much larger repetition periods.

In order to avoid the performance limitation introduced by the discrete state of recursive generators, we propose to use non-recursive arithmetic generators for generating random numbers. These implement a pure function $Y_t(t)$, which is solely dependent on the simulation time t. This allows to evaluate random numbers deterministically in continuous time without using discrete states.

In this work, we introduce the new random number generator **DIRCS** **I**mmediate **R**andom with **C**ontinuous **S**eed. It relies on the quick recovery of LCGs from a poor (i.e. small, non-random) seed Y_0, a property called diffusion capacity. If a poor seed Y_0 is chosen, an LCG will irrespectively generate high quality random numbers after a few iterations. This property allows to continuously seed an LCG with a very simple function of the time t, apply a few iterations and treat the resulting number as random, i.e.

$$Y_t(t) = LCG(...(LCG(\ seed(t)\ ))). \tag{5}$$

The quality of the generated random numbers U_i can be investigated using a number of different measures. All these measures quantify the fulfillment of the requirements that the random number must be (a) uniformly distributed and (b) uncorrelated for different time lags as requested by Eqs. 1 and 2. Fig. 3 confirms the LCG's diffusion capacity for small seeds between 1 and 200.

The uniformity of the distribution is checked here with the Kolmogorow-Smirnow test. The correlation is tested with a two-sided Z-test of the correlation. The given p-values indicate the confidence in the assumptions on a scale from 0 to 1. p-values larger than 0.10 indicate that the property under test is confirmed.

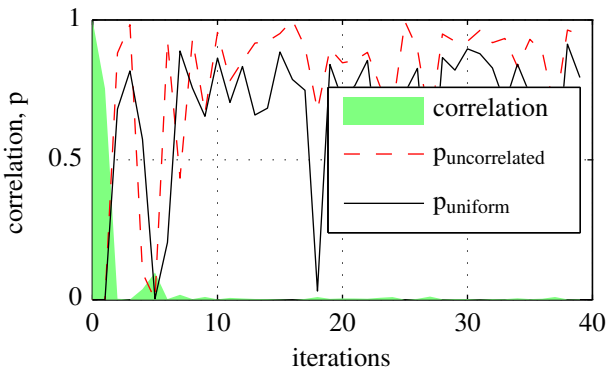


Figure 3: After ten iterations, the random numbers from an LCG can be assumed to be uniformly distributed and uncorrelated with the seed.

Table 1 gives an overview of the quality of the most important random number generators used in this work. They are compared to two standard solutions from the `Modelica_LinearSystems2` library [3] and the `Design.Experimentation` library [4]. It can be seen that all generators produce uncorrelated uniform random numbers.

Table 1: All RNGs produce uniform and uncorrelated random numbers.

Generator	Uniform	Uncorrelated
WELL1024a	$p = 0.396$	$p = 0.405$
LCG	$p = 0.456$	$p = 0.253$
DIRCS	$p = 0.432$	$p = 0.523$
LinearSystems2	$p = 0.508$	$p = 0.373$
Design	$p = 0.305$	$p = 0.899$

3.2 Probability Density Functions

Each uniform random number U_i generated with the methods presented in the previous section must be transformed to match the desired noise characteristics. The first step is to apply a mapping

$$U_i \mapsto X_i \tag{6}$$

according to a probability density function (PDF) $f(x)$. This yields random numbers X_i with a specified distribution. Informally, the function $f(x)$ describes the probability that a random variable X_i equals x.

There are several methods to achieve this step (see e.g. [5]). If an analytic PDF is given, then often its primitive $F(x) = \int_{-\infty}^{x} f(\tilde{x})d\tilde{x}$ can be derived. This cumulative density function (CDF) describes the probability that the random variable X_i is smaller or equal to x. Using the inverse of the CDF, a random number with the given distribution can be analytically calculated by $X_i = F^{-1}(U_i)$. This method is illustrated below for the heavy-tailed Cauchy-Lorentz distribution with the location parameter μ and the scale parameter γ.

$$f_{\text{Cauchy-Lorentz}}(x) = \frac{1}{\pi}\left(\frac{\gamma}{(x-\mu)^2+\gamma^2}\right) \tag{7}$$

$$F_{\text{Cauchy-Lorentz}}(x) = \frac{1}{\pi}\arctan\left(\frac{x-\mu}{\gamma}\right)+\frac{1}{2} \tag{8}$$

$$F^{-1}_{\text{Cauchy-Lorentz}}(U_i) = \gamma\tan\left(\pi\left(U_i-\frac{1}{2}\right)\right)+\mu \tag{9}$$

$$= X_{i,\text{Cauchy-Lorentz}} \tag{10}$$

If an analytical inverse of the CDF cannot be derived, different methods have to be employed. A prominent example is the normal distribution, which does not have an analytical CDF. To generate a normally distributed random variable, we employ the Box-Muller transform [6]. It uses two uniform random numbers U_{1i} and U_{2i} to calculate X_i according to

$$X_{i,\text{normal}} = \mu + \sigma\sqrt{-2\ln U_{1i}}\cos\left(2\pi U_{2i}\right). \tag{11}$$

Finally, if no direct transformation is given in the literature, some common distributions can also be calculated directly according to their definition. The Bates distribution e.g. describes the distribution of an average over n uniform random numbers and can be calculated directly from

$$X_{i,\text{Bates}} = \frac{1}{n}\sum_{k=1}^{n} U_{k\cdot i}. \tag{12}$$

3.3 Power Spectral Densities

In the previous sections, we have shown how to generate a discontinuous sequence of random variables X_i at an arbitrary sampling rate Δt. This sequence has to be processed further, in order to shape a continuous signal $r(t)$ with a specified frequency content. A common method is to apply a simple low-pass filter to the noise sequence. Although being common practice, this method has two main disadvantages:

1. The simulation speed is limited by the high sampling frequency of the noise signal, the small time-constant of the low-pass filter or both.

2. It is often unclear which statistical properties the filtered signal has. It often differs widely from the PDF of the discrete signal.

For many applications it is thus favorable to compute the continuous signal directly out of the discrete sequence without using dynamic states as in usual implementations of low-pass filters. In this work, we continuously interpolate the discrete sequence using kernel functions.

If a kernel function $K(t)$ is given, then the interpolation can be expressed as a linear combination of the kernel function with different weights X_i and offset $i\Delta t$:

$$r(t) = \sum_i X_i \cdot K(t - i\Delta t). \tag{13}$$

In order for this sum to be a proper interpolation, the kernel must equal 1 at the origin and 0 at all multiples of Δt:

$$K(i\Delta t) \overset{!}{=} \begin{cases} 1 & \text{for } i = 0 \text{ and} \\ 0 & \text{for } i \neq 0. \end{cases} \tag{14}$$

A prominent kernel function, which fulfills these constraints, is the *hat* function used to create a linear interpolation.

$$K_{\text{hat}}(t) = \begin{cases} 1 - \left|\frac{t}{\Delta t}\right| & \text{if } t \in [-\Delta t, +\Delta t] \\ 0 & \text{else} \end{cases} \tag{15}$$

DOI 10.3384/ECP14096837

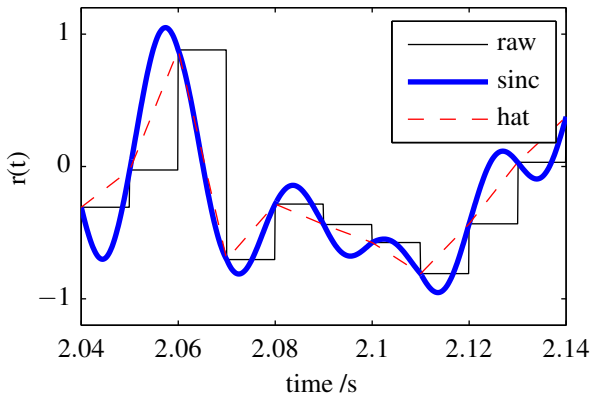


Figure 4: Different interpolations can be applied to the random sequence. The sinc interpolation achieves a very smooth signal $r(t)$.

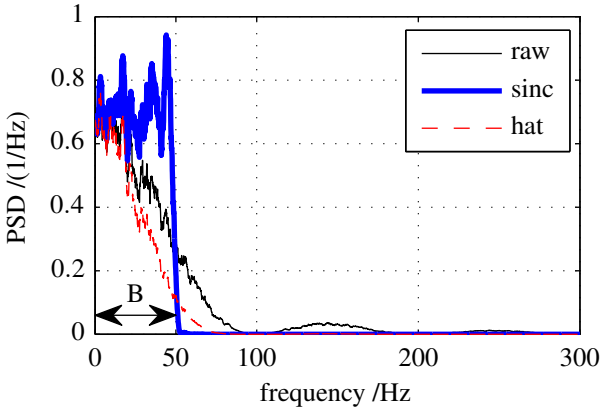


Figure 5: The frequency characteristic are shaped by the interpolation function. The sinc function achieves a very good low-pass characteristic.

The resulting signal $r(t)$ is a linear combination of the kernel functions. Its frequency content is thus fully determined by the frequency content of the kernel function. If the sum is truncated by limiting the number of involved sampling points X_i, additional content is introduced by the discontinuous support. In practice, however, the truncated contributions are often negligible, leading to acceptable approximations. This especially holds true for the hat function, which can be well truncated to only include two random samples.

Another important kernel function is the *normalized sinc* function:

$$K_{\text{sinc}}(t) = 2B\text{sinc}(2Bt) = \frac{\sin(2B\pi t)}{\pi t}. \tag{16}$$

It only contains frequencies up to its bandwidth B. If $B = 1/(2\Delta t)$ is chosen to match half the sampling rate, the normalized sinc function also fulfills the constraints of an interpolation kernel. Interpolation with the sinc function can thus be used to apply an optimal low-pass filter to the random sequence.

Figure 4 shows the different interpolation methods described above. The interpolation is applied to a random sequence X_i generated with 100 S/s. The frequency contents of the signals are compared in Fig. 5. The raw sample-and-hold signal of X_i contains frequencies higher than the sampling frequency. Using the interpolation with the sinc function, the frequency characteristic can be nicely limited to the desired cut-off frequency of half the sampling frequency. It is important to understand how the interpolation affects the statistical properties of the random signal $r(t)$. Since $r(t)$ is a weighted sum of statistically independent random variables, we can use the following two laws to determine the change of variance for any valid filter:

$$\text{Var}(X_i \cdot c) = \text{Var}(X_i) \cdot c^2, \tag{17}$$

$$\text{Var}(X_i + X_j) = \text{Var}(X_i) + \text{Var}(X_j). \tag{18}$$

Here, X_i and X_j denote the statistically independent random variables generated by the RNG and c denotes a constant weight. The variance of the random variables is fixed by the selected PDF and the constant c is given by the interpolation kernel. The time-dependent variance of $r(t)$ from Eq. 13 can thus be expressed as

$$\text{Var}(r(t)) = \text{Var}(X_i) \cdot \sum_i \left(K(t - i\Delta t)\right)^2. \tag{19}$$

Due to the constraints for interpolation kernels Eq. (14), the variance of $r(t)$ at the sampling points is equal to the variance of the random variable X_i. In between the sampling points, the variance of the random signal is a function of the time. We can compute the expectation value of the variance for the entire signal by formulating the integral over the interval Δt:

$$\begin{aligned} E\left[\text{Var}(r(t))\right] &= \frac{1}{\Delta t} && \int_{t=0}^{\Delta t} \text{Var}(r(t))\, dt \\ &= \frac{\text{Var}(X_i)}{\Delta t} && \int_{t=0}^{\Delta t} \sum_{i=-n+1}^{n} \left(K(t - i\Delta t)\right)^2 dt \\ &= \frac{\text{Var}(X_i)}{\Delta t} && \sum_{i=-n+1}^{n} \int_{t=0}^{\Delta t} \left(K(t - i\Delta t)\right)^2 dt \\ &= \frac{\text{Var}(X_i)}{\Delta t} && \int_{t=-n\Delta t}^{n\Delta t} K(t)^2\, dt. \end{aligned} \tag{20}$$

Doing this computation for the linear interpolation using the hat function leads to

$$E\left[\text{Var}(r(t))\right] = \frac{2}{3}\text{Var}(X_i). \tag{21}$$

Thus, any linear interpolation reduces the variance of the input signal by one third, independently of the PDF used to generate X_i. In a similar manner, the result can be computed for the optimal bandwith limitation. Fortunately,

$$\lim_{n\to\infty} \frac{1}{\Delta t} \int_{t=-n\Delta t}^{n\Delta t} K_{\mathrm{sinc}}(t)^2\, dt = 1 \qquad (22)$$

and hence the variance is only minorly affected for n chosen large enough. In fact, with $n = 3$, the expected variance is only changed by less than 5 %.

Not only the variance may be affected but also the shape of the PDF. It is possible to compute this effect for specific PDFs but the most general statement can be drawn from the central limit theorem. It states that a sum of many independent random variables (with finite variance) is approximately normally distributed. Hence we can state that the application of a filter transforms all PDFs with finite variance gradually to look like the normal distribution and that this effect is expected to be stronger the wider the domain of $K(t)$ is.

Our analysis suggests a very suitable combination for generating a continuous random noise signal: if the discrete noise is generated by a normal distribution and if it is interpolated by an ideal bandwith limitation, the resulting signal is also of a normal distribution with the same variance and a frequency characteristic below the cut-off frequency that represents white noise. The cut-off frequency will be half the sampling frequency.

4 Implementation in Modelica

The concepts laid out in Sec. 3 are implemented in the Modelica `Noise` library. An overview of the library's components is shown in Fig. 6. The library provides a single block `PRNG` as the interface to all of its facilities. This block is described here.

In order to provide the user with a convenient interface, the `PRNG` block combines all parts of the noise generation process described in Sec. 3. The block's parameter pane is shown in Fig. 8.

All three parts of the noise generation can be (almost) independently parametrized. The *Random Number Generator (RNG)* parameters allow the user to select whether a sampling-based or a sampling-free generator shall be used. Two selectors allow to maintain parametrized instances of both variants in the `PRNG` block. The parameters of the generators can all be set via the `PRNG`'s parameter interface. The criteria listed in Table 1 may assist in choosing a suitable RNG.

The *Probability Density Function (PDF)* can also be selected in the `PRNG`'s parameter pane along with its parameters. Some of the available distributions are shown in Fig. 7. Cauchy-Lorentz, Irwin-Hall, Bates and Discrete distributions are also available. The distribution should be selected according to the specification of the desired noise. Additional distributions can be implemented according to the literature by filling a common function interface.

Finally, a kernel function for the desired *Power Spectral Density (PSD)* can be selected and parametrized. The kernels are set up with standard parameters to match the update rate of the block. For example, the ideal low-pass filter is set up with a cut-off frequency of half the update rate. The implemented filters are shown in Fig. 9. A typical choice of the PSD would be to use the raw random sequence or the ideal low-pass interpolation with $n \geq 5$. Additional kernels can be added easily by the user.

The *(Pseudo-) Sampling* parameters of the `PRNG` block are the same as for most sampling Modelica blocks. A `startTime` determines the first sample to be generated. A `samplePeriod` determines the step-size of the sampler. For sample-free generators, the sampling is relaxed to a pseudo-sampling of the random number, i.e. imposing an upper limit to the update rate. An additional `infiniteFreq` switch can be used to completely disable (pseudo-)sampling and let the simulation tool handle the step-size of the generator based on the desired integration accuracy. This also disables the PSD filtering, because the interpolation needs discrete noise samples.

The *Enable/Disable* parameters finally are used to enable noise generation or to output a simple dummy variable `y_off`. It can be used to parametrically restore the ideal system without noise.

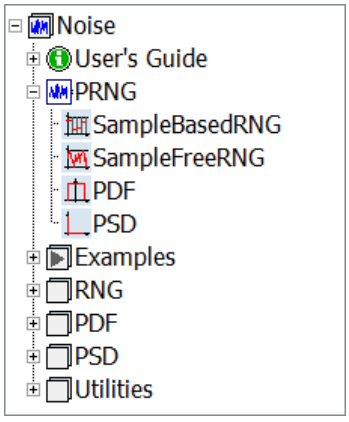


Figure 6: The `Noise` library provides a ready-to-use noise block as well as a collection of `RNG`, `PDF` and `PSD` implementations.

DOI 10.3384/ECP14096837

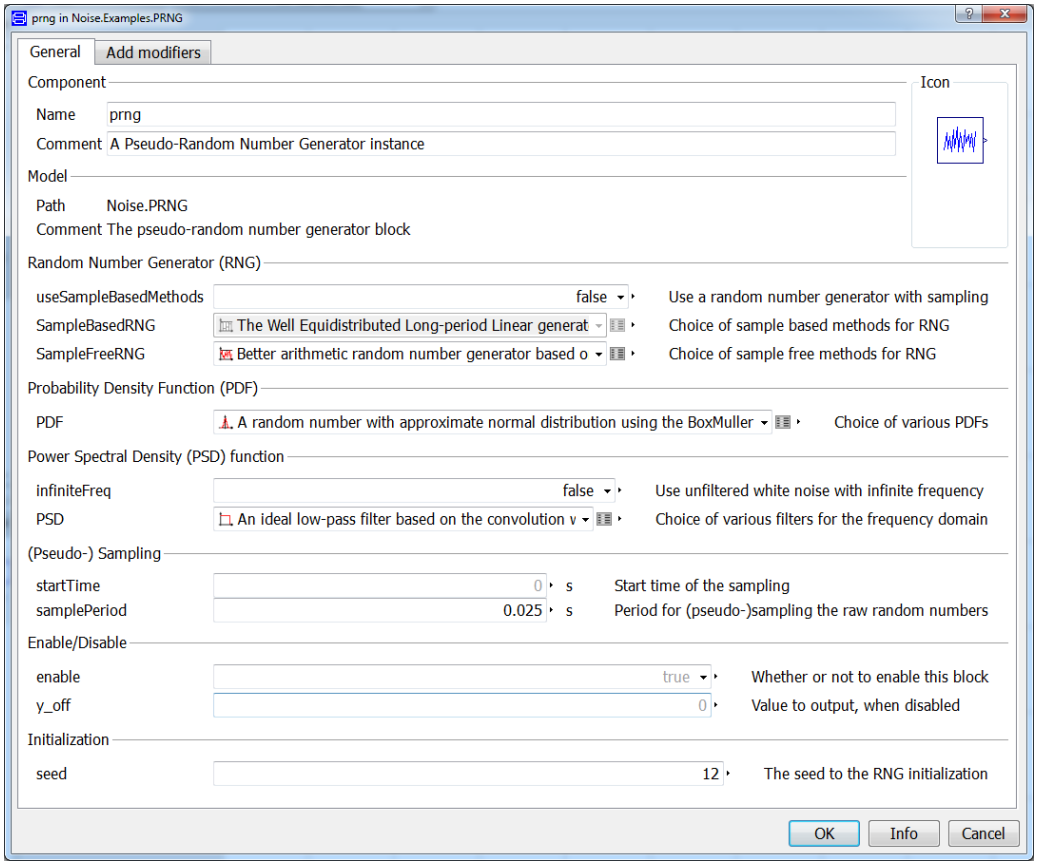
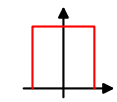
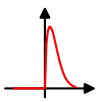


Figure 8: The `PRNG` block can be used to collectively set up custom noise by modularly combining all available `RNG`, `PDF` and `PSD` implementations. Additional switches are provided to set the `samplePeriod`, enable sample-free `RNG`s and enable infinite frequency emulation.



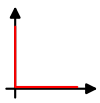
(a) Uniform

(b) BoxMuller

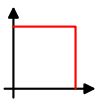


(c) Weibull

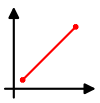
Figure 7: Some of the provided `PDF`s are a uniform, a normal and a Weibull distribution.



(a) Raw



(b) IdealLowPass



(c) LinearInterp

Figure 9: The provided `PSD`s include an unfiltered white noise, an ideal low-pass filter and a linear interpolation.

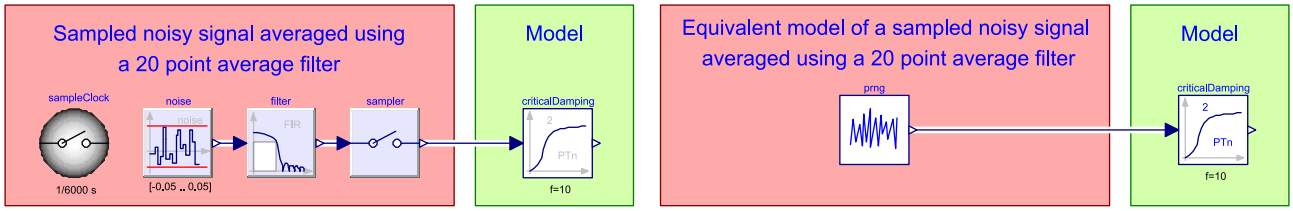
5 Evaluation of Filtered Noise

Relevant noise in realistic applications is in most cases sampled and filtered (see e.g. Fig. 2). The process of sampling and filtering changes the probability density as well as the frequency content of the signal. In the following sections, we will show how these characteristics can be modeled using the `Noise` library. First, a synthetic example is used.

In this section, the probability distribution and frequency content of noise generated with a standard approach and with the `Noise` library are compared. A digital sensor is used as an example. The sensor has a uniform noise distribution with amplitudes between -0.05 rad and 0.05 rad. The signal is sampled with 6 kS/s. The signal is subsequently smoothed with a running mean filter using 20 samples and then down-sampled by a factor of 20. This procedure is representative for a typical angular resolver signal used for control purposes. In order to introduce a model of a simple system, a `CriticalDamping` block from the Modelica standard library is used. It has a fixed cut-off frequency of 10 Hz and a variable number of states.

5.1 Model using the `LinearSystems` library

In Fig. 10a, the reference implementation using the `Modelica_LinearSystems` library is shown. Uniform noise is generated with at a rate of 6 kS/s, filtered with a running mean FIR filter and then down-sampled.



(a) `LinearSystems` (b) `Noise`

Figure 10: Generation of realistic filtered noise using the `LinearSystems` and the new `Noise` library.

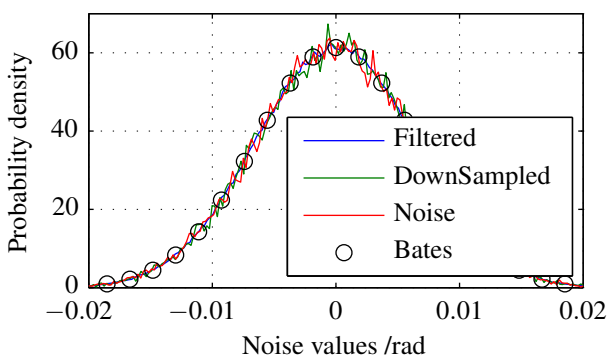


Figure 11: Histograms of the different noise signals

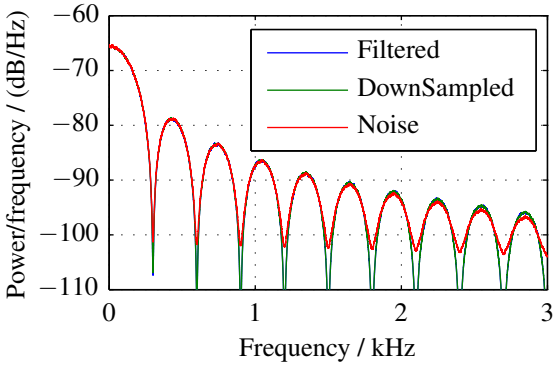


Figure 12: Frequency analysis of the noise signals.

5.2 Model using the `Noise` library

Using the `Noise` library, it is possible to generate the filtered and down-sampled noise signal without sampling at a high frequency. In order to achieve an equivalent signal, the PDF must be adjusted to match the distribution of the filtered signal. This is done selecting the Bates distribution described in Sec. 3.2 with $n = 20$. The noise signal can then be generated at the down-sampled update rate of 300 S/s. Additionally, the DIRCS generator is used to suppress all time-events in the simulation. The Modelica block diagram is shown in Fig. 10b.

5.3 Probability density distribution

The empirical probability density functions of the generated noise signals are shown in Fig. 11. To show that down-sampling the signal has no influence on the PDF, the down-sampled, as well as the original signals are shown. The probability density function of the generated noise signal is obtained experimentally by simulating both noise models for 200 s and by using 250 bins between the minimal and maximal values (-0.05 to 0.05). The results of the analysis show a good match between the different signals.

5.4 Frequency content

The frequency content of the signal is an important property of a noisy signal. To show that also the frequency content of the PRNG block mimics the frequency content of the sampled and filtered signal Welch's power spectral density analysis is applied to the signals (see Sec. 2). The results are shown in Figure 12. The plot shows that the frequency contents of all signals correspond very well. Especially the low frequency content up till 300 Hz is a very good match.

5.5 Simulation times

In order to compare the simulation times of the presented models, the models are simulated again for 200 s. As a reference, a standard noise block from the `LinearSystems` library without downsampling is included using 300 Samples/s, the same update rate as the PRNG block updates internally. Only 10 output points are generated in order to make sure the output routine does not influence the simulation times. The simulations are performed using Dymola 2014 FD01 Beta 3 on an Intel® Xeon® E5-1620 processor. The results of the simulations are summarized in Table 2. Additionally to the model structure, the number of states in the `CriticalDamping` system is varied.

The results clearly show the advantage of using the `Noise` library for modeling noise in a system. The simulation time for the `Noise` model with a system of second order is six times faster than the standard approach. If the complexity of the system is increased by using a number of 50 states, the `Noise` model is 26 times faster

DOI 10.3384/ECP14096837

Table 2: Simulation times and number of steps of the models with noise for 200 seconds simulation. `LS 6kS/s filtered` marks the Noise from Figure 10a, `LS 300S/s raw` marks the simulation results of the noise generation using the `LinearSystems` library using a direct noise output of the signal and `LS 300S/s Bates` using a Bates distribution using an averaging of 20 random samples at each timestep.

Model	States	Time	Events
`LS 6kS/s filtered`	2	45 s	12e6
`LS 300S/s raw`	2	6 s	60e3
`LS 300S/s Bates`	2	8.7 s	60e3
`Noise`	2	7.5 s	0
`LinearSystems`	50	256 s	12e6
`LS 300S/s raw`	50	23 s	60e3
`LS 300S/s Bates`	50	24.5 s	60e3
`Noise`	50	9.7 s	0

than the model using `LinearSystems` noise. The approaches without downsampling (`LS 300S/s` methods) have a better performance. The Bates distribution at 300S/s has a similar calculation time as the method using the `Noise` library. However, at higher model levels, for a system with 50 states, the speedup ratio is a little over 2. This result show that integrator restarting due to the event generation becomes more expensive at increasing system complexity.

6 Industrial Application Example

In order to test the performance of the presented noise models in an application of industrial complexity, the example of an electro mechanical actuator is chosen. This actuator is generated using the `Actuator` toolbox [7]. In Figure 13, an overview of the model is given.

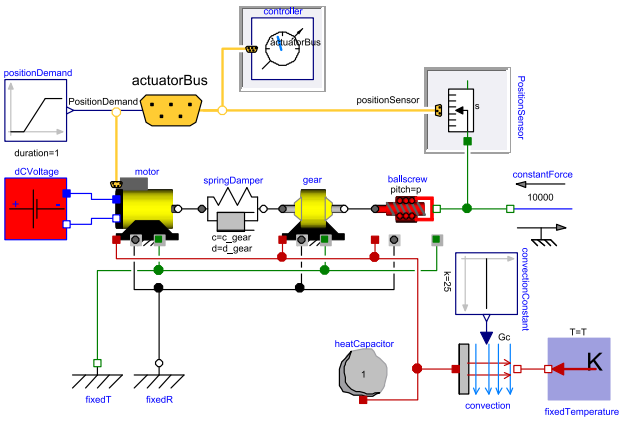


Figure 13: Actuator model overview. The motor position sensor is used to include the noise effects.

The position sensor of the motor can be chosen from an ideal sensor or two sensor versions with noise. The sensor readings are used for controlling the motor current as well as the speed and position of the actuator. To obtain the motor speed, position is differentiated. For these reasons, adding noise to the system is expected to strongly influence the system's behavior.

To test the presented noise generation and compare it with a traditional approach, two sensors with noise were derived. The first sensor uses a traditional sampled noise using the blocks from `Modelica_LinearSystems.Sampled`, the second uses the `Noise` library presented in this work. The two noise models are the same as presented in Sec. 5. The noise is assumed to be additive as shown in Fig. 1.

6.1 Simulation results

The actuator model is simulated with each of the three sensor versions. The actuator is commanded to follow a change in position of 43 mm at $t = 1$ s. Figure 14 shows the responses with the three sensor versions. The results match very well. The simulations were carried out using the Dassl integrator with a tolerance of 10^{-4}. For the following comparison, only 10 output intervals are requested in order not to influence the simulation times by the output intervals. The simulation times and generated events are summarized in Table 3.

6.2 Time and state events

One of the main benefits of the presented noise model is that no time events are generated. As expected, the sampled noise using the `LinearSystems` library generates 15000 time events, which is the product of the sample rate and the simulation time. The model built with the `Noise` library, however, does not generate any time events. The amount of state events generated by

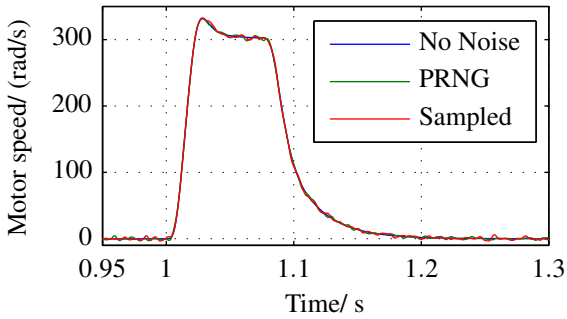


Figure 14: Resulting trajectory of the actuator using the three sensor versions. The actuator follows a commanded position change of 43 mm.

Table 3: Simulation results of the actuator models over 2.5 seconds.

		Events	
Model	Time	Time	State
No Noise	0.037 s	1	12
`LinearSystems`	27.5 s	15000	1642
`Noise`	10.5 s	1	1791

both models is roughly the same. These events are generated by the nonlinearities in the model. Mainly the inverter reacts to the high-frequency changes of the demanded current, which is generated by the noisy sensor readings.

6.3 Simulation speed

Especially in more complex models the penalty of an event becomes large. Restarting the integrator can use as much time as an integrator step itself. In Table 3, the simulation time of the models is shown. The noise itself has a big penalty on the simulation performance. This is expected, as the position sensor is used in all control algorithms. The simulation time using the `Noise` library, however, is decreased with respect to the standard implementation due to the reduced number of time events.

7 Conclusions and Outlook

We have shown how to properly implement a noise signal in Modelica by selecting a high-quality random number generator (RNG) and by specifying a probability density function (PDF) as well as a power spectral density (PSD) of the desired signal. In order to aid the user in this process, we have proposed a library with modular implementations of the three parts of the noise generation. The library provides a convenient interface to set all parameters. It is further possible for the user to freely implement new modules.

The proposed combination of sample-free RNGs and continuous PSDs provides for a satisfactory increase in simulation speed by a factor between 2.5 in a real world actuator example and up to 26 using an academic model.

Time events due to the noise model can be completely eliminated from the simulation by using the proposed continuous DIRCS random number generator, which relies on continuously seeding a standard RNG with a function of the time.

Further extensions of the library can be seen in taking advantage of the Modelica 3.3 function arguments. The library could additionally be improved in modularity by introducing separate seeding functions. In this way, also the continuously seeded generator could be modularized. Some convenient features such as global seeding or seeding based on the machine time may also be addressed in future versions. Additionally, the current interpolation filter may be extended to allow for arbitrary filter functions implemented by the user.

Acknowledgements

We thank our colleague Andreas Knoblach for his valuable comments on the present paper.

References

[1] Crispin W Gardiner. *Handbook of Stochastic Methods: for Physics, Chemistry and the Natural Sciences*. Springer Berlin, 2nd edition, 1985.

[2] François Panneton, Pierre L'ecuyer, and Makoto Matsumoto. Improved long-period generators based on linear recurrences modulo 2. *ACM Transactions on Mathematical Software (TOMS)*, 32(1):1–16, 2006.

[3] Marcus Baur, Martin Otter, and Bernhard Thiele. Modelica libraries for linear control systems. In *Proceedings of 7th International Modelica Conference, Como, Italy, September*, pages 20–22, 2009.

[4] Dassault Systèmes AB. *Dymola User's Manual – Volume 2*, 2011.

[5] Ralf Korn, Elke Korn, and Gerald Kroisandt. *Monte Carlo Methods and Models in Finance and Insurance*. Financial Mathematic Series. Chapman & Hall / CRC Press, 2010. ISBN 978-1-4200-7618-9.

[6] George EP Box and Mervin E Muller. A note on the generation of random normal deviates. *The Annals of Mathematical Statistics*, 29(2):610–611, 1958.

[7] Franciscus L. J. van der Linden, Clemens Schlegel, Markus Christmann, Gergely Regula, Christopher Hill, Paulo Giangrande, Jean-Charles Maré, and Imanol Egaña. Implementation of a Modelica Library for Simulation of Electromechanical Actuators for Aircraft and Helicopters. In *Proceedings of the 10th International Modelica Conference*, 2014.