StochBB – Stochastic Building Blocks Manual

Analyzing complex systems of dependent random variables

Hannes Matuschek

Deparment of Mathematics,
Focus area *Dynamics of complex systems*,
University of Potdsam, Germany

1 Introduction

Frequently, complex systems are described in terms of stochastic processes, as the underlying deterministic process is too complex to be modeled exactly or as the process is indeed random. It is not always the random process itself that is of interest, but a derived quantity. For example, the distribution of waiting times until the process reaches a certain state. In the field of cognitive psychology, random processes are frequently used to describe each processing stage in a chain of stages that leads to a response. The state of each random process itself is usually not measurable but the total response time of all processing stages involved. Although each processing stage is modeled as a random process, the waiting-time of a single stage is just a random variable and the complete system is then a system of dependent random variables.

StochBB is able to describe and analyze complex systems of dependent random variables by combining simple ones (representing single stages with a known waiting-time distribution) to a complex system. For example, consider the following independent random variables

$$X_1 \sim \Gamma(10, 100)\,, X_2 \sim \Gamma(20, 50) \text{ and } X_3 \sim \text{Exp}(0.01)\,,$$

which are described completely by their distribution. This means that the time, the processing stage X_1 needs to complete is gamma-distributed with shape $k = 10$ and scale $\theta = 100$. Analogously, the stages X_2 and X_3 are defined by their own waiting-time distribution.

From these basic building blocks, a more complex system can be assembled by combining them. Using the example above, one may define a new processing stage that is a chain of the stages $X_1, \ldots, X_3$. This simple chain then describes the successive processing of information entering the first stage described by X_1. Once the first processing stage finished, its result gets forwarded to the second stage described by X_2 and finally to the last stage described by X_3. The waiting time of the complete chain is again a random variable that is the sum of all random variables, or expressed mathematically

$$Y = X_1 + X_2 + X_3\,.$$

SochBB determines the probability density function (PDF) or cumulative probability function (CDF) of the waiting-time distribution of the process Y analytically (as far as possible) or resorts

to a numeric method if the analytic approach fails. More over it provides an efficient and correct sampler for the system of random variables.

2 Representation and reductions of random variables

Continuing the example above, please note that the sum of random variables commutes. Hence the random variable Y remains the same if defined as $Y = X_1 + X_3 + X_2$ instead of $Y = X_1 + X_2 + X_3$. Moreover, the distribution of the sum of X_1 and X_3 can be determined analytically as $X' = (X_1 + X_3) \sim \Gamma(11, 100)$. Hence the random variable Y can now be expressed as $Y = X_2 + X'$, and only a single numeric convolution is necessary to obtain the PDF of the random variable Y.

StochBB implements several reductions of the system of random variables, exploiting mathematical identities of random variables. To this end, it allows to obtain the PDFs and CDFs of random variables efficiently.

2.1 Affine transformations of random variables

An affine transformation of the random variable X has the form $Y = a\,X + b$, where $a \neq 0$ and b are real values. Although affine transformations of random variables are not frequently used directly in a system of random variables, they may appear as a result of other reductions of the system. The PDF and CDF of the random variable Y defined above, are then

$$f_Y(y) = \frac{1}{a} f_X\left(\frac{y-b}{a}\right) \text{ and } F_Y(y) = F_X\left(\frac{y-b}{a}\right).$$

Of course, an affine transformation of an affine transformed random variable X is also a simple affine transformation of the random variable. Hence the following reduction is implemented

$$c\,(a\,X + b) + d \longrightarrow (a\,c)\,X + (c\,b + d)\,.$$

2.2 Sums of random variables

Sums of random variables have been introduced briefly above and may represent a chain of processing stages being triggered sequentially. The sum itself is a derived random variable that depends on all mutually independent variables being summed up. The PDF of the sum Y is then the convolution of all PDFs of the summed variables. That is

$$Y = \sum_{i=0}^{N} X_i \text{ where } X_i \sim f_i(x) \text{ mutually independent}$$
$$Y \sim f_1(x) * \cdots * f_N(x)\,.$$

The direct numerical convolution of the underlying distributions can be computationally expensive if the number of PDFs is large. Assuming that all distributions are well supported on a common interval, however, allows for a fast numerical convolution by means of FFT convolution

[3]. For the FFT convolution, all densities $f_i(x)$ are evaluated on a common regular gird and the evaluation of the convolution on that gird can then be computed easily. This, however, requires that the grid is chosen such that all densities being convoluted as well as the result are well supported on the chosen interval. Moreover, the grid must be fine enough to capture the details of all distributions.

The numerical convolution, like any numerical approach, is only an approximation. Hence StochBB tries to perform the convolutions analytically before resorting to the numerical approach.

First all sums of random variables are flattened. That is,

$$\begin{array}{l} Y_1 = X_1 + X_2 \\ Y_2 = Y_1 + X_3 \end{array} \longrightarrow \begin{array}{l} Y_1 = X_1 + X_2 \\ Y_2 = X_1 + X_2 + X_3 \end{array} .$$

Then, the distribution of the sum is derived. Here the following reductions are performed.

$$\begin{aligned} \delta(x - x_0) * U[a, b](x) &\longrightarrow U[a + x_0, b + x_0](x) , \\ \delta(x - x_0) * \phi(x; \mu, \sigma) &\longrightarrow \phi(x; \mu + x_0, \sigma) , \\ \phi(x; \mu_1, \sigma_1) * \phi(x; \mu_2, \sigma_2) &\longrightarrow \phi(x; \mu_1 + \mu_2, \sqrt{\sigma_1^2 + \sigma_2^2}) , \\ \Gamma(x; k_1, \theta) * \Gamma(x; k_2, \theta) &\longrightarrow \Gamma(x; k_1 + k_2, \theta) , \end{aligned}$$

where $\delta(\cdot - x_0)$ is the delta distribution located at x_0, $U[a, b](\cdot)$ the uniform distribution on the interval $[a, b]$, $\phi(\cdot; \mu, \sigma)$ the normal distribution with mean μ and standard deviation σ and $\Gamma(\cdot; k, \theta)$ the gamma distribution with shape k and scale θ.

2.3 Minimum and maximum of random variables

The minimum or maximum of some given random variables, e.g. $Y = \min\{X_1, X_2\}$ or $Y = \max\{X_1, X_2\}$ are them self random variables. These derived random variables can be used to express the waiting time of a processing stage that consists of several independent processes being performed in parallel (in contrast to sequential processes described by sums of random variables). If the processing stage finishes once the first of the parallel processes finishes, the resulting waiting time can be expressed by the minimum of the underlying random variables. If the stage finishes once all underlying processes are finished, the resulting waiting time can be described by the maximum.

If the two independent random variables X_1 and X_2 are distributed according to the (cumulative) distribution functions $F_1(x)$ and $F_2(x)$, the maximum of these two random variables $Y = \max\{X_1, X_2\}$ is distributed according to the probability function $F_Y(y) = F_1(y) \cdot F_2(y)$. Consequently, its PDF is $f_Y(y) = f_1(y) \cdot F_2(y) + f_2(y) \cdot F_1(y)$, where $f_1(\cdot)$ and $f_2(\cdot)$ are the PDFs of the random variables X_1 and X_2. Likewise, the CDF and PDF of the minimum can be expressed as $F_Y(y) = 1 - (1 - F_1(y)) \cdot (1 - F_2(y))$ and therefore $f_Y(y) = f_1(y)(1 - F_2(y)) + f_2(y)(1 - F_1(y))$.

For applying these equations to derive the CDF and PDFs of the minimum and maximum random variables, the underlying random variables must be independent. To achieve independence where possible, the following reductions are performed on maximum as well as minimum random variables.

Again, first maximum and minimum structures were flattened, for example

$$\begin{array}{l} Y_1 = \max\{X_1, X_2\} \\ Y_2 = \max\{Y_1, X_3\} \end{array} \longrightarrow \begin{array}{l} Y_1 = \max\{X_1, X_2\} \\ Y_2 = \max\{X_1, X_2, X_3\} \end{array} ,$$

then, possible common terms are collected like

$$\max\{X_1 + X_2, X_3 + X_2\} \longrightarrow \max\{X_1, X_3\} + X_2 \,.$$

The latter transformation does not ensure independence of random variables, but decreases the complexity of setting-up a complex system of random variables by resolving simple-structured dependencies between random variables.

2.4 Mixture of random variables

A mixture is the weighted sum of random variables

$$Y = \frac{w_1 X_1 + \cdots + w_N X_N}{w_1 + \cdots + w_N} \,,$$

where w_i are positive weights. The result Y is also a random variable with the PDF

$$f_Y(y) = \frac{w_1 f_1(x) + \cdots + w_N f_N(x)}{w_1 + \cdots + w_N} \,,$$

where $f_i(\cdot)$ is the PDF of the i-th random variable X_i. The CDF of the mixture is obtained analogously. A mixture can be used to describe a random path-selection in a system of processing stages.

2.5 Conditional random variables

The *conditional* random variable selects one of two random variables (e.g. Y_1 or Y_2) depending on the condition $X_1 < X_2$. That is

$$Z = \begin{cases} Y_1 & \text{if } X_1 < X_2 \\ Y_2 & \text{else} \,, \end{cases}$$

where X_1, X_2, Y_1 and X_1, X_2, Y_2 are mutually independent random variables. The variables Y_1 and Y_2 do not need to be mutually independent. Given that the possible result variables Y_1 and Y_2 are independent from the condition, the result variable is then a simple mixture where the weight is given by the probability of $X_1 < X_2$. Hence the density of the result variable Z is

$$\begin{aligned} f_Z(z) &= Pr[X_1 < X_2]\, f_{Y_1}(z) + (1 - Pr[X_1 < X_2])\; f_{Y_2}(z) \\ &= f_{Y_1}(z) \int_{-\infty}^{\infty} F_{X_1}(x)\, f_{X_2}(x)\, dx + f_{Y_2}(z) \int_{-\infty}^{\infty} F_{X_2}(x)\, f_{X_1}(x)\, dx \,. \end{aligned}$$

2.6 Conditional chained random variables

One of the few non-textbook examples of a derived random variable is the conditional chained random variable. It can be defined as

$$Z = \begin{cases} X_1 + Y_1 & \text{if } X_1 < X_2 \\ X_2 + Y_2 & \text{else}\,, \end{cases}$$

where X_1, X_2, Y_1 and X_1, X_2, Y_2 are mutually independent. Y_1 and Y_2 may be dependent random variables. Although being similar to the conditional random variable, there is an important difference: Both possible outcomes (X_1+Y_1 and X_2+Y_2) are not independent from the condition (i.e. X_1+Y_1 depends trivially on X_1). Therefore, the simple conditional random variable cannot be used here.

The conditional chained random variable can be used to describe two independent parallel processing stages where the fastest stage will trigger another stage. For example, if X_1 wins, it triggers Y_1 and if X_2 wins it triggers Y_2. In contrast to the conditional chained random variable, the conditional random variable does not trigger a third stage but *selects* the value of a third stage. Therefore, the actual waiting time of the *winning* stage does not have an influence on the selected one and may lead to non-causal waiting times if X_1 and X_2 are larger than Y_1 and Y_2 (i.e., the response is faster than the processes selecting the response). The conditional chained random variable always maintains causality.

The conditional chained random variable is certainly not common and to my knowledge not covered in text books. Hence the density for it must be obtained first. Given that the cases ($X_1 < X_2$ and $X_2 < X_1$) are mutually exclusive, the density of Z, $f_Z(z)$ can be expressed as a simple sum. Precisely

$$\begin{aligned} f_Z(z) = \iiint_{-\infty}^{\infty} f(x_1, x_2, y_1 | z = x_1 + y_1, x_1 < x_2)\, dx_1\, dx_2\, dy_1 \\ + \iiint_{-\infty}^{\infty} f(x_1, x_2, y_2 | z = x_2 + y_2, x_2 < x_1)\, dx_1\, dx_2\, dy_2\,. \end{aligned}$$

Given that X_1, X_2 and Y_1 are mutually independent, the first integral can be reduced to

$$\begin{aligned} &\iiint_{-\infty}^{\infty} f(x_1, x_2, y_1 | z = x_1 + y_1, x_1 < x_2)\, dx_1\, dx_2\, dy_1 \\ &\quad = \iiint_{-\infty}^{\infty} f_{X_1}(x_1)\, f_{X_2}(x_2)\, f_{Y_1}(y_1)\, \delta(z - x_1 - y_1)\, H(x_2 - x_1)\, dx_1\, dx_2\, dy_1 \\ &\quad\quad = \iint_{-\infty}^{\infty} f_{X_1}(x_1)\, f_{X_2}(x_2)\, f_{Y_1}(z - x_1)\, H(x_2 - x_1)\, dx_1\, dx_2 \\ &\quad\quad = \int_{-\infty}^{\infty} f_{X_1}(x_1)\, f_{Y_1}(z - x_1) \int_{-\infty}^{\infty} f_{X_2}(x_2)\, H(x_2 - x_1)\, dx_2\, dx_1 \\ &\quad\quad\quad = \int_{-\infty}^{\infty} f_{X_1}(x_1)\, f_{Y_1}(z - x_1) \int_{x_1}^{\infty} f_{X_2}(x_2)\, dx_2\, dx_1 \\ &\quad\quad\quad\quad = \int_{-\infty}^{\infty} f_{X_1}(x_1)\, \left(1 - F_{X_2}(x_1)\right)\, f_{Y_1}(z - x_1)\, dx_1\,. \end{aligned}$$

Analogously, the second integral can be reduced to

$$\iiint_{-\infty}^{\infty} f(z, x_1, x_2, y_2 | z = x_2 + y_2, x_2 < x_1)\, dx_1\, dx_2\, dy_2 \\ = \int_{-\infty}^{\infty} f_{X_2}(x_2)\,(1 - F_{X_1}(x_2))\; f_{Y_2}(z - x_2)\, dx_2\,.$$

The final density of Z is then given by

$$f_Z(z) = \int_{-\infty}^{\infty} f_{X_1}(x_1)\; (1 - F_{X_2}(x_1))\; f_{Y_1}(z - x_1)\, dx_1 \\ + \int_{-\infty}^{\infty} f_{X_2}(x_2)\,(1 - F_{X_1}(x_2))\; f_{Y_2}(z - x_2)\, dx_2\,, \\ = [f_{X_1}\; (1 - F_{X_2})] * f_{Y_1} + [f_{X_2}\,(1 - F_{X_1})] * f_{Y_2}\,,$$

and it can be evaluated using the same *trick* used for chains of random variables.

2.7 Compound random variables

The most complex derived random-variable type is the compound random variable. That is a random variable X distributed according to a parametric distribution $X \sim f_{X|A}(x; A)$ with parameter A. A, however, is itself a random variable with its own distribution $A \sim g(a)$. The distribution of the compound random variable X is then obtained by marginalizing the parameter of the PDF $f_{X|A}(\cdot; a)$ as

$$f_X(x) = \int f_{X|A}(x; a)\, g(a)\, da\,,$$

and the CDF is obtained analogously as

$$F_X(x) = \int F_{X|A}(x; a)\, g(a)\, da\,.$$

StochBB determines the PDF and CDF of X by performing the integral numerically.

3 Application programming interface

The application programming interface (API, see also [2]) allows to assemble complex systems of random variables programmatically in C++. All API classes are derived from the `Container` class which is an essential part of the memory management system used by StochBB. Usually a C++ programmer needs to keep track of all objects still in use and is responsible to free unneeded objects to avoid memory leaks. This can be a difficult task when dealing with complex structured objects cross-referencing each other. To ease the usage of StochBB, a *mark and sweep* garbage collector is implemented which keeps track of all objects being directly or indirectly reachable and freeing all unreachable objects. For this memory management system to work, it is necessary to treat all container objects like values although they represent references to objects allocated on the heap.

There are also Python and R packages (called `stochbb` too) that provide access to the classes and functions of the C++ API. This allows for a convenient construction of systems of random variables while maintaining the speed of the C++ implementation. These APIs are almost identical to the C++ one that R and `numpy` array are used instead of Eigen matrices and vectors.

The central class of StochBB is `Var`, representing a random variable. This could be a simple random variable having a specified distribution (`AtomicVar`) or a random variable that is derived from others like `Chain`, `Minimum`, `Maximum` or `Mixture`. All random variables (atomic and derived) have probability density functions attached. They can be accessed using the `Var::density` method which returns a `Density` object.

All `Density` objects have two methods, `Density::eval` evaluating the probability density function and `Density::evalCDF` evaluating the cumulative density or probability function. Assembling a system of random variables and evaluate their PDFs or CDFs is straight forward. Sampling, however, is not that trivial and is described below in some detail.

3.1 Assembling a system of random variables

In a first step, one may define a new gamma-distributed random variable with shape $k = 10$ and scale $\theta = 100$ as

```
#include <stochbb/api.h>
using namespace stochbb;

// [...]

Var X1 = gamma(10, 100);
```

Its PDF can then be evaluated as on a regular grid in $[0, 1000)$ with 1000 grid points as

```
// [...]

Eigen::VectorXd pdf(1000);
X1.density().eval(0, 1000, pdf);
```

The result of the evaluation is stored into the vector `pdf`. There are only very few basic or atomic random-variable types defined in StochBB:

Constructor	Parameters	Process description
`stochbb::delta`	`delay`	A constant delay or a process with a fixed waiting time.
`stochbb::unif`	`a`, `b`	A process with a uniform-distributed waiting time.
`stochbb::norm`	`mu`, `sigma`	A process with a normal-distributed waiting time.
`stochbb::gamma`	`k`, `theta`	A process with a gamma-distributed waiting time.
`stochbb::invgamma`	`alpha`, `beta`	A process with an inverse gamma-distributed waiting time.
`stochbb::weibull`	`k`, `lambda`	A process with a Weibull-distributed waiting time.

More complex processes can be derived by combining these atomic random variables or as special cases of them. For example, the exponential distribution $\mathrm{Exp}(\lambda)$ is equivalent to a Gamma distribution with $k = 1$ and $\theta = \lambda^{-1}$.

3.1.1 Affine transformation of random variables

An affine transformation of a random variable X is of the form $a\,X + b$, where $a \neq 0$ and b are real values. An affine transformation can be obtained using the overloaded * and + operators or using the `stochbb::affine` function. For example, the code

```
#include <stochbb/api.h>
using namespace stochbb;

// [...]

Var X = gamma(10, 100);
Var Y = 3*X + 1;
```

constructs a random variable `Y` being an affine transformed of the random variable `X`.

3.1.2 Sums of random variables

Beside the affine transfrom of random variables, the most basic derived random variable is a `Chain`. This type represents the *chaining* of random processes. Such a chain can be constructed using the overloaded `+` operator or the `stochbb::chain` function. For example

```
#include <stochbb/api.hh>
using namespace stochbb;

// [...]

Var X1 = gamma(10,100);
Var X2 = gamma(20, 50);
Var Y = X1 + X2;
```

3.1.3 Minimum and Maximum of random variables

Another simple derived random variable is the `Maximum` or `Minimum` class. As the names suggest, they represent the maximum or minimum of a set of random variables. They can be created using the overloaded standard library function `std::min` and `std::max` or the `stochbb::minimum` and `stochbb::maximum` functions. The latter take a vector of random variables.

```
#include <stochbb/api.hh>
using namespace stochbb;

// [...]

Var X1 = gamma(10,100);
Var X2 = gamma(20, 50);
Var Y = std::max(X1, X2);
```

3.1.4 Mixtures of random variables

Similar to the `Minimum` or `Maximum`, a mixture of random variables can be constructed using the `stochbb::mixture` function. This function takes a at least two variables and their associated

weight. Such a mixture can be considered as a random process which randomly selects the outcome of a set of other random processes, where the probability of selecting a specific process is given by the relative weight assigned to each process.

```
#include <stochbb/api.hh>
using namespace stochbb;

// [...]

Var X1 = gamma(10,100);
Var X2 = gamma(20, 50);
Var Y = mixture(1,X1, 2,X2);
```

In the example above, the random process Y will select the outcome of X_1 with a probability of $\frac{1}{3}$ and the outcome of X_2 with probability $\frac{2}{3}$.

3.1.5 Compound random variables

An important class of derived random processes are compound processes. There the parameters of the distribution of a random variable are themselves random variables. That is

$$X \sim f(x|A)\,, \quad A \sim g(a|\theta)\,,$$

where the random variable X is distributed as $f(x|A)$, parametrized by A, where A itself is a random variable distributed as $g(a|\theta)$, parametrized by θ.

Compound random variables are created using the same factory function like the atomic random variable types. In contrast to the atomic random variables, the factory functions take random variables as parameters instead of constant values.

```
#include <stochbb/api.hh>
using namespace stochbb;

// [...]

Var mu = gamma(10,100);
Var cnorm = norm(mu, 10);
```

instantiates a compound-normal distributed random variable, where the mean is gamma distributed while the standard deviation is fixed.

3.2 Sampling several dependent random variables

As mentioned above, sampling efficiently from a complex system of random variables is not trivial. First of all, it must be ensured that all atomic random variables are sampled only once. Otherwise, two dependent random variables may be sampled as independent. Moreover, the samples of derived random variables should be cached for efficiency.

StochBB provides a separate class that implements a proper sampler for a system of random variables, the `ExactSampler` class. This class allows to sample from several possibly dependent random variables simultaneously. Upon construction, the set of random variables to sample from, is specified. A sample from these random variables can then be obtained by the `ExactSampler::sample` method.

```
#include <stochbb/api.hh>
using namespace stochbb;

// [...]

Var X1 = gamma(10,100);
Var X2 = gamma(20, 50);
Var Y = std::min(X1, X2);
// Construct sampler
ExactSampler sampler(X1, X2, Y);
// Get 1000 samples
Eigen::MatrixXd samples(3, 1000);
sampler.sample(samples);
```

The `ExactSampler::sample` method takes a reference to a `Eigen::Matrix` where each column represents the random variable given to the constructor and each row an independent sample from the system.

For very large systems, sampling may get slow. Particularly if one is only interested in the marginal distribution of single random variables. For these cases, an approximate sampler for single random variables is provided, the `MarginalSampler`. This sampler uses an approximation of the inverse of the cumulative distribution function of a random variable to draw samples.

```
#include <stochbb/api.hh>
using namespace stochbb;

// [...]

Var X1 = gamma(10,100);
Var X2 = gamma(20, 50);
Var Y = std::min(X1, X2);
// Sample from Y on [0,500] in 1000 steps
MarginalSampler sampler(Y, 0, 500, 1000);
// Get 1000 samples
Eigen::VectorXd samples(1000);
sampler.sample(samples);
```

In this example, a `MarginalSampler` is constructed for the random variable `Y`. Using an approximation of its CDF on the interval $[0, 500)$ using 1000 steps. Then, the `MarginalSampler::sample` method is used to obtain 1000 independent samples.

4 Artificial example

Within this section, we show how a rather trivial stochastic model of a reading experiment can be implemented using StochBB's Python interface.

The model describes the fixation duration R as a simple chain of a lexical L, semantic S and occulomotor M stage. Each stage is represented by a Gamma distributed random variable, that is

$$\begin{aligned} L &\sim \Gamma(5\,f + 5, 10) \\ S &\sim \Gamma(10\,p + 5, 20) \\ M &\sim \Gamma(10, 30) \\ \Rightarrow R &= L + S + M\,, \end{aligned}$$

where f is the words log-frequency and p its predictability. In a reading experiment, participants usually do not read a single word with a fixed frequency and predictability but rather read several words with different frequencies and predictabilities. Hence it is necessary to model the predictabilities and frequencies as random variables too, for example

$$f \sim U[0,4]$$
$$p \sim U[0,1]\,.$$

With this, the lexical (L) and semantic (S) stages are not Gamma distributed anymore but compound distributed random variables.

The following code shows how this simple model of a reading experiment is implemented using the Python API of StochBB.

```
from numpy import *
import stochbb

# Corpus model:
#  Log word frequency (per mililon), is uniform distr. on the interval [0,4]
f = stochbb.uniform(0,4)
#  Predictability of word is unif. distributed on the interval [0,1]
p = stochbb.uniform(0,1)

# Experimental model:
#  "lexical" stage is gamma distributed with k=5*f+5, theta=10
L = stochbb.gamma(5*f + 5, 10)
#  "semantic" stage is gamma distributed with k=10*p+5, theta=20
S = stochbb.gamma(10*p + 5, 20)
#  "motor" stage is gamma distributed with k=10, theta=30
M = stochbb.gamma(10, 30)
#  "response latency" is simply R = L + S + M
R = L + S + M

# Evaluate marginal densities on interval [0,1200] ms
Tmin, Tmax, N = 0, 1200, 1000
t = linspace(Tmin, Tmax, N); dt = float(Tmax-Tmin)/N
pL = empty(N,); L.density().eval(Tmin, Tmax, pL)
pS = empty(N,); S.density().eval(Tmin, Tmax, pS)
pM = empty(N,); M.density().eval(Tmin, Tmax, pM)
pR = empty(N,); R.density().eval(Tmin, Tmax, pR)

# Sample 10k samples from system
sam = stochbb.ExactSampler([L,S,M,R])
X = empty((100000, 4))
sam.sample(X)
```

Figure 1 shows the results of the script. The histogram of the samples are in a good agreement with the PDFs obtained numerically.

5 Divergence point example

This example shows a simple *toy model* for some cognitive process that is able to produce a so called *divergence point* [4]. A divergence point of two response-latency distributions is the earliest time point at which the two response-latency distributions differ. Obviously, there exists no divergence point, if the two response latency distributions are analytic (e.g., the convolution of

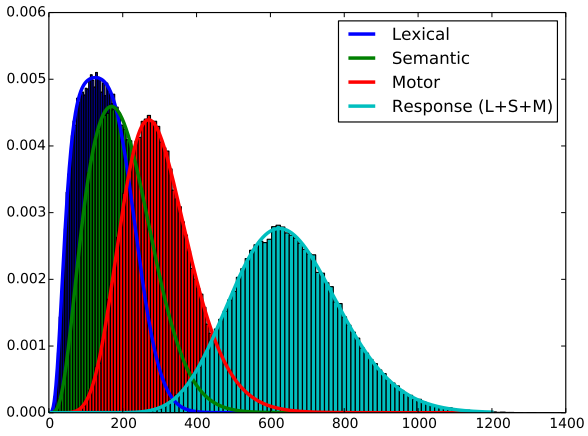

Fig. 1: Shows the histogram of the samples (boxes) and the numerically obtained densities (lines).

Gamma distributions, [1]). This example, however, not only shows how a non-analytic response-latency distribution may arise that allows for a divergence point, but actually has a divergence point[1].

This very simple model consists of 3 Gamma-distributed random variables C, X_1 and X_2, where the latter is a shifted Gamma distribution (here shifted by 300 ms, i.e. $X_3 \sim \Gamma(T - 300; k, \theta)$). The model is

$$\text{control: } R_c = C + X_1 \qquad \text{where } C \sim \Gamma(c; 3, 20),\ X_1 \sim \Gamma(x_1; 10, 30) \tag{1}$$

$$\text{experimental: } R_e = C + \min(X_1, X_2) \qquad \text{where } X_2 \sim \Gamma(x_2 - 300; 1, 70)\,. \tag{2}$$

This model can be read as: Under control condition, the stimulus triggers a common stage C which then triggers a second stage X_2 that itself immediately triggers the response. Under experimental condition, again the stimulus triggers the common stage C which then triggers X_1. Additionally, the common stage C also triggers the delayed stage X_2 in parallel to X_1. The response is then triggered by either X_1 or X_2, depending on which stage completes first. Consequently, the response latency under control condition is $R_e = C + \min(X_1, X_2)$.

The following code shows how this simple model is implemented using the Python API of StochBB.

```
from numpy import *
import stochbb;

# Processing model
d = 300
C = stochbb.gamma(3,20)
X1 = stochbb.gamma(10,30)
# shifted exponential
X2 = stochbb.gamma(1,70) + d
```

[1] At least one distribution being non-analytic is a necessary but not a sufficient condition for the existence of a divergence point.

```
# response latency control condition
#    is simply R = C + X1
Rc = C + X1
# and for the experimental condition
Re = C + stochbb.minimum(X1, X2)

# Eval PDF & CDF
Tmin, Tmax, N = 0, 1200, 1200;
Tc = empty((N,)); Rc.density().eval(Tmin, Tmax, Tc)
Te = empty((N,)); Re.density().eval(Tmin, Tmax, Te)
TCc = empty((N,)); Rc.density().evalCDF(Tmin, Tmax, TCc)
TCe = empty((N,)); Re.density().evalCDF(Tmin, Tmax, TCe)
```

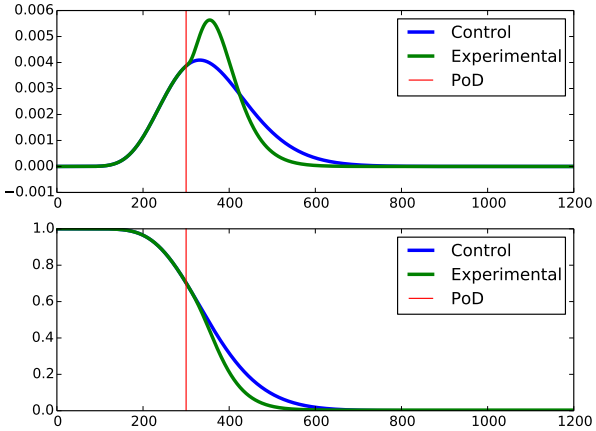

Fig. 2: PDFs (upper panel) and survival functions (lower panel) of the response latencies under control condition (blue lines) and experimental condition (green lines). The divergence point of the two response-latency distributions is shown as the red vertical lines.

Figure 2 shows the plots generated from the code above. The blue lines show the PDF (upper panel) and survival function (1-CDF, lower panel) of the response latency under control condition. Being a convolution of two Gamma distributions, it is an analytic function on the interval $(0, \infty)$ [1]. The green lines show the PDF and survival function of the response latency under control condition. As the distribution of X_2 is not analytic on the complete interval $(0, \infty)$ (discontinuity at $T = 300$ ms), a divergence point may exist at $T = 300$ (red vertical lines). In fact, that point is visible in both the PDF and the survival function. Hence, this simple model is able to produce a divergence point as both response-latency distributions are identical on the interval $[0, 300)$ and diverge thereafter.

References

[1] A. M. Mathai. Storage capacity of a dam with gamma type inputs. *Annals of the Institute of Statistical Mathematics*, 34(1):591–597, 1982.

[2] Hannes Matuschek. StochBB – Stochastic Building Blocks: Application Programming Interface, 2015. http://hmatuschek.github.io/stochbb.

[3] WH Press, BP Flannery, and SA Teukolsky. *Numerical recipes*, volume 547. Cambridge Univ Press, 2007.

[4] EM Reingold, ED Reichle, MG Glaholt, and Heather Sheridan. Direct lexical control of eye movements in reading: Evidence from a survival analysis of fixation durations. *Cognitive psychology*, 64(1):177–206, 2012.