Sampling Distributions in Julia

Andrew Dolgert

July 8, 2023

1 Introduction

In order to write a continuous-time simulation, we need to sample distributions. There are many ways to sample these distributions, and they involve different mathematical manipulations of those distributions. This document looks at the functions defined by Julia for manipulating distributions, in order to figure out how to write samplers.

2 Notation for Distributions

We need to use some accepted notation for a statistical distribution. Let's choose some variable names to use.

A *cumulative distribution* is the probability of an event before a given time. The random variate is T and t is a parameter. P denotes a probability, and F is the name we choose for a general cumulative distribution.

$$P(T \le t) = F(t) \tag{1}$$

The survival is the probability an event fires after time t.

$$P(T > t) = G(t) = 1 - F(t) \tag{2}$$

The *probability density function,* or pdf, is the derivative of the cumulative distribution.

$$f(t) = dF(t)/dt \tag{3}$$

The *hazard rate* is the probability per unit time of an event, given that it has not yet fired.

$$\lim_{\delta t \to 0} P(t < T \le t + \delta t) = \lambda(t) \tag{4}$$

Every continuous distribution can be written in terms of the hazard rate.

$$F(t) = 1 - e^{-\int_0^t \lambda(s) ds} \tag{5}$$

This means the pdf is also a function of hazard rate.

$$f(t) = \lambda(t) e^{-\int_0^t \lambda(s) ds} \tag{6}$$

The survival, in terms of the hazard rate, is one term.

$$G(t) = e^{-\int_0^t \lambda(s)ds} \tag{7}$$

The log of the survival is called the integrated hazard.

$$\ln G(t) = -\int_0^t \lambda(s)ds = -\Lambda(t) \tag{8}$$

3 Julia Distributions

Julia has a `Distributions.jl` library that contains univariate distributions, which are what we want. Let's walk through the functions this package offers for working with those distributions. We can write each function's mathematical equivalent.

3.1 Parameters

Julia has one function to retrieve all parameters of a distribution and a few functions that are specific to common parameters.

- `params(d::UnivariateDistribution)`—Gets all parameters for a distribution.
- `scale(d::UnivariateDistribution)`—If the distribution has a parameter θ and is written as $f(t/\theta)$, then θ is the scale.
- `rate(d::UnivariateDistribution)`—If the distribution has a parameter β and is written as $f(\beta t)$, then β is the rate. We see that $\beta\theta = 1$.
- `location(d::UnivariateDistribution)`—If the distribution has a parameter μ and is written as $f(t-\mu)$, then μ is the location. This will be important for re-sampling distributions which failed to fire.
- `shape(d::UnivariateDistribution)`—The shape is often a power, k, of t^k in the pdf.

3.2 Probability Evaluation

`insupport(d::UnivariateDistribution, x::Any)`—Whether t is in the domain of $f(t)$.

`pdf(d::UnivariateDistribution, x::Real)`—This is $f(t)$.

`logpdf(d::UnivariateDistribution, x::Real)`—It's $\ln(f(t))$. This is important because the pdf is a likelihood, so you get the log-likelihood.

`cdf(d::UnivariateDistribution, x::Real)`—$F(t)$.

`logcdf(d::UnivariateDistribution, x::Real)`—$\ln(F(t))$.

`logdiffcdf(d::UnivariateDistribution, x::Real, y::Real)`—$\ln(F(x)) - \ln(F(y))$.

Julia Function	Equation	Statistics name
`ccdf`	$S(x) = e^{-\int^x \lambda(s)ds}$	Survival
`cdf`	$F(x) = 1 - S(x) = 1 - e^{-\int^x \lambda(s)ds}$	Cumulative distribution function
`pdf`	$f(x) = \lambda(x)e^{-\int^x \lambda(s)ds}$	Probability distribution function
`logpdf`	$\ln\ f(x)$	Log-likelihood
`logcdf`	$\ln\ F(x)$	
`logdiffcdf`	$\ln\left(F(x_2) - F(x_1)\right)$	
`logccdf`	$-\int^x \lambda(s)ds$	Integrated hazard (negated)
`quantile`	$y = F^{-1}(x)$	Inverse cumulative distribution function
`cquantile`	$y = F^{-1}(1-x)$ so $x = S(y)$	Inverse Survival
`invlogcdf`	$x = \ln\ F(y)$ so $e^x = F(y)$	
`invlogccdf`	$x = -\int_0^y \lambda(s)ds$	Inverse integrated hazard

Table 1: This translates between Julia functions and hazard-based notation. We can use this to find the shortest path to our calculation in code.

`ccdf(d::UnivariateDistribution, x::Real)`—Survival, $G(t) = 1 - F(t)$.

`logccdf(d::UnivariateDistribution, x::Real)`—The log of the survival, which is called the integrated hazard, $\Lambda(t)$.

`invlogcdf(d::UnivariateDistribution, x::Real)`—The inverse function of log-CDF. That's

$$x = \ln F(y) \tag{9}$$

so

$$e^x = F(y) = 1 - e^{\int_0^y \lambda(s)ds}. \tag{10}$$

`invlogccdf(d::UnivariateDistribution, x::Real)`—This inverse is one I've used.

$$e^x = G(y) = e^{\int_0^y \lambda(s)ds} \tag{11}$$

That means this function is the inverse of the integrated hazard.

$$x = \Lambda(y) \tag{12}$$

4 Next Reaction for Non-Markov Processes

Gibson and Bruck describe the Next Reaction Method for non-Markov processes by focusing on one step of the algorithm. Assuming there was a draw for some time τ when the distribution F_{a_n} would fire, and there is now a change to create distribution $F_{a,n+1}$, the new draw should be

$$\tau' = F_{a,n+1}^{-1}\left(\frac{F_{a,n}(\tau) - F_{a,n}(t_n)}{1 - F_{a,n}(t_n)}\right). \tag{13}$$

Shifting to get rid of the inverse, this becomes

$$F_{a,n+1}(\tau') = \frac{F_{a,n}(\tau) - F_{a,n}(t_n)}{1 - F_{a,n}(t_n)}. \tag{14}$$

I suspect this equation will make a lot of sense if we write it in terms of survivals. Start by writing it as $S = 1 - F$

$$1 - F_{a,n+1}(\tau') = \frac{1 - F_{a,n}(\tau)}{1 - F_{a,n}(t_n)}. \tag{15}$$

Multiplying both sides by the denonminator shows a known rule about survival functions. The rule is that *conditional survival is multiplicative.*

$$\left(1 - F_{a,n}(t_n)\right)\left(1 - F_{a,n+1}(\tau')\right) = 1 - F_{a,n}(\tau). \tag{16}$$

The equation says that we expect the proposed survival of this process to time τ to be the same as the survival to from start to t_n and the survival from t_n to τ'.

Have you been anxious about the notation? The Gibson and Bruck paper failed with the notation, and we just copied it. The problem is how they annotated the zero-time for distributions. We can clarify it by writing the equation in terms of hazards.

$$\exp\left(-\int_{t_0}^{\tau} \lambda_n(s)ds\right) = \exp\left(-\int_{t_0}^{t_n} \lambda_n(s)ds\right) \exp\left(-\int_{t_n}^{\tau'} \lambda_{n+1}(s)ds\right) \tag{17}$$

The integrals cover the relevant durations and the hazards are the hazards of those distributions during those durations. This equation also leads to an observation. Anderson's method is the Next Reaction method in log space. Who knew? Everybody. Everybody knew.

We can make our notation in a way that will have an echo in the code. All software libraries treat distributions as starting at time zero and leave it to the client to shift their absolute time. Therefore, let's think about distributions as an enabling time that defines the zero, a current time, and the distribution, (t_e, t_n, f).

If we think of the first draw as a draw by inversion, we can write it as inversion of a conditional survival.

$$U = S(t_n - t_e, \tau - t_n) \tag{18}$$

Because $0 < U < 1$, this could equally well be $U = F(t_n - t_e, \tau - t_n)$ equally well. If the enabling time is in the future, that's fine, but it changes the draw to use a marginal survival.

$$U = S(0, \tau - t_e) = S(\tau - t_e) \tag{19}$$

This U is the right-hand side of 16.

When Gibson and Bruck discuss an update rule, it requires storing the remaining U.

1. When we make the first draw, we save U.

2. Each time, t_n, a transition is disabled, we calculate

$$U' = \frac{U}{S(t_n - t_e, \tau - t_n)} \tag{20}$$

3. When the transition is enabled or changed, we calculate its draw by inversion with the saved U',

$$U' = S(t_n - t_e, \tau' - t_n). \tag{21}$$

While the first draw of τ can be done using appropriate, non-inversion, methods for any distribution, it is a weakness of this method that later enabling of the transition requires using inversion, which can be much slower and more error-prone.

Let's walk through the cases because they clarify what we need to store within the sampler.

- First time enabling the transition. Sample any way possible from $S(t_n - t_e, \tau - t_n)$, but then calculate and store $U = S(t_n - t_e, \tau - t_n)$.
- Enabling a transition that fired. Sample any way possible from $S(t_n - t_e, \tau - t_n)$, but then calculate and store $U = S(t_n - t_e, \tau - t_n)$.
- Enabling a transition that was disabled before it fired.
  - General case. Sample by inversion $U = S(t_n - t_e, \tau - t_n)$.
  - The transition is picking up where it left off. We don't have to sample because the firing time is just pushed forward. Does this happen often enough to care? If the previous distribution was (t_e, t_n, f) when it was disabled at time t_n, we know the transition is merely shifted because, at time t_{n+1}, it has the form $(t_{n+1} - (t_e - t_n), t_{n+1}, f)$. In this case, $\tau' = \tau + t_{n+1} - t_n$, and there is no change to the stored U.
- Disabling a transition that did not fire. If a transition did not fire at time t_{n+1}, calculate conditional survival, $S(t_n - t_e, t_{n+1} - t_e)$ and adjust the stored U by dividing by the conditional survival.
- Disabling the transition that fired. If we adjust the conditional survival according to the formula above, it may close to zero but not equal. We need it to be $U = 0$, so set that value.

From the cases above, the sampler needs to store two kinds of information.

- Long-term values about all transitions. These are U, $t_e - t_n$, and f.
- Short-term values for currently-enabled transitions. These are t_e, t_n, f.

There are a couple of challenges. One is how to know when two floating-point values are equal, such as in the check that the transition is picking up where it left off. This is handled in code by calculating the machine epsilon for a floating point of the size of t_n. Multiply that by a small factor, such as 2 or 4, and use that as an error bound.

Another challenge is how to figure out which disabled transition is the one that fired. If the caller specifies it is disabling the transition that fired, then it's clear. If the caller doesn't specify, can we know? The transitions are in a heap where the soonest is on top. If the first transition that's disabled was the one on top and the next transition becomes the one on top, it could look like it was next to fire. We could check the times on those transitions in order to see whether the time of disabling matches the time of the top transition, but that's prone to error because it requires equality of floating-point numbers. There will definitely be some time when two numbers in a continuous time simulation are very close to each other.

On the other hand, there isn't a good way to use a Next Reaction sampler except to always choose the transition it sampled to be next. Let's store the response from the next() function and assume that's the one that fired when it gets disabled.

5 Timing Sampling Methods

The Next Reaction method (Gibson and Bruck) and the Modified Next Reaction method (Anderson) are essentially the same algorithm except that one samples in a linear space and one samples in a logarithmic space. Which one is better depends on the distribution we're sampling and the parameter ranges for that distribution. This section looks at tests of the distributions defined in Julia in order to classify which one goes in which category, linear or logarithmic sampling.

Below is a table that comes from a test in `test/nrmetric.jl`. The distributions come from `Distributions.jl`. Each distribution is tested on 10,000 values. The table will have the following values.

1. The space for the test, which is linear or logarithmic.

2. Error, as the log base-10 of the maximum error after sampling a survival and then inverting that back to get the sample. When the number is -16, that means the error is 10^{-16}.

3. Average error in log base-10, which is the trimmed mean over all runs of the error in the previous column.

4. Forward timing in seconds. For the linear space, this is the time to perform the `ccdf` call. For the logarithmic space, this is the time to perform the `logccdf` call, both 10,000 times. A time of 6e-4 seconds means each iteration took, on average, 6e-8 seconds.

5. Backward timing in seconds. For the linear space, this is the time to perform the `cquantile` call. For the logarithmic space, this is the time to perform the `invlogccdf` call, both 10,000 times.

Space	$\log_{10}\|\epsilon\|_1$	$\log_{10}\langle\epsilon\rangle$	forward [s]	backward [s]	both [s]
Arcsine{Float64}(a=0.0, b=1.0)					
Linear	-12.7	-16.4	1.01e-06	9.00e-07	1.91e-06
Log	-15.7	-16.3	2.26e-06	1.09e-06	3.35e-06
BetaPrime{Float64}(α=1.0, β=1.0)					
Linear	-15.8	-16.8	5.24e-06	2.66e-05	3.18e-05
Log	-14.1	-14.9	7.08e-06	3.34e-05	4.05e-05
Biweight{Float64}(μ=0.0, σ=1.0)					
Linear	-15.4	-16.2	7.00e-08	7.98e-06	8.05e-06
Log	-13.1	-15.9	1.14e-06	4.22e-05	4.33e-05
Beta{Float64}(α=1.0, β=1.0)					
Linear	-16.0	-16.9	5.37e-06	2.76e-05	3.30e-05
Log	-16.0	-17.1	6.95e-06	3.52e-05	4.21e-05
Cauchy{Float64}(μ=0.0, σ=1.0)					
Linear	-16.0	-17.4	9.80e-07	1.71e-06	2.69e-06
Log	-13.5	-14.5	3.79e-06	1.92e-06	5.71e-06
Cosine{Float64}(μ=0.0, σ=1.0)					
Linear	-10.9	-11.3	1.09e-06	9.10e-05	9.21e-05
Log	-5.4	-11.1	2.06e-06	8.41e-05	8.62e-05
Epanechnikov{Float64}(μ=0.0, σ=1.0)					
Linear	-16.0	-17.0	6.00e-08	7.56e-06	7.62e-06
Log	-14.7	-16.1	1.10e-06	3.73e-05	3.84e-05
Erlang{Float64}(α=1, θ=1.0)					
Linear	-15.8	-16.7	1.09e-05	2.83e-05	3.92e-05
Log	-14.4	-16.0	5.03e-06	3.51e-05	4.02e-05
Exponential{Float64}(θ=1.0)					
Linear	-16.3	-18.1	5.20e-07	1.75e-06	2.27e-06
Log	-Inf	-Inf	2.70e-07	5.00e-08	3.20e-07
Frechet{Float64}(α=1.0, θ=1.0)					
Linear	-16.0	-18.7	1.00e-06	1.42e-06	2.42e-06
Log	-14.4	-15.4	3.36e-06	2.14e-06	5.50e-06
Gamma{Float64}(α=1.0, θ=1.0)					
Linear	-15.8	-16.7	1.14e-05	2.94e-05	4.09e-05
Log	-14.4	-16.0	5.55e-06	3.70e-05	4.25e-05
GeneralizedPareto{Float64}(μ=0.0, σ=1.0, ξ=1.0)					
Linear	-16.1	-16.8	1.73e-06	2.23e-06	3.96e-06
Log	-14.1	-14.8	1.34e-06	2.66e-06	4.00e-06

Space	$\log_{10}\|\epsilon\|_1$	$\log_{10}\langle\epsilon\rangle$	forward [s]	backward [s]	both [s]
Gumbel{Float64}(μ=0.0, θ=1.0)					
Linear	-16.0	-16.9	1.00e-06	3.26e-06	4.26e-06
Log	Inf	Inf	3.95e-06	1.84e-06	5.79e-06
InverseGamma{Float64}(invd: Gamma{Float64}(α=1.0, θ=1.0) θ: 1.0)					
Linear	-15.5	-16.7	1.44e-05	3.76e-05	5.20e-05
Log	-14.3	-15.2	2.09e-05	4.31e-05	6.40e-05
InverseGaussian{Float64}(μ=1.0, λ=1.0)					
Linear	-15.5	-16.2	3.66e-06	3.61e-05	3.98e-05
Log	-13.6	-14.6	7.06e-06	1.37e-04	1.44e-04
JohnsonSU{Int64}(ξ=0, λ=1, γ=0, δ=1)					
Linear	-15.5	-16.7	3.17e-06	4.16e-06	7.33e-06
Log	-13.9	-14.9	7.34e-06	3.30e-06	1.06e-05
Kolmogorov()					
Linear	-15.6	-16.4	1.02e-06	1.02e-04	1.03e-04
Log	-1.0	-Inf	2.34e-06	1.13e-03	1.13e-03
Kumaraswamy{Float64}(a=1.0, b=1.0)					
Linear	-16.3	-16.9	8.60e-07	9.00e-07	1.76e-06
Log	-16.0	-16.8	1.35e-06	1.32e-06	2.67e-06
Laplace{Float64}(μ=0.0, θ=1.0)					
Linear	-16.9	-Inf	5.90e-07	1.89e-06	2.48e-06
Log	-11.7	-14.1	1.08e-06	8.10e-07	1.89e-06
Levy{Float64}(μ=0.0, σ=1.0)					
Linear	-15.2	-16.5	9.40e-07	1.00e-06	1.94e-06
Log	-14.1	-14.9	3.93e-06	1.68e-06	5.61e-06
Lindley{Float64}(θ=1.0)					
Linear	-15.7	-16.3	4.90e-07	1.53e-05	1.58e-05
Log	-12.7	-14.4	1.24e-06	1.57e-05	1.70e-05
Logistic{Float64}(μ=0.0, θ=1.0)					
Linear	-16.0	-17.2	6.60e-07	2.12e-06	2.78e-06
Log	-14.8	-18.2	2.69e-06	1.27e-06	3.96e-06
LogitNormal{Float64}(μ=0.0, σ=1.0)					
Linear	-15.5	-16.3	2.29e-06	2.20e-06	4.49e-06
Log	-16.0	-16.6	5.37e-06	2.80e-06	8.17e-06
LogNormal{Float64}(μ=0.0, σ=1.0)					
Linear	-15.5	-16.7	2.39e-06	1.96e-06	4.35e-06
Log	-14.1	-15.1	4.31e-06	2.00e-06	6.31e-06

Space	$\log_{10}\|\epsilon\|_1$	$\log_{10}\langle\epsilon\rangle$	forward [s]	backward [s]	both [s]
Normal{Float64}(μ=0.0, σ=1.0)					
Linear	-15.5	-16.7	9.80e-07	1.43e-06	2.41e-06
Log	-14.4	-15.3	4.79e-06	2.16e-06	6.95e-06
NormalCanon{Float64}(η=0.0, λ=1.0, μ=0.0)					
Linear	-15.5	-16.7	8.40e-07	1.47e-06	2.31e-06
Log	-14.4	-15.3	3.02e-06	2.26e-06	5.28e-06
Pareto{Float64}(α=1.0, θ=1.0)					
Linear	-16.0	-19.1	4.80e-07	5.10e-07	9.90e-07
Log	-14.1	-15.1	2.05e-06	9.00e-07	2.95e-06
PGeneralizedGaussian{Float64}(μ=0.0, α=1.4142135623730951, p=2.0)					
Linear	-16.0	-16.9	2.44e-06	2.30e-05	2.54e-05
Log	Inf	Inf	5.17e-06	3.10e-05	3.62e-05
Rayleigh{Float64}(σ=1.0)					
Linear	-16.0	-16.8	5.00e-07	1.76e-06	2.26e-06
Log	Inf	Inf	4.30e-07	1.60e-06	2.03e-06
Rician{Float64}(ν=0.0, σ=1.0)					
Linear	-15.6	-16.5	1.19e-05	7.51e-05	8.70e-05
Log	Inf	Inf	1.09e-05	3.98e-04	4.09e-04
SkewedExponentialPower{Rational{Int64}}(μ=0//1, σ=1//1, p=2//1, α=1//2)					
Linear	-16.0	-16.9	5.02e-06	2.51e-05	3.02e-05
Log	Inf	Inf	7.83e-06	3.35e-05	4.13e-05
SymTriangularDist{Float64}(μ=0.0, σ=1.0)					
Linear	-16.0	-17.0	1.60e-07	6.00e-07	7.60e-07
Log	-16.0	-16.8	3.64e-06	8.20e-07	4.46e-06
Triweight{Float64}(μ=0.0, σ=1.0)					
Linear	-15.2	-16.0	5.00e-07	9.57e-06	1.01e-05
Log	-11.3	-15.7	1.46e-06	5.65e-05	5.79e-05
Uniform{Float64}(a=0.0, b=1.0)					
Linear	-16.3	-16.9	5.00e-08	2.20e-07	2.70e-07
Log	-16.0	-16.8	9.10e-07	5.60e-07	1.47e-06
Weibull{Float64}(α=1.0, θ=1.0)					
Linear	-16.3	-18.1	8.40e-07	1.21e-06	2.05e-06
Log	-Inf	-Inf	6.40e-07	5.40e-07	1.18e-06

The tables highlight which sampling method is more accurate and which is faster. Sometimes these two are at odds. Here's how I made choices.

- If any value in the range was inaccurate enough to give a log value of 1, that's definitely not acceptable. We see this for the log-space sampling of Beta, for instance.
- If the accuracy is great for either choice, but one is much faster, let's go with the faster one. The Laplace distribution is a good example. The worst accuracy is 10^{-15}, but the log-space sampling is ten times faster.

The Laplace distribution is the only one I can see going either way. You can

either get ten times the accuracy or ten times the speed.