Sampling Distributions in Julia

Andrew Dolgert

July 10, 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, or cdf.

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

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.

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

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.

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

Because the survival has this form, where it's an integral in the exponential, we can think of intermediate waypoints in time.

$$S(t) = e^{-\int_0^{t_1} \lambda(s)ds} e^{-\int_{t_1}^{t} \lambda(s)ds} \tag{8}$$

Each of the terms is a conditional survival, written $S(t_1, t_2)$. In other words,

$$S(t) = S(0,t) = S(0,t_1)S(t_1,t), \tag{9}$$

so people say the conditional survival is multiplicative. Going back to the powerful probability language the conditional survival can be written in terms of the marginal survival.

$$P(T > t | T > t_1) = \frac{P(T > t)}{P(T > t_1)}. \tag{10}$$

Multiplying both sides shows that conditional survival is multiplicative.

$$P(T > t) = P(T > t_1)P(T > t | T > t_1) \tag{11}$$

Another way to look at a conditional survival is in the cdf space, where it becomes a fraction.

$$S(t_1, t) = \frac{S(t)}{S(t_1)} = \frac{1 - F(t)}{1 - F(t_1)} \tag{12}$$

Writing the survival in terms of the hazard also makes it easier to see how it relates to the pdf.

$$f(t) = -\frac{d}{dt}S(t) \tag{13}$$

The log of the survival is called the integrated hazard.

$$\ln S(t) = -\int_0^t \lambda(s)ds = -\Lambda(t) \tag{14}$$

You can get the hazard directly from survival and the pdf.

$$\lambda(t) = \frac{f(t)}{S(t)} \tag{15}$$

Rearranging slightly shows us a form we'll see later, $f(t) = \lambda(t)S(t)$.

What if we had a distribution described by $\lambda(t)$ and wanted to define a new distribution that started later. For instance, a Gamma distribution fits our data, but only if it starts after $t_0 = 0.1$? For such a distribution, the new survival would be

$$S'(t) = e^{-\int_{t_0}^{t_0+t} \lambda(s)ds} = \frac{S(t_0, t_0 + t)}{S(0, t_0)}. \tag{16}$$

An even easier way to think about how the distribution changes is to consider changes not to the exponential but to the integral inside the exponential.

$$\Lambda'(t) = \int_{t_0}^{t_0+t} \lambda(s)ds = \int_0^{t_0+t} \lambda(s)ds - \int_0^{t_0} \lambda(s)ds = \Lambda(t_0+t) - \Lambda(t_0) \quad (17)$$

If, for some reason, we need to think about shifting distributions, and if we can work in this space of survivals and integrated hazards, then there are concise ways to recalculate a shifted distribution from its original form.

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

Later we will use two pairs of functions from `Distributions.jl` to implement Next Reaction method. The first pair is the survival and its inverse: `ccdf()` and `cquantile()`. You can see in Tab. 1 that these are inverses of each other. The second pair is the same, but in the log-space.

`invlogccdf(d::UnivariateDistribution, x::Real)` can be translated into

$$e^x = G(y) = e^{\int_0^y \lambda(s)ds}. \quad (18)$$

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.

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

$$x = \Lambda(y) \tag{19}$$

In this package, the integrated hazard is called `logccdf`.

4 Next Reaction for Non-Markov Processes

Many of the calculations below rely on fluency with shifted distributions. Shifted distributions are the basis of the Next Reaction method and Modified Next Reaction method, so we will derive those methods here, but the further goal is to be comfortable with calculation of shifted distributions.

A sampler's main goal is to sample. Using the notation from Sec. 2, we write sampling as two steps. First, draw a random number from $0 \le U < 1$. Then invert the cdf to find the sampled relative time t.

$$U = F(t) \tag{20}$$

However, since the survival is one minus the cdf, we can also write this as

$$U = S(t) \tag{21}$$

with the same result.

Because simulations move through time and transitions are enabled and disabled, we have to agree on notation for time relative to the start of a transition and how it relates to absolute time within the simulation.

- t—Time relative to the zero-point of the distribution, as implemented in `Distributions.jl`. If you look up a gamma distribution in Wikipedia, it is relative to some implied start time at $t = 0$, and we denote its pdf with $f(t)$, as you saw in Sec. 2.

- t_0—Absolute time at which the transition is enabled.
- t_e—Absolute time at which to place the zero-point of the distribution. If $t_e < t_0$, it's equivalent to saying that we are cutting off the left side of the distribution, up to $t = t_0 - t_e$. If $t_e > t_0$, it's equivalent to saying the transition cannot possibly fire until after time absolute time t_e.
- t_n—We'll use this to represent some later absolute time at which the distribution changes its parameters in some way.

Let's start with the simplifying assumption that $t_e = t_0$. Use Eq. 21 to write sampling in absolute time.

$$u = S(\tau - t_0) \tag{22}$$

When we invert it, this is

$$\tau = t_0 + S^{-1}(u). \tag{23}$$

Another way to write this is to expand the exponential.

$$u = \exp\left(-\int_{t_0}^{\tau} \lambda(s - t_0)ds\right) \tag{24}$$

This shows that there is a linear integration inside the exponent. We can use this in the Next Reaction scheme.

Suppose there were a time t_0 at which a transition was enabled. Then, at a later time, t_n, a change in simulation state changed the parameters of the transition. Could we figure out what the drawn time, τ, would have been had we known that the parameters would change?

During the draw in Eq. 24, the distribution we draw from is described by its hazard, λ, so let's label the initial hazard λ_0 and the subsequent hazard λ_n.

$$\begin{aligned} u &= \exp\left(-\int_{t_0}^{\tau} \lambda_0(s - t_0)ds\right) && (25)\\ &= \exp\left(-\int_{t_0}^{t_n} \lambda_0(s - t_0)ds\right)\exp\left(-\int_{t_n}^{\tau'} \lambda_n(s - t_0)ds\right) && (26)\end{aligned}$$

We can think of the new draw, τ' as coming from the same element of the uniform random variate, u.

The algorithm of Gibson and Bruck starts with the draw from Eq. 24.

1. Draw an element u from a uniform random variate U.
2. Solve for a putative firing time τ by inverting the survival equation, Eq. 24.
3. If, at some later time t_n, there is a change to the parameters of the distribution, then calculate the conditional survival up to t_n and update the element u.

$$u' = u \exp\left(\int_{t_0}^{t_n} \lambda_0(s - t_0)ds\right) \tag{27}$$

4. Since you were able to invert the survival before, you can do it again, this time with u' and the updated hazard rate.

$$u' = \exp\left(-\int_{t_n}^{\tau'} \lambda_n(s-t_0)ds\right) \tag{28}$$

Solving this gives the new draw, τ' from the old uniform variate, u.

If we return to Eq. 26, wouldn't it be easier to compute if we could take the logarithm of both sides?

$$\ln u = -\int_{t_0}^{\tau} \lambda_0(s-t_0)ds = -\int_{t_0}^{t_n} \lambda_0(s-t_0)ds - \int_{t_n}^{\tau'} \lambda_n(s-t_0)ds \tag{29}$$

This log-space calculation is what Anderson's modified next reaction recommends. It turns out that calculations in this space are both faster and more precise for most of the distributions commonly used in simulation, such as the Exponential, Gamma, and Erlang distributions.

XXX Working from here down.

Gibson and Bruck describe the Next Reaction Method for non-Markov processes by focusing on one step of the algorithm. Identify the transitions with a and the time points as T_n. 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

$$\tau' = F_{a,n+1}^{-1}\left(\frac{F_{a,n}(\tau) - F_{a,n}(t_n)}{1 - F_{a,n}(t_n)}\right). \tag{30}$$

Shifting to get rid of the inverse, this becomes

$$F_{a,n+1}(\tau') = \frac{F_{a,n}(\tau) - F_{a,n}(t_n)}{1 - F_{a,n}(t_n)}. \tag{31}$$

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

$$1 - F_{a,n+1}(\tau') = \frac{1 - F_{a,n}(\tau)}{1 - F_{a,n}(t_n)}. \tag{32}$$

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

$$(1 - F_{a,n}(t_n))\,(1 - F_{a,n+1}(\tau')) = 1 - F_{a,n}(\tau). \tag{33}$$

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.

$$\exp\left(-\int_{t_0}^{\tau}\lambda_n(s)ds\right) = \exp\left(-\int_{t_0}^{t_n}\lambda_n(s)ds\right)\exp\left(-\int_{t_n}^{\tau'}\lambda_{n+1}(s)ds\right) \quad (34)$$

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.

$$U = S(t_n - t_e, \tau - t_n) \quad (35)$$

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.

$$U = S(0, \tau - t_e) = S(\tau - t_e) \quad (36)$$

This U is the right-hand side of 33.

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
$$U' = \frac{U}{S(t_n - t_e, \tau - t_n)} \quad (37)$$
3. When the transition is enabled or changed, we calculate its draw by inversion with the saved U',
$$U' = S(t_n - t_e, \tau' - t_n). \quad (38)$$

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 appears twice, once with default parameters and once as a truncated distribution, which is Julia's way of limiting the distribution's support so that we can sample values that are later than a given time. 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	$\log_{10}\lvert\epsilon\rvert_1$	$\log_{10}\langle\epsilon\rangle$	forward [s]	backward [s]	both [s]
Arcsine{Float64}(a=0.0, b=1.0)					
Linear	-12.7	-16.5	8.34e-05	7.52e-05	1.59e-04
Log	-15.4	-16.3	1.99e-04	9.08e-05	2.90e-04
Truncated(Arcsine{Float64}(a=0.0, b=1.0); lower=0.01, upper=Inf)					
Linear	-12.9	-16.2	1.05e-04	6.43e-05	1.69e-04
Log	-15.4	-16.3	5.48e-04	1.54e-04	7.01e-04
BetaPrime{Float64}(α=1.0, β=1.0)					
Linear	-15.7	-16.8	5.40e-04	2.52e-03	3.06e-03
Log	-13.9	-15.0	7.25e-04	3.41e-03	4.13e-03
Truncated(BetaPrime{Float64}(α=1.0, β=1.0); lower=0.01, upper=Inf)					
Linear	-15.6	-16.4	5.73e-04	2.52e-03	3.09e-03
Log	-13.7	-14.9	8.31e-04	2.64e-03	3.47e-03
Biweight{Float64}(μ=0.0, σ=1.0)					
Linear	-13.4	-14.2	7.00e-06	7.92e-04	7.99e-04
Log	-11.1	-13.9	1.14e-04	4.20e-03	4.31e-03
Truncated(Biweight{Float64}(μ=0.0, σ=1.0); lower=0.01, upper=Inf)					
Linear	-13.0	-13.6	3.30e-05	7.99e-04	8.32e-04
Log	-11.1	-13.3	2.59e-04	8.85e-04	1.14e-03
Beta{Float64}(α=1.0, β=1.0)					
Linear	-16.0	-16.9	5.25e-04	2.51e-03	3.04e-03
Log	-16.0	-17.1	6.72e-04	3.41e-03	4.08e-03
Truncated(Beta{Float64}(α=1.0, β=1.0); lower=0.01, upper=Inf)					
Linear	-15.7	-16.5	5.50e-04	2.51e-03	3.06e-03
Log	-15.7	-16.6	8.00e-04	2.41e-03	3.21e-03
Cauchy{Float64}(μ=0.0, σ=1.0)					
Linear	-16.0	-17.5	7.09e-05	1.37e-04	2.08e-04
Log	-13.5	-14.5	3.41e-04	1.65e-04	5.06e-04
Truncated(Cauchy{Float64}(μ=0.0, σ=1.0); lower=0.01, upper=Inf)					
Linear	-15.5	-16.2	8.79e-05	1.34e-04	2.22e-04
Log	-13.3	-14.4	4.18e-04	1.74e-04	5.93e-04
Cosine{Float64}(μ=0.0, σ=1.0)					
Linear	-10.8	-11.3	8.26e-05	8.42e-03	8.50e-03
Log	-5.4	-11.1	2.43e-04	8.72e-03	8.96e-03
Truncated(Cosine{Float64}(μ=0.0, σ=1.0); lower=0.01, upper=Inf)					
Linear	-10.5	-11.0	9.33e-05	8.35e-03	8.44e-03
Log	-5.4	-11.1	3.68e-04	8.15e-03	8.52e-03

Space	$\log_{10}\lvert\epsilon\rvert_1$	$\log_{10}\langle\epsilon\rangle$	forward [s]	backward [s]	both [s]
Epanechnikov{Float64}(μ=0.0, σ=1.0)					
Linear	-15.7	-17.0	3.27e-06	6.92e-04	6.95e-04
Log	-12.2	-16.1	9.10e-05	3.54e-03	3.64e-03
Truncated(Epanechnikov{Float64}(μ=0.0, σ=1.0); lower=0.01, upper=Inf)					
Linear	-15.2	-16.2	5.79e-06	5.76e-04	5.82e-04
Log	-12.9	-16.0	2.37e-04	6.77e-04	9.14e-04
Erlang{Float64}(α=1, θ=1.0)					
Linear	-15.6	-16.8	1.27e-03	3.27e-03	4.55e-03
Log	-14.4	-15.8	5.34e-04	3.35e-03	3.89e-03
Truncated(Erlang{Float64}(α=1, θ=1.0); lower=0.01, upper=Inf)					
Linear	-15.5	-16.3	1.38e-03	3.33e-03	4.70e-03
Log	-11.7	-13.6	6.18e-04	2.91e-03	3.53e-03
Exponential{Float64}(θ=1.0)					
Linear	-16.3	-18.1	4.24e-05	1.58e-04	2.00e-04
Log	-Inf	-Inf	2.11e-05	7.10e-07	2.18e-05
Truncated(Exponential{Float64}(θ=1.0); lower=0.01, upper=Inf)					
Linear	-15.7	-16.3	6.57e-05	9.02e-05	1.56e-04
Log	-11.7	-13.6	1.40e-04	1.29e-04	2.69e-04
Frechet{Float64}(α=1.0, θ=1.0)					
Linear	-16.0	-18.6	8.49e-05	1.29e-04	2.14e-04
Log	-3.0	-15.3	3.17e-04	1.99e-04	5.17e-04
Truncated(Frechet{Float64}(α=1.0, θ=1.0); lower=0.01, upper=Inf)					
Linear	-15.8	-16.8	9.61e-05	1.37e-04	2.33e-04
Log	-14.1	-15.0	4.39e-04	2.03e-04	6.42e-04
Gamma{Float64}(α=1.0, θ=1.0)					
Linear	-15.6	-16.8	1.33e-03	3.43e-03	4.75e-03
Log	-14.4	-15.8	6.87e-04	3.94e-03	4.62e-03
Truncated(Gamma{Float64}(α=1.0, θ=1.0); lower=0.01, upper=Inf)					
Linear	-15.5	-16.3	1.35e-03	3.47e-03	4.83e-03
Log	-11.7	-13.6	8.42e-04	3.15e-03	3.99e-03
GeneralizedPareto{Float64}(μ=0.0, σ=1.0, ξ=1.0)					
Linear	-16.0	-16.9	1.63e-04	2.12e-04	3.75e-04
Log	-14.0	-15.0	1.26e-04	2.46e-04	3.72e-04
Truncated(GeneralizedPareto{Float64}(μ=0.0, σ=1.0, ξ=1.0); lower=0.01, up					
Linear	-15.7	-16.5	1.92e-04	2.08e-04	4.01e-04
Log	-13.8	-14.8	3.39e-04	2.66e-04	6.05e-04

Space	$\log_{10}\lvert\epsilon\rvert_1$	$\log_{10}\langle\epsilon\rangle$	forward [s]	backward [s]	both [s]
Gumbel{Float64}(μ=0.0, θ=1.0)					
Linear	-15.8	-16.9	9.47e-05	3.05e-04	4.00e-04
Log	Inf	Inf	3.64e-04	1.48e-04	5.12e-04
Truncated(Gumbel{Float64}(μ=0.0, θ=1.0); lower=0.01, upper=Inf)					
Linear	-15.5	-16.0	1.15e-04	1.62e-04	2.77e-04
Log	-11.5	-13.3	4.97e-04	2.04e-04	7.02e-04
InverseGamma{Float64}(invd: Gamma{Float64}(α=1.0, θ=1.0) θ: 1.0)					
Linear	-15.5	-16.8	1.34e-03	3.37e-03	4.71e-03
Log	-3.0	-15.2	7.87e-03	4.89e-03	1.28e-02
Truncated(InverseGamma{Float64}(invd: Gamma{Float64}(α=1.0, θ=1.0) θ: 1.					
Linear	-15.5	-16.6	1.41e-03	3.47e-03	4.87e-03
Log	-13.8	-15.0	8.03e-03	5.59e-03	1.36e-02
InverseGaussian{Float64}(μ=1.0, λ=1.0)					
Linear	-15.3	-16.2	3.32e-04	3.31e-03	3.64e-03
Log	-13.3	-14.6	6.10e-04	1.44e-02	1.50e-02
Truncated(InverseGaussian{Float64}(μ=1.0, λ=1.0); lower=0.01, upper=Inf)					
Linear	-15.3	-16.3	3.37e-04	3.28e-03	3.62e-03
Log	-12.2	-13.8	8.69e-04	7.41e-03	8.27e-03
JohnsonSU{Int64}(ξ=0, λ=1, γ=0, δ=1)					
Linear	-15.4	-16.6	2.37e-04	3.82e-04	6.19e-04
Log	-13.8	-14.9	6.70e-04	2.95e-04	9.65e-04
Truncated(JohnsonSU{Int64}(ξ=0, λ=1, γ=0, δ=1); lower=0.01, upper=Inf)					
Linear	-15.1	-16.1	2.84e-04	2.46e-04	5.31e-04
Log	-12.5	-13.7	6.94e-04	4.55e-04	1.15e-03
Kolmogorov()					
Linear	-15.3	-16.4	1.45e-04	9.54e-03	9.69e-03
Log	-0.8	-Inf	2.21e-04	1.06e-01	1.06e-01
Truncated(Kolmogorov(); lower=0.01, upper=Inf)					
Linear	-15.2	-16.3	1.88e-04	9.45e-03	9.64e-03
Log	Inf	Inf	3.29e-04	1.09e-02	1.12e-02
Kumaraswamy{Float64}(a=1.0, b=1.0)					
Linear	-16.3	-16.9	8.06e-05	7.82e-05	1.59e-04
Log	-16.0	-16.9	1.17e-04	1.19e-04	2.36e-04
Truncated(Kumaraswamy{Float64}(a=1.0, b=1.0); lower=0.01, upper=Inf)					
Linear	-16.0	-16.5	8.19e-05	9.56e-05	1.78e-04
Log	-16.0	-16.5	2.99e-04	1.34e-04	4.33e-04

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.

- Some distributions have trouble with log-space sampling in the tails. The Gumbel distribution is a good example. It works fine for values in between -2 and 2, but linear sampling remains stable outside that region, so we take the safer bet.

- 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. Another example is Cosine, where the log-space sampling gets an accuracy of $10^{-5.4}$ but the linear-space sampling accuracy is $10^{-10.9}$. While log-space is faster, we should surely use the linear sampling.

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.

The Exponential distribution has a problem with truncation. This should be fast and accurate but fails on both counts.

6 Competing Processes

In this section, we ask how to calculate the likelihood of a trajectory. A trajectory is a single element of the joint random variables $\{X_n, T_n\}$ where X_n is the state at time n and T_n is the time at time n. The curly-braces denote the set of all times n. Our goal is to be able to calculate this quantity numerically using a simulation.

Start by asking the cumulative distribution function of a single step of a trajectory. Think of distributions, defined relative to a start time at T_n. Each is set to fire at some next time x_a. The joint cumulative density function from one time T_n to the next, T_{n+1} can be decomposed into independent density functions.

$$\begin{aligned} F_n(x_1, x_2, \ldots, x_m) &= \int_0^{x_1} \cdots \int_0^{x_m} f(\xi_1, \xi_2, \ldots, \xi_m) d\xi_m \ldots d\xi_1 & (39) \\ &= \int_0^{x_1} \cdots \int_0^{x_m} \prod_a f_a(\xi_a) d\xi_m \ldots d\xi_1 & (40) \\ &= \prod_a \left[\int_0^{\xi_a} f_a(\xi_a) d\xi_m \ldots d\xi_1 \right] & (41) \end{aligned}$$

If the next time step is at time $t = T_{n+1} - T_n$, then the joint cumulative distribution function is $F_n(t) = \prod_a F_a(t)$.

Now that we have the cumulative distribution function, it may be easier to see the likelihood of a single step of the trajectory. Given competing processes, the likelihood is the probability that one particular trajectory, called α, fires between time t and $t + \delta t$, and all other transitions fire later. We'll use $S(t)$ for the survival and $h(t)$ for the hazard.

$$\begin{aligned} c_\alpha(t)dt &= f_\alpha(t) \prod_{a \neq \alpha} \left[\int_t^\infty f_a(\xi_a) dx_a \right] & (42) \\ &= f_\alpha(t) \prod_{a \neq \alpha} S_a(t) & (43) \end{aligned}$$

$$= \frac{f_\alpha(t)}{S_\alpha(t)} \prod_a S_a(t) \quad (44)$$

$$= h_\alpha(t) \prod_a S_a(t) \quad (45)$$

At time T_n, there is a continuous likelihood $c_\alpha(t)$ for each $\alpha \in a$. If we think about the sequence of $\{T_n\}$, the $c_\alpha(t)$ form a matrix called the core matrix.

We can translate the log-likelihood into Julia code. The first step is to express competing processes as long-lived competing processes. Account for the enabling times and for distributions that were enabled at a previous transition. Then rewrite this equation in terms of Julia functions.

Look, these values are remarkably similar to those computed for the Next Reaction method. And here's how they relate to Gillespie's method.

7 Common random numbers

Instead of calling `rand(rng, distribution)`, change it to `rand(rng, distribution, clock)`. Then keep track of a) the clock b) its n-th call and c) the survival associated with that call. Record choices and replay.

This works best with samplers that use fewer random numbers, such as the Next Reaction method. It's these lines we would want to replace. In the case of CRN, the input wouldn't be a random sample but would be a survival from which to determine the sample.

```
sample = rand(rng, distribution)
tau = te + sample
survival = survival_space(S, distribution, sample)
```

What if you used CRN and a direct method? I don't know.

8 Importance Sampling

Why would you calculate a likelihood? So that you can artificially increase the likelihood of trajectories with desired outcomes in order to improve their statistics.

8.1 Splitting

This version of importance sampling is simple to implement. When a trajectory is in the right direction, split it into multiple trajectories. Then, when counting results, down-weight those trajectories by the split. If one trajectory splits into 10, count the results by 1/10th.

A good example of splitting is a simulation that takes place on an energy landscape. Picture two valleys and a mountain pass between them. The simulation starts in one valley and rarely crosses the pass to the next valley. For those simulations that approach the pass, you could split the trajectories in order to increase the quality of statistics for the likelihood of reaching the other valley.

Splitting is implemented not in the sampler but in the driver of the simulation. You would copy the simulation state and the sampler state N times and then down-weight the contribution of each.

How would we implement that?

8.2 Exclusion

Here, we choose a transition and disallow it from firing. For instance, it's common in disease modeling to ask how large an outbreak would be, were there an outbreak. That's a conditional probability, conditioned on the disease never reaching extinction.

There are two ways to calculate an exclusion-sampled trajectory. Picture a herd of animals with only one infectious. It could either recover or infect another in the herd, and we'll exclude recovery, but we have to account for the decreased likelihood of the trajectory given that it excludes recovery.

One approach is to calculate the marginal probability of recovery and discount the ensuing draw. This works for sure.

The other approach is to leave recovery in the draws and use the draws themselves to do the downweighting. (Maybe?)

8.3 Weighting Probability

This is the most difficult version to do. You have to take the core matrix, aka the XXX. Then adjust each $c_\alpha(t)$ with a multiplicative weight. Then reconstitute the total survival, taking the hazard from the reweighted core matrix. Now draw from this distribution.

8.4 Examples

Example: Preventing extinction in disease simulation.

Example: Estimation of energy barrier.

9 Hamiltonian Monte Carlo

My favorite housing spread example.

10 Piecewise Deterministic Markov Processes

As long as the sampler can sample the distribution using the given methods, that distribution can be determined by any equation, including an ODE. In one version, the distribution is a delta function, and these completely work. You have to re-enable the distribution each time the ODE changes its prediction. In another version, there is still a continuous distribution, but it's determined by the ODE.

Show that a delta-function can be included in Fleck.