CTDE Documentation

Release 1.0

Drew Dolgert

April 13, 2016

Contents

A Julia library for stochastic simulation in continuous-time with time-dependent hazard rates.

Contents:

CHAPTER 1

Introduction

The CTDE library is designed to streamline the process of creating efficient simulations of a large class of stochastic processes as defined by Anderson and Kurtz[Anderson:2007]_. These processes naturally arise in many contexts including epidemiology *[Viet:2004]*, physiology, ecology, atmospheric sciences, reliability engineering and risk management. This broad range of applications suggest the value of designing a generic library for simulating complex semi-Markov processes, independent of the particular application area.

The unifying idea on which this library is based is that there typically are many different pathways for a complex system to evolve between timesteps. Each pathway can be viewed as an elementary stochastic process with a user specified time-dependent transition rates and a rule for modifying the overall internal state of the system. At each instant of time, these elementary processes "compete", figuratively speaking, for the chance to change the state of the whole system. Each time step in the simulation corresponds to an event – a "winner" is selected thus changing the internal state of the system and the sampling from the corresponding statistical distribution to determine the time increment. This competing process view provides a framework for users to develop simulations for complex models in an incremental manner.

It is easy to show that competing processes with exponentially distributed transition times have time-independent transition rates. This is the norm in some application areas such as chemical kinetics. In contrast, it is manifestly inappropriate for many biological applications such as physiology, ecology and epidemiology. For example, a classic paper by Stocks *[Stocks:1931]* clearly shows that the latent period for measles (the distribution times between infection and the appearance of symptoms) does not follow an exponential distribution. Stocks' raw data from cases in London circa 1931, along with optimal fits to exponential, gamma, Weibull, and log-normal distributions computed using the SciPy statistical library, are shown in *Figure 1. Distribution of latent periods for measles in London circa 1931*. The fit of the data to the exponential distribution is very poor while the fits to the other distributions are very good.

This simple example shows that exclusive reliance on exponential distributions may lead systematic biases in stochastic simulations of epidemiological process. Therefore, this library provides support for general semi-Markov models based on competing processes with general probability distributions of transition times.

1.1 Organization of library

It is implemented using three cooperating layers:

- **Finite state machine**: High-level interface for initializing the system, iterating over time steps and gathering relevant tracing data for post-processing.
- **Process Specification**: "Middleware" responsible for specifying a simulation with given constraints.
- **Competing Clocks Process**: Low-level coordination and bookkeeping related to the user-defined competing processes including distributions of transition times, modification of system state and various dependence relati

This organization has many practical advantages:

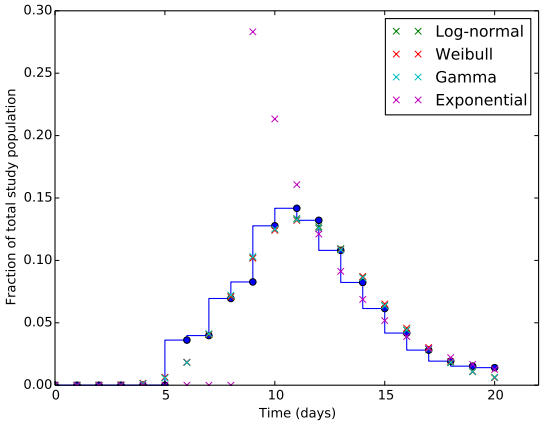

Fig. 1.1: Figure 1. Distribution of latent periods for measles in London circa 1931

- The competing clocks process layer can be viewed as a very efficient, general purpose, stochastic simulation engine that supports arbitrary statistical distributions for event times. This layer contains no model-specific user code, thus can be independently verified and validated.
- Typically, the itself model is completely defined by instantiating a state and transitions on that state, connected by a dependency graph. The underlying code does bookkeeping to track causality through the dependency graph.
- The library strictly enforces a separation of the static components that define the structural aspects of the model and the dynamic components that define the evolving state during a simulation. This separation makes it possible to detect many critical programming errors associated with multithreading at compile time.

1.2 Acknowledgements

This library was created by the Analytical Framework for Infectious Disease Dynamics (AFIDD) group at Cornell University in conjunction with the USDA Agricultural Research Service. This work was supported by the Science & Technology Directorate, Department of Homeland Security via interagency agreement no. HSHQDC-10-X-00138.

1.3 Availability and distribution

This library is in the public domain.

CHAPTER 2

Installation

This library is written in the Julia language.

```
Pkg.clone("git@github.com:adolgert/CTDE.jl.git")
```

It project requires the following packages, which should install when you run *Pkg.clone*.

- Distributions
- DataStructures
- Logging

The examples will also require the *Gadfly* package.

CHAPTER 3

Background

This addresses two main points, how to specify a model for the library using distributions defined by hazards and why such a specification, with its initial conditions, is sufficient to define the trajectory for a model.

3.1 The Hazard from Survival Analysis

3.1.1 Discrete case

The discrete case is much easier to understand than the continuous case because it can be explained without employing any results from calculus. Throughout this section, $\mathbf{X}$ will be assumed to real-valued random variable. For example, $\mathbf{X}$ could represent latent periods for measles.

It frequently happens that random samples of the real valued variables such as $\mathbf{X}$ are actually analyzed on a discrete scale. For example Stocks' data on latent periods of measles in *Figure 1. Distribution of latent periods for measles in London circa 1931* is based on daily visits by patients.

The (cumulative) distribution of $\mathbf{X}$ is defined as

$$F_X(k) = \mathcal{P}[x \leq k]$$

assuming $F_X(\infty) = 1$. The density can be expressed as the difference in adjacent values of the distribution

$$\begin{aligned} f_X(k) &= \mathcal{P}[X = k] && (3.1)\\ &= \mathcal{P}[X \leq k] - \mathcal{P}[X \leq k-1] && (3.2)\\ &= F_X(k) - F_X(k-1) && (3.3) \end{aligned}$$

For Stocks' data in *Figure 1. Distribution of latent periods for measles in London circa 1931*, the density at day k should be interpreted as the probability of the appearance of symptoms since the previous visit on day $k-1$.

The *hazard* is defined as the conditional probability that the value of a random selection from $\mathbf{X}$ equals k given it this value is already known to exceed $k-1$. Using the usual rules for computing conditional probabilities, the hazard is

given by the following ratio

$$to$$

$$h_X(k) = \mathcal{P}[X = k \mid k-1 < X] = \frac{f_X(k)}{1 - F_X(k-1)} \quad (3.4)$$

$$= \mathcal{P}[X = k \mid k-1 < X] \frac{f_X(k)}{1 - F_X(k-1)} \quad (3.4)$$

In the case of Stocks' data, the hazards shown in *Figure 2. Estimated hazards of latent periods for measles in London circa 1931* would correspond to the probability of symptoms appearing at day k given that the patient had not displayed symptoms at any previous visit. As time goes on, patients who have already developed symptoms effectively reduce the pool of patients in the study who are still in a state where they might first present symptoms on day k. This is the origin of the term in the denominator.

On any given day, the hazard for latent periods can be interpreted as the rate of appearance of symptoms per asymptomatic (infected but not yet symptomatic) patient per day. For example, the hazard inferred from the Weibull distribution is approximately 0.15 on day 10. In other words, 15% of the patients that are asymptomatic on day 9 will present symptoms when examined on day 10.

This interpretation is extremely important because it connects a hazard with a rate for a specific process, and that rate has well defined units of measurement. In addition, it clarifies how rate parameters should be estimated from observational data. Failure to account for the shrinking pool over time is commonplace. In this case it would lead to a systematic errors in the estimation of process rates, especially at long times when the depletion effect is most pronounced.

3.1.2 Continuous case

The random variable $\mathbf{X}$ is again assumed to be a real-valued, but the measurements will not be binned as above. The cumulative distribution not an integer k but a continuous time interval, τ.

$$F_X(\tau) = P[x \leq \tau]$$

assuming $F_X(\infty) = 1$. The density is the derivative of the cumulative distribution. The concept of the hazard is part of survival analysis, where survival is $G_X(\tau) = 1 - F_X(\tau)$, and represents the probability the random variable, a time interval, is longer than τ. One expression for the hazard is that the density of the random variable is equal to the probability it survives to a time τ multiplied by the hazard rate for firing at time τ, or, in probabilities,

$$P[\tau < x \leq \tau + d\tau]d\tau = P[\tau < x]P[\tau < x \leq \tau + d\tau + d\tau | \tau < x].$$

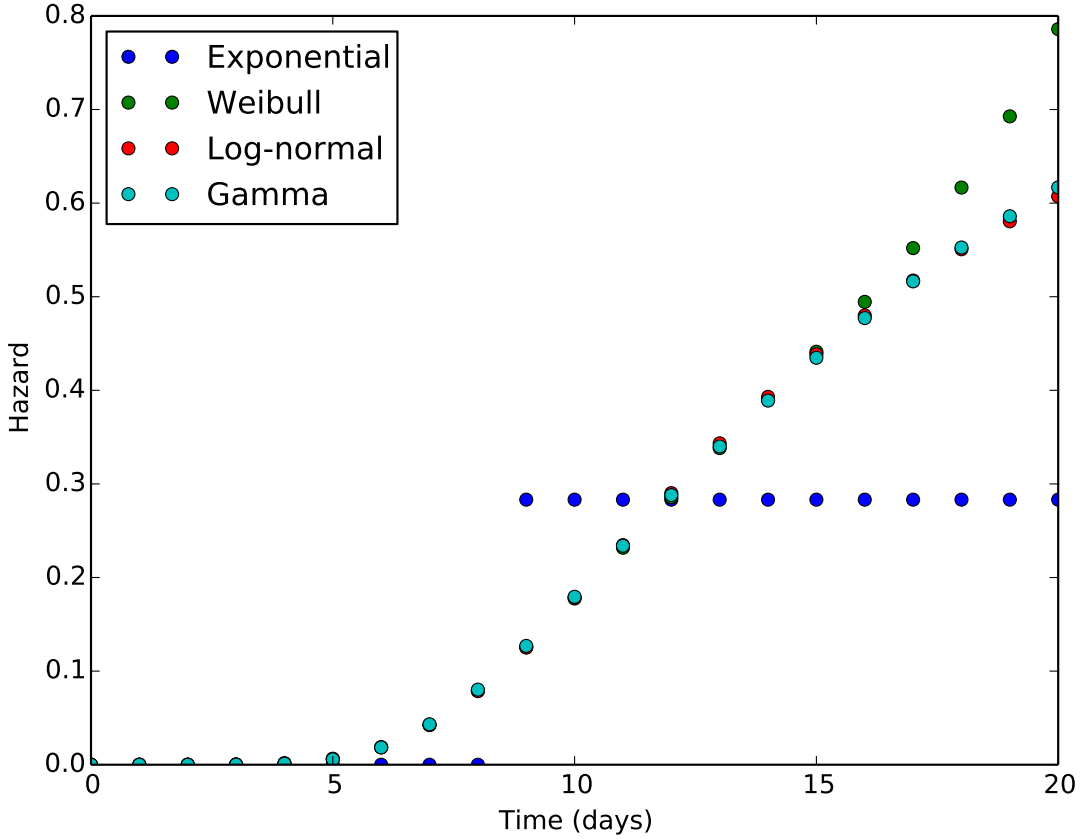

Fig. 3.1: Figure 2. Estimated hazards of latent periods for measles in London circa 1931

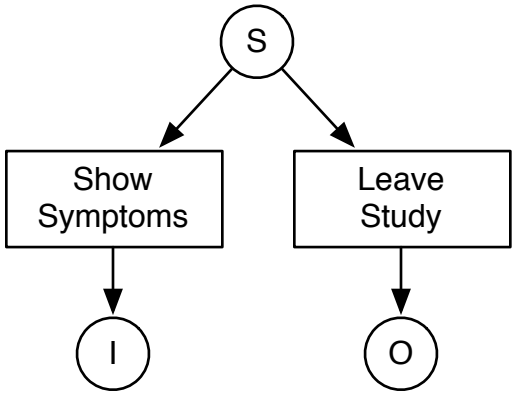

Fig. 3.2: Figure 3. Each participant of the Stocks study could either become symptomatic or leave the study. Focusing on the hazard accounts for the effect of those who leave.

Writing this same equation with its almost-sure equivalents defines the continuous hazard, $\lambda_X(\tau)$,

$$f_X(\tau) = G_X(\tau)\lambda_X(\tau).$$

This is a rearrangement away from the definition of the discrete case.

3.2 Finite State Machines Generate Trajectories

This library accepts a specification of a model in terms of hazards, an initial condition, and produces trajectories. This set of high-level steps to simulation (specify, initialize, step) has a well-defined abstraction called a *finite state machine.* It isn't the finite state machine familiar to programmers but a mathematical model, coming from category theory, for a particularly simple class of computing systems. At a conceptual level, a finite state machine can be considered a black box that receives a sequence of input signal and produces an output signal for each input signal. Internally, the black box maintains a *state* – some sort of finite summary representation of the sequence of input signals encountered so far. For each input signal, the box performs two operations. In both cases, the decision depends on the current internal state and the identity of the input signal just received.

- **Chose next state**
- **Generate output token**

It is helpful to view the finite state machine layer as a mechanism to simulate a *Markov chain* or *Markov process*.

3.3 Markov Chain for Discrete-Time Trajectories

Roughly speaking, a *Markov chain*, $\mathbf{X}$, is a probabilistic system that makes random jumps among a finite set of distinct states, $s_0, s_1, s_2, \ldots, s_N$ such that the probability of choosing the next state, X_{n+1} depends only on the current state, X_n. In mathematical terms, the conditional probabilities for state transitions must satisfy

$$\mathcal{P}[X_{n+1} = s_l | X_0 = s_i, X_1 = s_j, \ldots, X_n = s_k] = \mathcal{P}[X_{n+1} = s_l | X_n = s_k]$$

Since more distant history does not affect future behavior, Markov chains are sometimes characterized as *memoryless*.

This relation can be iterated to compute the conditional probabilities for multiple time steps

$$\mathcal{P}[X_{n+2} = s_m | X_n = s_k] = \sum_l \mathcal{P}[X_{n+2} = s_m | X_{n+1} = s_l] \mathcal{P}[X_{n+1} = s_l | X_n = s_k]$$

Note, the transition probabilities $\mathcal{P}[X_{n+1} = s_l | X_n = s_k]$ may depend on time (through the index n). These so-called time-inhomogeneous Markov chains arise when the system of interest is driven by external entities. Chains with time-independent conditional transition probabilities are called time-homogeneous. The dynamics of a time-homogeneous Markov chain is completely determined by the initial state and the transition probabilities. All processes considered in this document are time-homogeneous.

3.4 Markov Process for Continuous-Time Trajectories

A *Markov process* is a generalization of the Markov chain such that time is viewed as continuous rather than discrete. As a result, it makes sense to record the times at which the transitions occur as part of the process itself.

The first step in this generalization is to define a stochastic process $\mathbf{Y}$ that includes the transition times as well as the state, $Y_n = (s_j, t_n)$.

Fig. 3.3: Figure 4. **Realization of a continuous time stochastic process and associated Markov chain.**

The second step is to treat time on a truly continuous basis by defining a new stochastic process, $\mathbf{Z}$, from $\mathbf{Y}$ by the rule $Z_t = s_k$ in the time interval $t_n \leq t < t_{n+1}$ given $Y_n = (s_k, t_n)$. In other words, $\mathbf{Z_t}$ is a piecewise constant version of $\mathbf{Y}$ as shown in *Figure 4. Realization of a continuous time stochastic process and associated Markov chain.*

A realization of the process $\mathbf{Y}$ is defined by the closed diamonds (left end points) alone. Similarly, a realization of the process $\mathbf{Z_t}$ is illustrated by the closed diamonds and line segments. The closed and open diamonds at the ends of the line segment indicate that the segments include the left but not the right end points.

The memoryless property for Markov processes is considerably more delicate than in the case of Markov chain because the time variable is continuous rather than discrete. In the case of $\mathbf{Y}$, the conditional probabilities for state transitions of must satisfy

$$\mathcal{P}[Y_{n+1} = (s_l, t_{n+1})|Y_0 = (s_i, t_0), Y_1 = (s_j, t_1), \ldots, Y_n = (s_k, t_n)] = \mathcal{P}[Y_{n+1} = (s_l, t_{n+1})|Y_n = (s_k, t_n)]$$

The proper generalization of the requirement of time-homeogeneity stated previously for Markov chains is that joint probability be unchanged by uniform shifts in time

$$\mathcal{P}[Z_{t+\tau}|Z_{s+\tau}] = \mathcal{P}[Z_t|Z_s]$$

for $0 < s < t$ and $\tau > 0$. Stochastic processes with shift invariant state transition probabilities are called *stationary*.

When we examined hazard rates above, we were examining the rate of transitions for a Markov process. The overall probability of the next state of the Markov process is called the core matrix,

$$\mathcal{P}[Z_t|Z_s] = Q_{ij}(t_{n+1} - t_n)$$

indicating a state change between the states (s_i, s_j). The derivative of this is a rate,

$$q_{ij}(t_{n+1} - t_n) = \frac{dQ_{ij}(t_{n+1} - t_n)}{dt},$$

which is a joint distribution over states and time intervals. Normalization for this quantity sums over possible states and future times,

$$1 = \int_0^\infty \sum_j q_{ij}(s)ds.$$

The survival, in terms of the core matrix, is

$$G_i(\tau) = 1 - \int_0^\tau \sum_k q_{ik}(s)ds.$$

This means our hazard is

$$\lambda_{ij}(\tau) = \frac{q_{ij}(\tau)}{1 - \int_0^\tau \sum_k q_{ik}(s)ds}.$$

For the measles example, the set of future states j of each individual include symptomatic and all the possible other ways an individual leaves the study, so you can think of j = left town. In practice, we build a hazard in two steps. First, count the probability over all time for any one eventual state j. This is the same stochastic probability π_{ij} that is seen in Markov chains. Second, measure the distribution of times at which intervals enter each new state j, given that they are headed to that state. This is called the holding time, $h_{ij}(\tau)$, and is a conditional probability. Together, these two give us the core matrix,

$$q_{ij}(\tau) = \pi_{ij} h_{ij}(\tau).$$

Note that $h_{ij}(\tau)$ is a density whose integral $H_{ij}(\tau)$ is a cumulative distribution. If we write the same equation in terms of probabilities, we see that it amounts to separating the Markov process into a marginal and conditional distribution.

$$
\begin{aligned}
& & to \\
q_{ij}(\tau) &= \\
\frac{d}{d\tau} P[Z_t|Z_s] & \\
&= \\
\frac{d}{d\tau} P[s_j|s_i,t_n] P[t_{n-1} - t_n \leq \tau | s_i, s_j, t_n] & \\
&= \\
P[s_j|s_i,t_n] \frac{d}{d\tau} P[t_{n-1} - t_n \leq \tau | s_i, s_j, t_n] & \\
&= \\
\pi_{ij} \frac{d}{d\tau} H_{ij}(\tau) & \\
&= \\
\pi_{ij} h_{ij}(\tau) & \text{(3.-8)}
\end{aligned}
$$

$$
\begin{aligned}
& = \\
\frac{d}{d\tau} P[Z_t|Z_s] & = \\
\frac{d}{d\tau} P[s_j|s_i,t_n] P[t_{n-1} - t_n \leq \tau | s_i, s_j, t_n] & \\
P[s_j|s_i,t_n] \frac{d}{d\tau} P[t_{n-1} - t_n \leq \tau | s_i, s_j, t_n] & \\
\pi_{ij} \frac{d}{d\tau} H_{ij}(\tau) & = \\
\pi_{ij} h_{ij}(\tau) & \\
& \text{(3.-9)}
\end{aligned}
$$

Choosing the other option for the marginal gives us the waiting time formulation for the core matrix. It corresponds to asking first what is the distribution of times at which the next event happens, no matter which event, and then asking

which events are more likely given the time of the event.

$$
\begin{aligned}
& & to \\
q_{ij}(\tau) &= \\
\frac{d}{d\tau} P[Z_t | Z_s] & \\
&= \\
\frac{d}{d\tau} P[s_j | s_i, t_n, t_{n+1}] P[t_{n-1} - t_n \leq \tau | s_i, t_n] & \\
&= \\
\frac{d}{d\tau} (\Pi_{ij}(\tau) W_i(\tau)) & \\
&= \\
\pi_{ij}(\tau) \frac{d}{d\tau} W_i(\tau) & \\
&= \\
\pi_{ij}(\tau) w_i(\tau) & \text{(3.-9)}
\end{aligned}
$$

$$
\begin{aligned}
& = \\
\frac{d}{d\tau} P[Z_t | Z_s] & = \\
\frac{d}{d\tau} P[s_j | s_i, t_n, t_{n+1}] P[t_{n-1} - t_n \leq \tau | s_i, t_n] & \\
\frac{d}{d\tau} (\Pi_{ij}(\tau) W_i(\tau)) & = \\
\pi_{ij}(\tau) \frac{d}{d\tau} W_i(\tau) & = \\
\pi_{ij}(\tau) w_i(\tau) & \\
& \text{(3.-10)}
\end{aligned}
$$

While the waiting time density $w_i(\tau)$, is the derivative of the waiting time, we won't end up needing to relation $\pi_{ij}(\tau)$ to $\Pi_{ij}(\tau)$ when finding trajectories or computing hazards, so the more complicated relationship won't be a problem.

CHAPTER 4

CTDE API

4.1 Physical State

The physical state is composed of a set of disjoint substates. Each substate can be read and written with a key. That means anything you want to be state should have a *getindex* and *setindex!*. A simple example is *state=zeros(Int, 10)*. A more complicated example, this state is a count of bees at different plants.

```
type PhysicalState
    plants::Array{Int,2}
    bees::Array{Int,2}
end

function getindex(state::PhysicalState, x, y)
    state.bees[x, y]
end

function setindex!(state::PhysicalState, value, x, y)
    state.bees[x, y]=value
end
```

When specifying the process, intensities and firing functions can depend on the state only through a certain set of indexes, which you specify. In this example, the indices will be of the form of a tuple, *(x, y)* of type *Tuple{Int, Int}*.

4.2 Partial Process

The library defines the partial process, *CTDE.PartialProcess*, which is used to specify the simulation and compute trajectories.

```
process=PartialProcess(physical_state)
```

Then specify the process by adding transitions.

4.3 Transitions

Each transition has two logical parts, an intensity function which describes when the transition is enabled and how long after enabling it will fire, and a firing function which describes what the transition does to the state when it fires.

AddTransition!(process::PartialProcess, hazard::Intensity, hazard_dependencies, firing::Fu

- *process::PartialProcess* is the partial process just defined above.
- *hazard::Intensity* is a hazard rate, derived from the abstract class Intensity, as will be described below.
- *hazard_dependencies* is a list of indices into the physical state that will be passed to the intensity for this transition.
- *firing::Function* is a function that changes the state, as described below.
- *firing_dependencies* is a list of indices into the physical state that will be passed to the intensity for this transition.
- *name* is a friendly name to use when the process tells you what transition fired.
- *keywords* means that you can pass values such as *index=3* to the transition as messages to the samplers. Some samplers need hints about how to organize their work.
- Returns nothing.

4.4 Intensity

An intensity says when the transition is enabled or disabled and, when it is enabled, the distribution of times at which it may fire.

class `Intensity`
An *abstract* class for transition intensities. There are methods defined on this abstract class which help implement intensities.

`Enabled`(*intensity::Intensity*)
Returns a boolean to say whether the intensity is currently enabled. The helper method returns *intensity.enabled* for the object passed.

`Reset!(intensity::RecoverIntensity, time::Float64, state, keys..)`
When a transition fires, this is called to tell the intensity that it must forget all past observations of the state and determine, from the state at the values specified by the *keys*, what is the new distribution going forward.

`Update!(intensity::RecoverIntensity, time, state, keys...)`
This is the workhorse of the intensity distribution. Given the state at the given set of keys, the intensity chooses its current distribution for the hazard rate. It returns a symbol to report what happened. That symbol is either *:Unmodified*, *:Disabled*, *:Enabled*, or *:Modified*. The last choice, *:Modified*, means that the hazard was nonzero and is now nonzero but with a different distribution.

`Sample`(*intensity::Intensity*, *when::Float64*, *rng::MersenneTwister*)
samples the current distribution for the hazard, given that it has not yet fired by time *when*. This method calls *Sample(intensity.distribution)*, so a type which defines a *distribution* member doesn't need to reimpliment this method.

`Putative`(*intensity::Intensity*, *when::Float64*, *exponential_interval::Float64*)
integrates the current distribution for the hazard to determine at what time it will have used up an integrated hazard equal to *exponential_interval*. This is a way to sample distributions for Gibson and Bruck's Next Reaction Method or Anderson's method. This method calls *Putative(intensity.distribution)*, so a type which defines a *distribution* member doesn't need to reimpliment this method.

class `MemoryIntensity`
Many intensities can be defined with three quantities, 1) whether they are enabled given the current state, 2) given that they are enabled, the current set of parameters for their distribution, and 3) the particular distribution. This intensity is called a *memory* intensity because it sets the enabling time of the distribution when it is enabled, but it doesn't change that enabling time if, when the state changes, the parameters for the distribution change. It *remembers* the first enabling time.

MemoryIntensity (*invariant::Function*, *distribution*)
The distribution is a TransitionDistribution. The function is described just below.

invariant (*time*, *state*, *keys...*)
This function looks at the state, as indexed by the keys and returns a tuple with two values, whether the transition should be enabled, according to the current state, and, if so, what are the values of the parameters for the distribution of this transition. It looks like `(Bool, Array{Any,1})`.

class MemorylessIntensity
Many intensities can be defined with three quantities, 1) whether they are enabled given the current state, 2) given that they are enabled, the current set of parameters for their distribution, and 3) the particular distribution. This intensity is called a *memoryless* intensity because it sets the enabling time of the distribution not only it is enabled, but whenever a change of the state causes the parameters, those returned by the invariant, to change. When that happens, it *forgets* the original enabling time.

For example, a simulation of susceptible-infectious-susceptible individuals defines a recovery intensity that ensures that the last infectious person never recovers. It sounds mean, but it helps certain long-running simulations get data.

Listing 4.1: sis.jl

```
function Recover(state, who)
    state[who]=0
    [who]
end

function RecoverParameters(time, state, who, others...)
    enabled=(state[who]==1)
    # Forbid recovery if this is the only one infectious.
    found_nonzero=false
    for nz_idx = 1:length(others)
        if state[others[nz_idx]]>0
            found_nonzero=true
            break
        end
    end
    (enabled && found_nonzero, [1.0, 2.0])
end
    for midx = 1:N
        hazard=MemorylessIntensity(RecoverParameters,
                TransitionWeibull(1.0, 2.0))
                # TransitionExponential(parameters[:Gamma]))
        depends=[midx]
        for dep_idx=1:N
            if dep_idx!=midx
                push!(depends, dep_idx)
            end
        end
        # We add the index=transition_idx only for Direct methods.
        AddTransition!(process,
            hazard, depends,
            Recover, [midx],
            "r$midx", index=transition_idx )
```

There are helper methods, defined on the abstract type Intensity, which reduce the amount of code required for implementation.

```
type InfectIntensity <: Intensity
    distribution::TransitionDistribution
    enabled::Bool
```

```
    InfectIntensity(dist)=new(dist, 0.0, false)
end

function Reset!(intensity::InfectIntensity, time, state, who, whom)
    distribution.enabling_time=time
    Update!(intensity, time, state, who, whom)
end

function Update!(intensity::InfectIntensity, time, state, who, whom)
    modified=:Undefined
    enabled=(state[who]==1 && state[whom]==0)
    if enabled!=intensity.enabled
        if enabled
            intensity.distribution.enabling_time=time
            modified=:Enabled
        else
            modified=:Disabled
        end
        intensity.enabled=enabled
    else
        modified=:Unmodified
    end
    modified
end
```

In general, the intensity can depend on any state since it last fired or the start of the simulation. In practice, an intensity will examine the state to create parameters for the distribution. The *WrappedDistribution* is a good example of the interface distributions support.

4.5 Transition Distributions

The distributions an intensity needs have different methods from distributions in Julia's *Distributions* module.

class `TransitionDistribution`
This **abstract** class is a base class for the continuous univariate distributions used by intensities.

4.5.1 WrappedDistribution

class `WrappedDistribution`
This class uses the available distributions in the *Distributions* package to meet the API needed by the simulation. It's likely less efficient and possibly numerically inaccurate for some distributions, but here goes. Its members are *relative_distribution* and *enabling_time*.

`WrappedDistribution` (*dist::Distributions.ContinuousUnivariateDistribution*, *enabling_time::Float64*)
The constructor. Pass in a distribution from the Distributions package.

`Sample(distribution::WrappedDistribution, now::Float64, rng::MersenneTwister)`
This samples the distribution using the given random number generator. It calls *quantile(distribution, now, rand(rng))*.

`HazardIntegral` (*dist::WrappedDistribution*, *t1*, *t2*)
This integrates the hazard from time *t1* to time *t2* using *logccdf(dist, t1-te)-logccdf(dist, t2-te)* where *te* is the enabling time.

ImplicitHazardIntegral (*dist::WrappedDistribution*, *cumulative_hazard::Float64*, *when::Float64*)
This is the inverse of the hazard integral, so *ImplicitHazardIntegral(dist, HazardIntegral(dist, t1, t2), t1)=t2*.

EnablingTime (*dist::WrappedDistribution*)
Return the enabling time. It's a common parameter of all of these distributions.

EnablingTime!(dist::WrappedDistribution, time::Float64)
Set the enabling time.

Parameters (*dist::WrappedDistribution*)
This returns the parameters for the distribution. The exact set depends on the underlying distribution.

4.5.2 Exponential

TransitionExponential (λ*::Real*, *enabling_time::Real*)
The rate, λ, is the hazard rate, not a scale.

$$F(T) = 1 - exp\left[-\lambda(T - Te)\right]$$

4.5.3 Weibull

TransitionWeibull (θ*::Float64*, *k::Float64*)
The scale is θ. k is the exponent.

$$F(T) = 1 - exp\left[-\left(\frac{T - Te}{\theta}\right)^k\right]$$

4.5.4 Gamma

TransitionGamma (α, β, *enabling_time*)
α is the shape parameter, β the inverse scale parameter, also called a rate parameter.

$$f(t) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}$$

4.5.5 LogLogistic

TransitionLogLogistic (α, β)

$$F(t) = \left[1 + \left(\frac{t - t_e}{\alpha}\right)^{-\beta}\right]^{-1}$$

4.5.6 NelsonAalen

class NelsonAalenDistribution
Given a list of times the distribution either fired or was right-censored, meaning it failed to fire, this constructs an estimator of the distribution that can be sampled.

Because each transition in the process will either fire or be interrupted, this estimator can be used to ask whether the firing of each transition matches the expected distribution.

cdf (*dist::NelsonAalenDistribution*, *bypoint::Int*)

4.5.7 Empirical

class EmpiricalDistribution
This class estimates a distribution given samples of the times at which that distribution fired. First make the object and then use *push!* to add values. Finally, call *build!* before sampling from it. This is useful for testing distributions.

EmpiricalDistribution()
Constructor.

push!(ed::EmpiricalDistribution, value::Float64)
Add a sample to the list.

build!(ed::EmpiricalDistribution)
Internally, it needs to sort the list of samples.

mean(*ed::EmpiricalDistribution*)

variance(*ed::EmpiricalDistribution*)

kolmogorov_smirnov_statistic(*ed::EmpiricalDistribution*, *other*)
The *other* is a distrubition for which *cdf* is defined. This returns two values, the maximum difference between the two distributions and whether that maximum difference meets the 0.05 confidence interval for the hypothesis that they are the same distribution.

4.6 Firing Function

The firing function is a function that modifies state. Its signature has to be

FiringFunction(*state*, *keys...*)

the firing function returns a list of all substates which could be affected by having fired. While the list of hazard dependencies and firing dependencies is state, the list of what was affected by firing is not. As an example, recovery and infection for a disease model could look like If the firing function reads or writes mutable state other than that specified by the indices in keys, then the simulation will be incorrect.

```
function Recover!(state, who)
    state[who]=0
    [who]
end

function Infect!(state, who)
    state[who]=1
    [who]
end
```

4.7 Simulation Observer

Every time the simulation determines the next time and transition, it changes the state and then calls an observer with the signature

StateObserver(*state*, *affected_keys*, *clock_name*, *time::Float64*)

- *state* is the state of the system.
- *affected_keys* are those substates which were affected when the last transition fired.

- *clock_name* is the name given to that transition.
- *time* is the time at which that transition happened.
- Returns: The observer returns a boolean indicating whether the simulation may continue.

In practice, this function is a closure which adds data to a list of data, or writes that list to disk. For instance,

```
function Observer(out::ScreenObserver)
    function sobserve(state::Array{Int,1}, affected_keys, clock_name, time::Float64))
        AddEntry!(out, state[affected_keys[1], time])
    end
end
```

4.8 Sampler

Define a sampler. There are a few to choose among.

class `NextReactionHazards`
This sampler uses a variant of Gibson and Bruck's next reaction algorithm, described by Anderson and Kuntz.

`NextReactionHazards``()`
Constructor.

class `FixedDirect`
This is an optimized direct reaction sampler. It assumes there are a fixed number of transitions in the system and that every transition is given an ordinal with keywords of the form *index*=<*Int*>.

`FixedDirect``(`*N::Int*`)`
Constructor. *N* is the number of transitions in the system.

class `NaiveSampler`
This is appropriate only for simulations where no transition is ever reenabled after it fires or is disabled. It is equivalent to Next Reaction in this case.

4.9 Running a Simulation

Once the process is created and sampler chosen, a single function runs the simulation. In this example, the *MakeProcess* function creates state and adds transitions to the process.

```
rng=MersenneTwister(333333)
N=3
parameters=Dict(:Gamma =>1.0, :Beta => 1.0)
process, state=MakeProcess(N, parameters, rng)
observer=SamplingObserver(N, 1000)
sampler=NextReactionHazards()

RunSimulation(process, sampler, Observer(observer), rng)
```

CHAPTER 5

Examples

- Well-mixed SIR as a test of Ball and Nasell's predictions of SIR final size distributions.
- Bouncing Rabbits which move on a two-dimensional board, without stepping on each other, but they infect each other, sadly.
- Susceptible-Infectious-Susceptible This is a good test of the samplers because results are well-known. It plots the master probabilities for a regenerative process.
- Object-based Simulation This doesn't use dictionary keys in order to access substates of the physical state. It uses the objects themselves, which lends it to more free-form simulation.

CHAPTER 6

Detailed Description of Distributions

This describes implementation of distributions for sampling of distributions for stochastic simulation in continuous time. This kind of simulation makes specific demands on what calls a distribution must support, and those calls are different from what libraries provide. This is a guide for implementation of new distributions and a way to ensure that those implemented look correct. If something is wrong here, it matters, so file a bug report.

6.1 Notation

First, let's affix notation. The cumulative distribution function of every regular distribution can be written as an integral over its hazard rate, λ

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

All algorithms for stochastic simulation treat distributions as being defined in absolute time, specified as an enabling time, t_e,

$$F(t, t_e) = 1 - e^{-\int_0^{t-t_e} \lambda(s)ds}.$$

Working with distributions in absolute time is a simple shift of the time scale and will be ignored in further discussions, although the enabling time, t_e, will certainly appear in code. Fig. (6.1).

The density function is the derivative of the cumulative distribution function,

$$f(t) = \frac{dF(t)}{dt} = \lambda(t)e^{-\int_0^t \lambda(s)ds}.$$

The survival is

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

Because survival is multiplicative, we further label the survival from time t_0 to t_1 as

$$G(t_0, t_1) = \frac{G(t_1)}{G(t_0)} = e^{-\int_{t_0}^{t_1} \lambda(s)ds}$$

6.2 Requirements for a Continuous-Time Simulation

6.2.1 Shifted Sample

The First Reaction method requires that we sample a distribution given that we known it has not yet fired by a time t_0. The statement that it hasn't fired by time t_0 creates a new distribution from which to sample. If the old distribution

had the hazard $G(t) = G(0, t)$, it could be written as

$$G(0, t) = G(0, t_0)G(t_0, t).$$

It is this partial survival, since t_0, that we want to sample. Solving for $G(t_0, t)$ and subtracting both sides from 1,

$$1 - G(t_0, t) = \frac{G(0, t_0) - G(0, t)}{G(0, t_0)}.$$

Written in terms of the cumulative distribution functions, the cdf of the new distribution, which we'll call $F(t, t_0)$, is

$$F(t, t_0) = \frac{F(t) - F(t_0)}{1 - F(t_0)}$$

This kind of distribution could be sampled by a rejection method, but the default way to sample it is by inversion, which means generating a uniform random value between $[0, 1]$ and solving $U = F(t)$ for t. For Eq. (6.2.1), this becomes

$$\begin{aligned}
& to \\
U = & \\
F(t, t_0, t_e) & \\
= & \\
\frac{F(t, t_e) - F(t_0, t_e)}{1 - F(t_0, t_e)} & \\
U(1 - F(t_0, t_e)) = & \\
F(t, t_e) - F(t_0, t_e) & \\
F(t, t_e) = & \\
U(1 - F(t_0, t_e)) + F(t_0, t_e) & \\
F(t - t_e) = & \\
U(1 - F(t_0 - t_e)) + F(t_0 - t_e) & \\
t - t_e = & \\
F^{-1}\left[U(1 - F(t_0 - t_e)) + F(t_0 - t_e)\right] & \\
t = & \\
t_e + F^{-1}\left[U(1 - F(t_0 - t_e)) + F(t_0 - t_e)\right] & (6.0)
\end{aligned}$$

$$\begin{aligned}
& = \\
F(t, t_0, t_e) & = \\
\frac{F(t, t_e) - F(t_0, t_e)}{1 - F(t_0, t_e)} & = U(1 - F(t_0, t_e)) \\
F(t, t_e) - F(t_0, t_e) & = F(t, t_e) \\
U(1 - F(t_0, t_e)) & = F(t_0, t_e)F(t - t_e) \\
U(1 - F(t_0 - t_e)) & = F(t_0 - t_e)t - t_e \\
F^{-1}\left[U(1 - F(t_0 - t_e)) + F(t_0 - t_e)\right] & t \\
t_e + F^{-1}\left[U(1 - F(t_0 - t_e)) + F(t_0 - t_e)\right] & \\
(6.0) &
\end{aligned}$$

We will call this operation **SampleShifted.**

6.2.2 Hazard Rate for Next Reaction

The Next Reaction method requires sampling a distribution such that the quantile is saved, so that later adjustments to the distribution can use the same quantile.

During a simulation, the hazard rate, λ, is a function of the state of the system, $X(t)$. The state of the system only changes in jumps, so the hazard rate is effectively a series of distributions in time. For instance, a hazard rate, from enabling time T_0 to firing time T_3, might have three parts.

$$
\begin{aligned}
& & to \\
\lambda(\{X_0, T_0\}, t) &= h_0(t) \\
T_0 \leq t &< T_1 \\
\lambda(\{X_0, T_0, X_1, T_1\}, t) &= h_1(t) \\
T_1 \leq t &< T_2 \\
\lambda(\{X_0, T_0, X_1, T_1, X_2, T_2\}, t) &= h_2(t) \\
T_2 \leq t &< T_3
\end{aligned} \tag{6.1}
$$

$$
\begin{aligned}
&T_0 \leq t < T_1 \lambda(\{X_0, T_0, X_1, T_1\}, t) = h_1(t) \\
&T_1 \leq t < T_2 \lambda(\{X_0, T_0, X_1, T_1, X_2, T_2\}, t) = h_2(t) \\
&T_2 \leq t < T_3
\end{aligned} \tag{6.1}
$$

The algorithm therefore samples for a firing time from $h_0(t)$ as soon as the transition is enabled, but that time will turn out to be wrong (we call it a putative time). Later, the algorithm will resample using $h_1(t)$ using the original sample's quantile and taking time off the clock. If the first sample were by inversion, it would look like solving this equation for t (still ignoring enabling times),

$$U = 1 - \exp\left(\int_0^t h_0(s)ds\right).$$

Then a later sample would use the same U, but with knowledge that the distribution now contains a new part, $h_1(t)$,

$$U = 1 - \exp\left(-\int_0^{t_1} h_0(s)ds\right)\exp\left(-\int_{t_1}^{t} h_1(s)ds\right)$$

Anderson had the bright idea to write the quantile as an exponential quantile, $1 - U = e^{\ln(1-U)}$, so that the equation requires only addition of integrated hazards,

$$to$$

$$\ln(1-U) =$$
$$-\int_0^{t_1} h_0(s)ds - \int_{t_1}^{t} h_1(s)ds$$
$$\int_{t_1}^{t} h_1(s)ds =$$
$$-\ln(1-U) - \int_0^{t_1} h_0(s)ds. \quad (6.1)$$

=

$$=-\int_0^{t_1} h_0(s)ds - \int_{t_1}^{t} h_1(s)ds \int_{t_1}^{t} h_1(s)ds$$
$$-\ln(1-U) - \int_0^{t_1} h_0(s)ds.$$

(6.1)

As the underlying distribution, $h_i(t)$, changes, the right hand side gets smaller and smaller. Let's call the sum of all consumed hazard γ,

$$\gamma = \sum_i \int_{t_i}^{t_{i+1}} h_i(s)ds$$

The algorithm therefore needs three operations from the distribution.

1. **MeasuredSample**—Sample the distribution, returning the exponential quantile. Calling the random number generator, "rng," and the putative time t_p, it's

$$(\text{rng}, h_0(t)) \mapsto (t_p, -\ln(1-U)).$$

2. **ConsumeSample**—Consume remaining quantile for the next sample. If the sum of hazard in the past is called γ, then

$$(\gamma, h_i(t), t_i) \mapsto (\gamma', t_{i+1})$$

3. **Putative**—Generate a new putative time from the exponential quantile and the consumed hazard,

$$(-\ln(1-U), \gamma, t_i, h_i(t)) \mapsto p_t$$

The nice part about the first step is that there is no need to sample by inversion. Any sampling method will do, as long as the exponential quantile is calculated.

6.2.3 Cumulative Distributions for Next Reaction

The original form of the Next Reaction, by Gibson and Bruck, was written in terms, not of the hazards, but of the cumulative distribution functions. This form remains useful because some distributions are much simpler, or more accurate, to sample as cdfs instead of sampling from their hazard rates.

Returning to Eq. (6.2.2), this can be rewritten as

$$1 - U = \exp\left(-\int_0^{t_1} h_0(s)ds\right) \exp\left(-\int_{t_1}^{t} h_1(s)ds\right)$$

In terms of the survival functions, this becomes

$$1 - U = G_0(0, t_1)G_1(t_1, t)$$

If we wish to solve this for t, then, in terms of the survival, it looks like

$$G_1(t_1, t) = \frac{1 - U}{G_0(0, t_1)}$$

Writing the left-hand side as a cumulative distribution function requires the transformation

$$G(a, b) = \frac{G(b)}{G(a)} = \frac{1 - F(b)}{G(a)}$$

so we have

$$F_1(t) = 1 - G_1(t_1)\frac{1 - U}{G_0(0, t_1)}$$

This generalizes to many steps as

$$F_j(t) = 1 - G_j(t_j)(1 - U)\prod_i^j \frac{G_i(t_i)}{G_i(t_{i+1})}$$

Let's call the running product on the right δ,

$$\delta = \prod_i^j \frac{G_i(t_i)}{G_i(t_{i+1})}$$

Then the algorithm requires three operations

1. **MeasuredSample**—Sample the distribution, returning the quantile. Calling the random number generator, "rng," and the putative time t_p, it's

$$(\text{rng}, h_0(t)) \mapsto (t_p, 1 - U)\,.$$

2. **ConsumeSample**—Consume remaining quantile for the next sample.

$$(\delta_i, h_i(t), t_i) \mapsto (\delta_{i+1}, t_{i+1})$$

3. **Putative**—Generate a new putative time from the exponential quantile and the consumed hazard,

$$(1 - U, \delta_i, t_i, h_i(t)) \mapsto p_t$$

As you can see by comparison with the hazards version, it's simple to write the algorithm to accommodate either method of sampling. Therefore, each distribution can choose which interface to support.

6.3 Testing Distributions

6.3.1 EmpiricalDistribution

The type *EmpiricalDistribution* will estimate the mean and variance of a distribution, given samples from it. More importantly, it can compare those samples with a given functional form of a distribution using the Kolmogorov-Smirnov test. Often the distributions in this library have corresponding distributions in the main library, just with a different interface.

There are `test` functions in after the definition of each distribution to do these tests.

6.3.2 Emulating Next Reaction Sampling

Take a `MeasuredSample` and then repeatedly consume the sample, with the same distribution, and finally use `Putative` to get a value. It should agree with the original sample.

6.3.3 Exact Solution to Paradigmatic Systems

There are some stochastic systems, such as a random walk, or the SIR model, for which exact results are known for certain distributions.

The Ball Nasell example shows this. It turns out there is a minor error in Ball Nasell we found this way. The Weiss example makes a similar measurement.

6.4 Using Julia's Distributions

Julia's continuous univariate distributions support a fixed interface. In this section, we look at how to translate any distribution into the operations above.

In this table, `d` is the distribution.

`cdf(d,t)`	$F(t)$
`quantile(d,q)`	$F^{-1}(q)$
`logcdf(d,t)`	$\ln(F(t))$
`ccdf(d,t)`	$G(t)$
`logccdf(d,t)`	$-\int_0^t \lambda(s)ds$
`quantile(d,q)`	$F^{-1}(q)$
`cquantile(d,q)`	$F^{-1}(1-q) = G^{-1}(q)$
`invlogcdf(d,lp)`	$F^{-1}(e^{l_p})$
`invlogccdf(d,lp)`	$G^{-1}(e^{l_p})$ or $-\int_0^{t(l_p)} \lambda(s)ds = l_p$
`randexp(rng)`	$-\ln(1-U)$

A shifted sample, from Eq. (6.2.1), which ends with

$$t = t_e + F^{-1}\left[U(1 - F(t_0 - t_e)) + F(t_0 - t_e)\right]$$

transliterates to

Listing 6.1: wrappeddistribution.jl

```
function MeasuredSample(d::WrappedDistribution, t0::Float64, rng)
    U=rand(rng)
    te=d.enabling_time
    value=te+quantile(d.relative_distribution,
        U+(1-U)*cdf(d.relative_distribution, t0-te))
    (value, -log(1-U))
end
```

The next two pieces concern the hazard. The goal is to find the integral of the hazard between two absolute times, t_1 and t_2, where both are $t_{1,2} \geq t_0$. This is

$$\int_{t_1-t_e}^{t_2-t_e} \lambda(s)ds = \int_0^{t_2-t_e} \lambda(s)ds - \int_0^{t_1-t_e} \lambda(s)ds.$$

In terms of the given methods, this would be, noting the minus sign in the table,

Listing 6.2: wrappeddistribution.jl

```
function ConsumeSample(dist::WrappedDistribution, xa, start, finish)
    if xa<0
        xa=0
    end
    xa+HazardIntegral(dist, start, finish)
end

function HazardIntegral(dist::WrappedDistribution, t1, t2)
    # logccdf is log(1-cdf(d, x))
    rel=dist.relative_distribution
    te=dist.enabling_time
    logccdf(rel, t1-te)-logccdf(rel, t2-te)
end
```

Looking back to Eq. (6.2.2),

$$\int_{t_1}^{t} h_1(s)ds = -\ln(1-U) - \int_0^{t_1} h_0(s)ds, \quad (6.\text{-}15)$$

we can label `xa` the combination of the exponential quantile and the sums of integrals on the right-hand side.

Listing 6.3: wrappeddistribution.jl

```
function Putative(dist::WrappedDistribution, when,
        interval, consumed_interval)
    ImplicitHazardIntegral(dist, interval-consumed_interval, when)
end

function ImplicitHazardIntegral(dist::WrappedDistribution, xa, t0)
    rel=dist.relative_distribution
    te=dist.enabling_time
    t=te+invlogccdf(rel, -xa+logccdf(rel, t0-te))
    @assert(t>=t0)
    t
end
```

6.5 Exponential

The exponential distribution is constructed with a hazard rate, even though the internal distributions object uses a scale, which is $\theta = 1/\lambda$,

Listing 6.4: exponentialdistribution.jl

```
type TransitionExponential <: TransitionDistribution
    hazard::Float64
    enabling_time::Float64
    TransitionExponential(rate::Real)=new(rate, 0.0)
end
```

It doesn't matter how we sample the distribution, as long as we return its quantile. This samples using `Base.randexp`, which uses the ziggurat method for a sample that's much faster than inversion. The value returned by `randexp` is equivalent to $-\ln(1-U)$.

Listing 6.5: exponentialdistribution.jl

```
function MeasuredSample(d::TransitionExponential, now::Float64, rng)
    u=randexp(rng)
    (now+u/d.hazard, u)
end
```

The hazard integral for constant hazards is $(t_2 - t_1)\lambda$.

Listing 6.6: exponentialdistribution.jl

```
function HazardIntegral(dist::TransitionExponential, start, finish)
    @assert(finish>=start)
    (finish-start)*dist.hazard
end

function ConsumeSample(dist::TransitionExponential, xa, start, finish)
    xa = xa<0 ? 0 : xa
    xa+HazardIntegral(dist, start, finish)
end
```

Even inverting the hazard integral is an increment with a multiplication.

Listing 6.7: exponentialdistribution.jl

```
function ImplicitHazardIntegral(dist::TransitionExponential,
        cumulative_hazard, current_time)
    @assert(cumulative_hazard>=0)
    current_time+cumulative_hazard/dist.hazard
end

function Putative(dist::TransitionExponential, when,
        interval, consumed_interval)
    ImplicitHazardIntegral(dist, interval-consumed_interval, when)
end
```

6.6 Weibull

Like the exponential distribution, the Weibull distribution has an integrable hazard rate, which makes implementation straightforward. Unfortunately, the use of the parameter λ in the definition of the Weibull is at odds with our use of it as a hazard rate, but it's just a scale parameter here.

$$F(t) = 1 - \exp\left[\left(\frac{t - t_e}{\lambda}\right)^k\right]$$

The constructor uses this cdf.

Listing 6.8: weibulldistribution.jl

```
type TransitionWeibull <: TransitionDistribution
    parameters::Array{Float64,1}
    te::Float64
    TransitionWeibull(lambda, k)=new([lambda, k], 0.0)
end
```

From the cdf, the hazard rate is

$$\Lambda(t) = \int_0^t \lambda(s)ds = \left(\frac{t - t_e}{\lambda}\right)^k$$

The inverse, where we ask when the integral equals $l_u = -\ln(1 - U)$, is

$$t = t_e + \lambda l_u^(1/k)$$

The version in the code is overachieving because it allows for shifting the distribution.

Listing 6.9: weibulldistribution.jl

```
function MeasuredSample(distribution::TransitionWeibull, now::Float64, rng)
    (λ, k)=distribution.parameters
    d=now-distribution.te
    value=0
    mlogU=randexp(rng)
    if d>0
```

Given that the hazard is already integrated in Eq. (6.6), integrating the hazard is algebraic.

Listing 6.10: weibulldistribution.jl

```
function HazardIntegral(dist::TransitionWeibull, last, now)
    (λ, k)=dist.parameters
    if now-dist.te>eps(Float64)
        return ((now-dist.te)/λ)^k - ((last-dist.te)/λ)^k
    else
        return 0::Float64
    end
end

function ConsumeSample(dist::TransitionWeibull, xa, start, finish)
    xa=(xa<0) ? 0 : xa
    xa+HazardIntegral(dist, start, finish)
end
```

Listing 6.11: weibulldistribution.jl

```
function ImplicitHazardIntegral(dist::TransitionWeibull,
        cumulative_hazard, when)
    (λ, k)=dist.parameters
    if when-dist.te>eps(Float64)
        return dist.te + λ*(cumulative_hazard + ((when-dist.te)/λ)^k)^(1.0/k)
    else
        return dist.te + λ*(cumulative_hazard)^(1.0/k)
    end
end

function Putative(dist::TransitionWeibull, when,
        interval, consumed_interval)
    ImplicitHazardIntegral(dist, interval-consumed_interval, when)
end
```

6.7 Log-Logistic

Working from wikipedia, because Gradstein and Ryzhik is too heavy to lift.

$$F(x;\alpha,\beta) = \frac{1}{1+(x/\alpha)^{-\beta}}.$$

We shift this to

$$F(t,t_e) = \frac{1}{1+((t-t_e)/\alpha)^{-\beta}}.$$

The pdf is

$$f(x;\alpha,\beta) = \frac{(\beta/\alpha)(x/\alpha)^{\beta-1}}{(1+(x/\alpha)^{\beta})^2}.$$

The quantile is

$$F^{-1}(p;\alpha,\beta) = \alpha\left(\frac{p}{1-p}\right)^{1/\beta}.$$

Survival

$$G(t) = 1 - F(t) = \frac{1}{1+(t/\alpha)^{\beta}}.$$

Hazard

$$\lambda(t) = \frac{f(t)}{G(t)} = \frac{(\beta/\alpha)(t/\alpha)^{\beta-1}}{1+(t/\alpha)^{\beta}}$$

Lastly, we need `invlogccdf(d,lp)`, which is $G_d^{-1}(e^{l_p})$, or $-\int_0^t (l_p)\lambda(s)ds = l_p$.

$$\begin{aligned} l_p &= \ln(G(t)) \\ e^{l_p} &= G(t) \\ e^{l_p} &= \frac{1}{1+(t/\alpha)^{\beta}} \\ e^{-l_p} &= 1+(t/\alpha)^{\beta} \\ (t/\alpha)^{\beta} &= 1-e^{-l_p} \\ t/\alpha &= (1-e^{-l_p})^{1/\beta} \\ t &= \alpha(1-e^{-l_p})^{1/\beta} \end{aligned}$$

6.8 Gamma

We will define paramaters from the shape α and rate β.

$$f(x) = \frac{\beta^{\alpha}}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}$$

where

$$\Gamma(t) = \int_0^{\infty} x^{t-1} e^{-x} dx.$$

The CDF is

$$F(x;\alpha,\beta) = \frac{\gamma(\alpha,\beta x)}{\Gamma(\alpha)}$$

where γ is the (lower) incomplete gamma function,

$$\gamma(x;\alpha) = \int_0^x t^{\alpha-1} e^{-t} dt$$

In our back pocket, from `Boost::Math`, are $\Gamma(x)$, $\ln(|\Gamma(x)|)$, digamma, which is

$$\psi(x) = \frac{d}{dx}\ln(\Gamma(x)) = \frac{\Gamma'(x)}{\Gamma(x)},$$

gamma ratio, which is $\Gamma(a)/\Gamma(b)$, gamma delta ratio, which is $\Gamma(a)/\Gamma(a+\Delta)$, and the set of incomplete gamma functions. In order, they are normalized lower incomplete, normalized upper, incomplete full (non-normalized) lower incomplete, and full (non-normalized) upper incomplete gamma functions.

$$\begin{aligned} \text{gamma\_p}(a,z) &= \frac{\gamma(a,z)}{\Gamma(a)} = \frac{1}{\Gamma(a)}\int_0^z t^{a-1}e^{-t}dt \\ \text{gamma\_q}(a,z) &= \frac{\Gamma(a,z)}{\Gamma(a)} = \frac{1}{\Gamma(a)}\int_z^0 t^{a-1}e^{-t}dt \\ \text{tgamma\_lower}(a,z) &= \gamma(a,z) = \int_0^z t^{a-1}e^{-t}dt \\ \text{tgamma}(a,z) &= \Gamma(a,z) = \frac{1}{\Gamma(a)}\int_z^0 t^{a-1}e^{-t}dt \end{aligned}$$

There are a set of inverses of incomplete gamma functions and derivatives of incomplete gamma functions. OK, back to what we need.

$$\begin{aligned} F(x;\alpha,\beta) = &\quad \text{gamma\_p}(\alpha,\beta x) \\ F^{-1}(y;\alpha,\beta) = &\quad \text{gamma\_p\_inv}(\alpha,y)/\beta \end{aligned}$$

The hazard integral, in terms of the cdf, is

$$\begin{aligned} \int_{t_1-t_e}^{t_2-t_e} \lambda(s)ds = &\quad -\ln(1-F(t_2-t_e)) + \ln(1-F(t_1-t_e)) \\ = &\quad \ln\left[\frac{1-F(t_1-t_e)}{1-F(t_2-t_e)}\right]. \end{aligned}$$

Can we simplify this into something provided?

$$\begin{aligned} \int_{t_1-t_e}^{t_2-t_e} \lambda(s)ds = &\quad \ln\left[\frac{1-\frac{\gamma(\alpha,\beta(t_1-t_e))}{\Gamma(\alpha)}}{1-\frac{\gamma(\alpha,\beta(t_2-t_e))}{\Gamma(\alpha)}}\right] \\ = &\quad \ln\left[\frac{\Gamma(\alpha)-\gamma(\alpha,\beta(t_1-t_e))}{\Gamma(\alpha)-\gamma(\alpha,\beta(t_2-t_e))}\right] \\ \gamma(\alpha,\beta(t_1-t_e)) = &\quad \int_0^{\beta(t_1-t_e)} t^{\alpha-1}e^{-t}dt \end{aligned}$$

It looks like we might do best just with

```
Ga=tgamma(a)
hazint(te, t1, t2)=log((Ga-tgamma_lower(a,b*(t1-te)))/
    (Ga-tgamma_lower(a,b*(t2-te))))
```

Our other goal for Gamma distributions is to get the inverse hazard. This can be seen as two steps. First find the integral

$$l_p = -x + \left[\int_0^{t0-t_e} \lambda(s)ds\right].$$

Then solve for t' in

$$l_p = -\int_0^{t'-t_e} \lambda(s)ds.$$

Or, we could write this as

$$l_e = e^{-x}e^{-\int_0^{t0-t_e}\lambda(s)ds} = e^{-x}(1-F(t_0-t_e))$$

and

$$l_e = e^{-\int_0^{t'-t_e}\lambda(s)ds} = 1-F(t'-t_e).$$

All at once,

$$\begin{aligned} F(t'-t_e) = &\quad 1-e^{-x}(1-F(t_0-t_e)) \\ t' = &\quad t_e + F^{-1}\left(1-e^{-x}(1-F(t_0-t_e))\right). \\ F(t_0-t_e) = &\quad \text{gamma\_p}(\alpha,\beta(t_0-t_e)) \\ F^{-1}(y) = &\quad \text{gamma\_p\_inv}(\alpha,y)/\beta \end{aligned}$$

So here is our inverse hazard integral.

```
quad=1-exp(-x)*(1-gamma_p(a,b*(t0-te)))
tp=te + gamma_p_inv(a, quad)/b
```

6.9 Uniform Distribution

Maybe this one will be easier. This distribution has two parameters, a start time and an end time, t_a and t_b. The pdf is constant, $f(t) = 1/(t_b - t_a)$ between $t_a \leq t < t_b$. The CDF is just the integral of that, $F(t) = (t - t_a)/(t_b - t_a)$. The integrated hazard will have nonzero cases for for $t_1 < t_a < t_2 < t_b$, $t_1 < t_a < t_b < t_2$, $t_a < t_1 < t_2 < t_b$, $t_a < t_1 < t_b < t_2$. It is zero for $t_1 < t_2 < t_a$ and $t_a < t_b < t_1 < t_2$

$$\int_{t_1-t_e}^{t_2-t_e} \lambda(s)ds = \ln\left[\frac{1 - F(t_1 - t_e)}{1 - F(t_2 - t_e)}\right]$$

If $t_a \leq t_n - t_e < t_b$, then $F(t_n - t_e) = (t_n - t_e - t_a)/(t_b - t_a)$. Otherwise it is 0 or 1. It should never be the case that a uniform distribution does not fire before t_b. The hazard integral always sums over time already past in the simulation. Nevertheless, it will be necessary to check for overflow near t_b, and it would help to keep the two logs separated, instead of in the fraction.

What about the inverse of the hazard integral? $F^{-1}(x) = t_a + (t_b - t_a)x$ Therefore, for $t_a \leq t_0 - t_e$,

$$t' = t_e + t_a + (t_b - t_a)\left[1 - e^{-x}\left(1 - \frac{t_0 - t_e - t_a}{t_b - t_a}\right)\right]$$

and for $t_0 - t_e < t_a$,

$$t' = t_e + t_a + (t_b - t_a)\left[1 - e^{-x}\right]$$

6.10 Triangular Distribution

The cumulative distribution function for the triangular distribution with endpoints a and b and midpoint m is

$$\begin{array}{ll} \dfrac{(t-a)^2}{(b-a)(m-a)} & a \leq t \leq m \\ 1 - \dfrac{(b-t)^2}{(b-a)(b-m)} & m < t \leq b. \end{array}$$

This makes the survival

$$\begin{array}{ll} 1 - \dfrac{(t-a)^2}{(b-a)(m-a)} & a \leq t \leq m \\ \dfrac{(b-t)^2}{(b-a)(b-m)} & m < t \leq b. \end{array}$$

6.10.1 Simple Sample

The cutoff is at $t = m$, which is

$$\begin{aligned} U' &= \frac{(m-a)^2}{(b-a)(m-a)} \\ &= \frac{m-a}{b-a} \end{aligned}$$

so first check whether U is greater than that. Then, for U less than that,

$$\begin{aligned} t &= a + [U(b-a)(m-a)]^{1/2} \quad U \leq U' \\ t &= b - [(1-U)(b-a)(b-m)]^{1/2} \quad U' < U \end{aligned}$$

6.10.2 Shifted Sample

If this is sampled after some time, x, then the thing we want to invert is

$$U = \frac{F(t) - F(x)}{G(x)}$$

so

$$F(t) = UG(x) + F(x)$$

which, for $a < t \leq m$ and $a < x \leq m$, is

$$\begin{aligned} \frac{(t-a)^2}{(b-a)(m-a)} &= U\left[1 - \frac{(x-a)^2}{(b-a)(m-a)}\right] + \frac{(x-a)^2}{(b-a)(m-a)} \\ (t-a)^2 &= U(b-a)(m-a) + (1-U)(x-a)^2 \\ t &= a + \left[U(b-a)(m-a) + (1-U)(x-a)^2\right]^{1/2} \end{aligned}$$

For $m < t \leq b$ and $m < x \leq b$, this is

$$\begin{aligned} 1 - \frac{(b-t)^2}{(b-a)(b-m)} &= U\frac{(b-x)^2}{(b-a)(b-m)} + 1 - \frac{(b-x)^2}{(b-a)(b-m)} \\ -(b-t)^2 &= U(b-x)^2 - (b-x)^2 \\ t &= b - (b-x)\sqrt{1-U} \end{aligned}$$

In the case that $m < t \leq b$ and $a < x \leq m$, the result is

$$\begin{aligned} 1 - \frac{(b-t)^2}{(b-a)(b-m)} &= U\left[1 - \frac{(x-a)^2}{(b-a)(m-a)}\right] + \frac{(x-a)^2}{(b-a)(m-a)} \\ 1 - \frac{(b-t)^2}{(b-a)(b-m)} &= U + \frac{(x-a)^2(1-U)}{(b-a)(m-a)} \\ \frac{(b-t)^2}{(b-a)(b-m)} &= (1-U) - \frac{(x-a)^2(1-U)}{(b-a)(m-a)} \\ (b-t)^2 &= (1-U)(b-a)(b-m) - \frac{(x-a)^2(1-U)(b-m)}{(m-a)} \\ t &= b - \left[(1-U)(b-a)(b-m) - \frac{(x-a)^2(1-U)(b-m)}{(m-a)}\right]^{1/2} \end{aligned}$$

6.10.3 Sampling from Quantile

The equation we have to solve is

$$F(t) = 1 - G(t_j)(1-u)\prod_i \frac{G_i(t_i)}{G_i(t_{i+1})},$$

given to us as

$$F(t) = 1 - G(t_j)(1-u)\gamma,$$

so, in terms of survivals, it's

$$G(t) = G(t_j)(1-u)\gamma$$

The value on the right is all known. If $t_j < m$, the cutoff is

$$G' = \frac{b-m}{(b-a)(b-m)}$$

Below that,

CHAPTER 7

References

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Bibliography

[Anderson:2007] 4. (a) Anderson, "A modified next reaction method for simulating chemical systems with time dependent propensities and delays.," J. Chem. Phys., vol. 127, no. 21, p. 214107, Dec. 2007.

[Gillespie:1978] 4. (a) Gillespie, "Monte Carlo simulation of random walks with residence time dependent transition probability rates," J. Comput. Phys., vol. 28, no. 3, pp. 395–407, Sep. 1978.

[Grimm2010] 22. Grimm, U. Berger, D. L. DeAngelis, J. G. Polhill, J. Giske, and S. F. Railsback, "The ODD protocol: A review and first update," Ecol. Modell., vol. 221, no. 23, pp. 2760–2768, Nov. 2010.

[Hartig2011] 6. Hartig, J. M. Calabrese, B. Reineking, T. Wiegand, and A. Huth, "Statistical inference for stochastic simulation models–theory and application.," Ecol. Lett., vol. 14, no. 8, pp. 816–27, Aug. 2011.

[Howard:1971] R. A. Howard, "Dynamic Probabilistic Systems. Vol. II: Semi-Markov and Decision Processes" (J. Wiley and Sons, 1971).

[Howard2007] 18. (a) Howard, Dynamic Probabilistic Systems: Semi-Markov and Decision Processes. Mineola, NY: Dover, 2007.

[Meyers:2013] Scott Meyers, The new C++.

[Pyke1961] 18. Pyke, "Markov Renewal Process: Definition and Preliminary Properties," Ann. Math. Stat., vol. 32, no. 4, pp. 1231–1242, 1961.

[Stocks:1931] P. Stocks, "Incubation period of measles," British Medical Journal 1(3655): p. 157.

[Viet:2004] A.-F. Viet, C. Fourichon, H. Seegers, C. Jacob, and C. Guihenneuc-Jouyaux, "A model of the spread of the bovine viral-diarrhoea virus within a dairy herd.," Prev. Vet. Med., vol. 63, no. 3–4, pp. 211–36, May 2004.