# Diffusing Ideas

## Software, noise and building mathematical toys

**Robert J. Hardwick**

February 9, 2023

# Introduction

*Diffusing Ideas* is a book of research exploration and software development which I have written for the interest of mathematically-inclined programmers and computational scientists. It's the output of many interrelated projects over several years which have sought to generalise the computational mathematics of simulating, statistically inferring, manipulating and automatically controlling stochastic phenomena as far as possible.

The book accompanies a lot of new open-source scientific software, written predominantly in Python and Go. A major motivation for creating these new tools is to prepare a foundation of code from which to develop new and more complex applications. I also hope that the resulting framework will enable anyone to explore and study new phenomena effectively, regardless of their scientific background.

The need to properly test all this software has also provided a wonderful excuse to study and play with an extensive range of mathematical toy models. I've chosen these models based on my broad background of interests, but also to illustrate the remarkable cross-disciplinary applicability of stochastic processes. However, it's often true that mathematical formalities can obscure the computations that a programmer must implement. So, while I've tried to be as ambitious as possible with the level of technical sophistication in these models, I've also tried to write the expressions in a computer-friendly way.[1]

A quick note on the code: any software that I describe in this book (including the software which compiles the book itself) will always be shared under a MIT License in a public Git repository.[2] Forking these repositories and submitting pull requests for new features or applications is strongly encouraged too, though I apologise in advance if I don't follow these up very quickly as all of this work has to be conducted independently in free time, outside of work hours.

No quest would be complete without a guide, so to end this introduction, I think it makes the most sense to outline the key milestones and their motivations within the context of the overall research project. My core aims, which comprise the four major parts of this book, are answers to the following set of interdependent research questions:

**Part 1.** How do we simulate a general set of stochastic phenomena?

**Part 2.** How do we then learn/identify the answer to **Part 1** from real-world data?

**Part 3.** How do we simulate a general set of control policies to interact with the answer to **Part 1**?

**Part 4.** How do we then optimise the answer to **Part 3** to achieve a specified control objective?

[1] E.g., more 'matrices' and less 'operators'.

[2] The repositories will always be somewhere on this list: https://github.com/umbralcalc?tab=repositories.

# Table of contents

# Part 1. How do we simulate a general set of stochastic phenomena?

1

# Building a generalised simulator

**Concept.** To design and build a generalised simulation engine that is able to generate samples from a 'Pokédex' of possible stochastic processes that a researcher might encounter. A 'Pokédex' here is just my fanciful description for a very general class of multidimensional stochastic processes that pop up everywhere in taming the mathematical wilds of real-world phenomena, and which also leads to a name for the software itself: the 'stochadex'. With such a thing pre-built and self-contained, it can become the basis upon which to build generalised software solutions for a lot of different problems. For the mathematically-inclined, this chapter will require the introduction of a new formalism which we shall refer back to throughout the book. For the programmers, the public Git repository for the code that is described in this chapter can be found here: https://github.com/umbralcalc/stochadex.

## 1.1 Computational formalism

Before we dive into software design of the stochadex, we need to mathematically define the general computational approach that we're going to take in order to be able to describe as general a set of stochastic phenomena as possible. From experience, it seems reasonable to start by writing down the following formula which describes iterating some arbitrary process forward in time (by one finite step) and adding a new row each to some matrix $X' \to X$

$$X^i_{\mathtt{t}+1} = F^i_{\mathtt{t}+1}(X', \mathtt{t}) \,, \tag{1.1}$$

where: $i$ is an index for the dimensions of the 'state' space; $\mathtt{t}$ is the current time index for either a discrete-time process or some discrete approximation to a continuous-time process; $X$ is the next version of $X'$ after one timestep (and hence one new row has been added); and $F^i_{\mathtt{t}+1}(X', \mathtt{t})$ as the latest element of an arbitrary matrix-valued function. As we shall discuss shortly, $F^i_{\mathtt{t}+1}(X', \mathtt{t})$ may represent not just operations on deterministic variables, but also on stochastic ones. There is also no requirement for the function to be continuous.

The basic computational idea here is illustrated in Fig. 1.1; we iterate the matrix $X$ forward in time by a row, and use its previous version $X'$ as an entire matrix input into a function which populates the elements of its latest rows. Pretty simple! But why go to all this trouble of storing

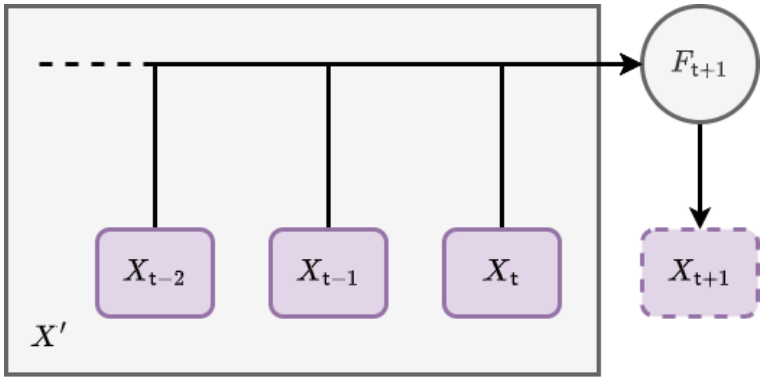


Figure 1.1: Graph representation of Eq. (1.1).

matrix inputs for previous values of the same process? It's true that this is mostly redundant for *Markovian* phenomena, i.e., processes where their only memory of their history is the most recent value they took. However, for a large class of stochastic processes a full memory[1] of past values is essential to consistently construct the sample paths moving forward. This is true in particular for *non-Markovian* phenomena, where the latest values don't just depend on the immediately previous ones but can depend on values which occured much earlier in the process as well.

For more complex physical models and integrators, the distinct notions of 'numerical timestep' and 'total elapsed continuous time' will crop up quite frequently. Hence, before moving on further details, it will be important to define the total elapsed time variable $t(\mathsf{t})$ for processes which are defined in continuous time. Assuming that we have already defined some function $\delta t(\mathsf{t})$ which returns the specific change in continuous time that corresponds to the step $\mathsf{t}-1 \to \mathsf{t}$, we will always be able to compute the total elapsed time through the relation

$$t(\mathsf{t}) = \sum_{\mathsf{t}'=0}^{\mathsf{t}} \delta t(\mathsf{t}') \,. \tag{1.2}$$

This seems a lot of effort, no? Well it's important to remember that our steps in continuous time may not be constant, so by defining the $\delta t(\mathsf{t})$ function and summing over it we can enable this flexibility in the computation.

So, now that we've mathematically defined a really general notion of iterating the stochastic process forward in time, it makes sense to discuss some simple examples. For instance, it is frequently possible to split $F$ up into deteministic (denoted $D$) and stochastic (denoted $S$) matrix-valued functions like so

$$F^i_{\mathsf{t}+1}(X', \mathsf{t}) = D^i_{\mathsf{t}+1}(X', \mathsf{t}) + S^i_{\mathsf{t}+1}(X', \mathsf{t}) \,. \tag{1.3}$$

In the case of stochastic processes with continuous sample paths, it's also nearly always the case with mathematical models of real-world systems that the deterministic part will at least contain the term $D^i_{\mathsf{t}+1}(X', \mathsf{t}) = X^i_{\mathsf{t}}$ because the overall system is described by some stochastic differential equation. This is not a really requirement in our general formalism, however.

[1] Or memory at least within some window.

What about the stochastic term? For example, if we wanted to consider a *Wiener process noise*, we can define $W^i_{\mathsf{t}}$ is a sample from a Wiener process for each of the state dimensions indexed by $i$ and our formalism becomes

$$S^i_{\mathsf{t}+1}(X', \mathsf{t}) = W^i_{\mathsf{t}+1} - W^i_{\mathsf{t}} \,. \tag{1.4}$$

One draws the increments $W^i_{\mathsf{t}+1} - W^i_{\mathsf{t}}$ from a normal distribution with a mean of 0 and a variance equal to the length of continuous time that the step corresponded to $\delta t(\mathsf{t}+1)$, i.e., the probability density $P^i_{\mathsf{t}+1}(x)$ of the increments $x^i = W^i_{\mathsf{t}+1} - W^i_{\mathsf{t}}$ is

$$P^i_{\mathsf{t}+1}(x) = \mathsf{NormalPDF}[x^i; 0, \delta t(\mathsf{t}+1)] \,. \tag{1.5}$$

Note that for state spaces with dimensions $> 1$, we could also allow for non-trivial cross-correlations between the noises in each dimension.

In another example, to model *geometric Brownian motion noise* we would simply have to multiply $X^i_{\mathsf{t}}$ to the Wiener process like so

$$S^i_{\mathsf{t}+1}(X', \mathsf{t}) = X^i_{\mathsf{t}}(W^i_{\mathsf{t}+1} - W^i_{\mathsf{t}}) \,. \tag{1.6}$$

Here we have implicitly adopted the Itô interpretation to describe this stochastic integration. Given a carefully-defined integration scheme other interpretations of the noise would also be possible with our formalism too, e.g., Stratonovich[2] or others within the more general '$\alpha$-family' [1, 2, 3].

We can imagine even more general processes that are still Markovian. One example of these in a single-dimension state space would be

$$S^0_{\mathsf{t}+1}(X', \mathsf{t}) = g[W^0_{\mathsf{t}+1}, t(\mathsf{t}+1)] - g[W^0_{\mathsf{t}}, t(\mathsf{t})] \tag{1.7}$$

$$= \left[\frac{\partial g}{\partial t} + \frac{1}{2}\frac{\partial^2 g}{\partial x^2}\right]\delta t(\mathsf{t}+1) + \frac{\partial g}{\partial x}(W^0_{\mathsf{t}+1} - W^0_{\mathsf{t}}) \,, \tag{1.8}$$

where $g(x,t)$ is some continuous function of its arguments which has been expanded out with Itô's Lemma on the second line. Note also that the computations in Eq. (1.8) could be performed with numerical derivatives in principle if the function were extremely complicated.

Let's now look at a more complicated type of noise. For example, *fractional Brownian motion* $[B_H]_{\mathsf{t}}$ with Hurst exponent $H$. Following Ref. [4], we can simulate this process in one of our state space dimensions by modifying a standard Wiener process like so

$$S^0_{\mathsf{t}+1}(X', \mathsf{t}) = \frac{(W^0_{\mathsf{t}+1} - W^0_{\mathsf{t}})}{\delta t(\mathsf{t}+1)} \int_{t(\mathsf{t})}^{t(\mathsf{t}+1)} \mathrm{d}t' \frac{(t'-t)^{H-\frac{1}{2}}}{\Gamma(H+\frac{1}{2})} {}_2F_1\left(H - \frac{1}{2}; \frac{1}{2} - H; H + \frac{1}{2}; 1 - \frac{t'}{t}\right) , \tag{1.9}$$

where $S^0_{\mathsf{t}+1}(X', \mathsf{t}) = [B_H]_{\mathsf{t}+1} - [B_H]_{\mathsf{t}}$. The integral in Eq. (1.9) can be approximated using an appropriate numerical procedure (like the trapezium rule). In the expression above ${}_2F_1$ and $\Gamma$ are the ordinary hypergeometric and gamma functions, respectively.

So far we have mostly been discussing noises with continuous sample paths, but we can easily adapt our computation to discontinuous sample paths as well. For instance, *Poisson process noises* would generally take the form

$$S^i_{\mathsf{t}+1}(X', \mathsf{t}) = [N_\lambda]^i_{\mathsf{t}+1} - [N_\lambda]^i_{\mathsf{t}} \,, \tag{1.10}$$

[2] Which would implictly give $S^i_{\mathsf{t}+1}(X', \mathsf{t}) = (X^i_{\mathsf{t}+1} + X^i_{\mathsf{t}})(W^i_{\mathsf{t}+1} - W^i_{\mathsf{t}})/2$ for Eq. (1.6).

where $[N_\lambda]^i_{\mathsf{t}}$ is a sample from a Poisson process with rate $\lambda$. One can think of this process as counting the number of events which have occured up to the given interval of time, where the intervals between each succesive event are exponentially distributed with mean $1/\lambda$. Such a simple counting process could be simulated exactly by explicitly setting a newly-drawn exponential variate to the next continuous time jump $\delta t(\mathsf{t}+1)$ and iterating the counter. Other exact methods exist to handle more complicated processes involving more than one type of 'event', such as the Gillespie algorithm [5] — though these techniques are not always be applicable in every situation.

Is using step size variation always possible? If we consider a *time-inhomogeneous Poisson process noise*, which would generally take the form

$$S^i_{\mathsf{t}+1}(X', \mathsf{t}) = [N_{\lambda(\mathsf{t}+1)}]^i_{\mathsf{t}+1} - [N_{\lambda(\mathsf{t})}]^i_{\mathsf{t}}\,, \tag{1.11}$$

the rate $\lambda(\mathsf{t})$ has become a deterministically-varying function in time. In this instance, it likely not be accurate to simulate this process by drawing exponential intervals with a mean of $1/\lambda(\mathsf{t})$ because this mean could have changed by the end of the interval which was drawn. An alternative approach (which is more generally capable of simulating jump processes but is an approximation) first uses a small time interval $\tau$ such that the most likely thing to happen in this period is nothing, and then the probability of the event occuring is simply given by

$$p(\mathsf{event}) = \frac{\lambda(\mathsf{t})}{\lambda(\mathsf{t}) + \frac{1}{\tau}}\,. \tag{1.12}$$

This idea can be applied to phenomena with an arbitrary number of events and works well as a generalised approach to event-based simulation, though its main limitation is worth remembering; in order to make the approximation good, $\tau$ often must be quite small and hence our simulator must churn through a lot of steps. From now on we'll refer to this well-known technique as the *rejection method*.

There are a few extensions to the simple Poisson process that introduce additional stochastic processes. *Cox (doubly-stochastic) processes*, for instance, are basically where we replace the time-dependent rate $\lambda(\mathsf{t})$ with independent samples from some other stochastic process $\Lambda(\mathsf{t})$. For example, a Neyman-Scott process [6] can be mapped as a special case of this because it uses a Poisson process on top of another Poisson process to create maps of spatially-distributed points. In our formalism, a two-state implementation of the Cox process noise would look like

$$S^0_{\mathsf{t}+1}(X', \mathsf{t}) = \Lambda(\mathsf{t}+1) \tag{1.13}$$

$$S^1_{\mathsf{t}+1}(X', \mathsf{t}) = [N_{S^0_{\mathsf{t}+1}}]^i_{\mathsf{t}+1} - [N_{S^0_{\mathsf{t}}}]^i_{\mathsf{t}}\,. \tag{1.14}$$

Another extension is *compound Poisson process noise*, where it's the count values $[N_\lambda]^i_{\mathsf{t}}$ which are replaced by independent samples $[J_\lambda]^i_{\mathsf{t}}$ from another probability distribution, i.e.,

$$S^i_{\mathsf{t}+1}(X', \mathsf{t}) = [J_\lambda]^i_{\mathsf{t}+1} - [J_\lambda]^i_{\mathsf{t}}\,. \tag{1.15}$$

Note that the rejection method of Eq. (1.12) can be employed effectively to simulate any of these extensions as long as a sufficiently small $\tau$ is chosen.

All of the examples we have discussed so far are Markovian. Given that we have explicitly constructed the formalism to handle non-Markovian phenomena as well, it would be worthwhile going some examples of this kind of process too. *Self-exciting process noises* would generally take

the form

$$S^0_{\mathsf{t}+1}(X', \mathsf{t}) = \mathcal{I}_{\mathsf{t}+1}(X', \mathsf{t}) \tag{1.16}$$

$$S^1_{\mathsf{t}+1}(X', \mathsf{t}) = [N_{S^0_{\mathsf{t}+1}}]^i_{\mathsf{t}+1} - [N_{S^0_{\mathsf{t}}}]^i_{\mathsf{t}}\,, \tag{1.17}$$

where the stochastic rate $\mathcal{I}_{\mathsf{t}+1}(X', \mathsf{t})$ now depends on the history explicitly. Amongst other potential inputs we can see, e.g., Hawkes processes [7] as an example of above by substituting

$$\mathcal{I}_{\mathsf{t}+1}(X', \mathsf{t}) = \mu + \sum_{\mathsf{t}'=0}^{\mathsf{t}} \gamma[t(\mathsf{t}) - t(\mathsf{t}')](S^1_{\mathsf{t}'} - S^1_{\mathsf{t}'-1})\,, \tag{1.18}$$

where $\gamma$ is the 'exciting kernel' and $\mu$ is some constant background rate.

Note that this idea of integration kernels could also be applied back to our Wiener process. For example, another type of non-Markovian phenomenon that frequently arises across physical and life systems integrates the Wiener process history like so

$$S^0_{\mathsf{t}+1}(X', \mathsf{t}) = W^0_{\mathsf{t}+1} - W^0_{\mathsf{t}} \tag{1.19}$$

$$S^1_{\mathsf{t}+1}(X', \mathsf{t}) = \frac{1}{T}\sum_{\mathsf{t}'=0}^{\mathsf{t}} e^{-\frac{t(\mathsf{t})-t(\mathsf{t}')}{T}}(S^0_{\mathsf{t}'} - S^0_{\mathsf{t}'-1})\,, \tag{1.20}$$

where $T$ is some decay coefficient which quantifies the length of memory in continuous time.

So we've introduced the basic elements of our computational formalism and demonstrated how flexible the approach can be in simulating just about any stochastic phenomenon imaginable. Before progressing to algorithm design, it will be helpful to discuss some useful concepts that should enable us analyse the system later on in the book.

## 1.2 Useful probabilistic concepts

The general stochastic process that we defined with Eq. (1.1) also has an implicit *master equation* associated to it which fully describes the time evolution of the *probability density function* $P_{\mathsf{t}+1}(x)$ of the most recent matrix row $x = X_{\mathsf{t}+1}$ at time $\mathsf{t}$. This can be written as

$$P_{\mathsf{t}+1}(x) = \frac{1}{\mathsf{t}}\sum_{\mathsf{t}'=0}^{\mathsf{t}} \int_{\omega_{\mathsf{t}'}} \mathrm{d}x' P_{\mathsf{t}'}(x') P_{(\mathsf{t}+1)\mathsf{t}'}(x|x')\,, \tag{1.21}$$

where at the moment we are assuming the state space is continuous in each dimension and $P_{(\mathsf{t}+1)\mathsf{t}'}(x|x')$ is the conditional probability that the matrix row at time $(\mathsf{t}+1)$ will be $x = X_{\mathsf{t}+1}$ given that the row at time $\mathsf{t}'$ was $x' = X_{\mathsf{t}'}$. Eq. (1.21) implies an iterative relationship between probabilities which I've also illustrated in Fig. 1.2.

The factor of $1/\mathsf{t}$ in Eq. (1.21) is a normalisation factor — this just normalises the sum of all probabilities to 1 given that there is a sum over $\mathsf{t}'$. Note that, if the process is defined over continuous time, we would need to replace

$$\frac{1}{\mathsf{t}}\sum_{\mathsf{t}'=0}^{\mathsf{t}} \to \frac{1}{t(\mathsf{t})}\sum_{\mathsf{t}'=0}^{\mathsf{t}} \delta t(\mathsf{t}')\,. \tag{1.22}$$

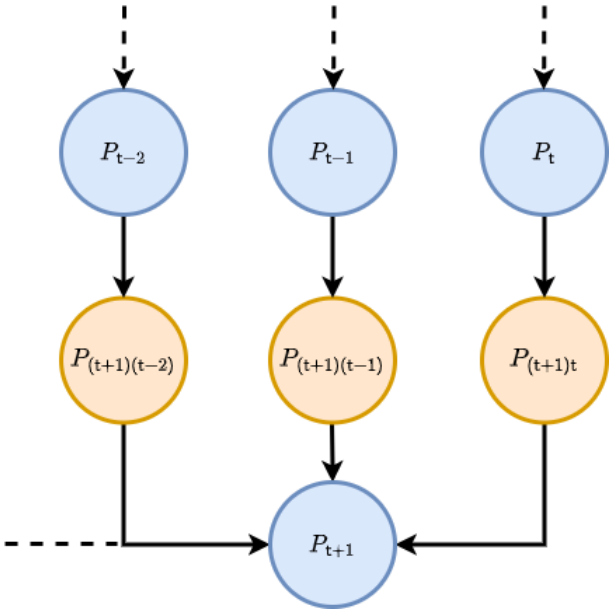


Figure 1.2: Graph representation of Eq. (1.21).

But what is $\omega_t$? You can think of this as just the domain of possible $x'$ inputs into the integral which will depend on the specific stochastic process we are looking at.

What if we wanted the joint distribution of both rows $P_{(t+1)t'}(x, x')$? One way to obtain this would be to extend Eq. (1.21) such that both matrix rows are marginalised over separately like so

$$P_{(t+1)t'}(x, x') = \frac{1}{(t'-1)t} \sum_{t''=0}^{t} \sum_{t'''=0}^{t'-1} \int_{\omega_{t''}} dx'' \int_{\omega_{t'''}} dx''' P_{t''t'''}(x'', x''') P_{(t+1)t''}(x|x'') P_{t't'''}(x'|x''') \,. \tag{1.23}$$

Given Eqs. (1.21) and (1.23) it's also possible to work out what the conditional probabilities would look like using the simple relation

$$P_{(t+1)t'}(x|x') = \frac{P_{(t+1)t'}(x, x')}{P_{t'}(x')} \,. \tag{1.24}$$

The implicit notation in Eq. (1.21) can hide some staggering complexity. To analyse the system in more detail, we can also do a kind of Kramers-Moyal expansion [8, 9] to approximate it like this

$$\begin{aligned} P_{t+1}(x) = \frac{1}{t} \sum_{t'=0}^{t} P_{t'}(x) - \frac{1}{t} \sum_{t'=0}^{t} \sum_{i=1}^{d} \frac{\partial}{\partial x^i} \left[ \alpha^i_{(t+1)t'}(x) P_{t'}(x) \right] \\ + \frac{1}{2t} \sum_{t'=0}^{t} \sum_{i=1}^{d} \sum_{j=1}^{d} \frac{\partial}{\partial x^i} \frac{\partial}{\partial x^j} \left[ \beta^{ij}_{(t+1)t'}(x) P_{t'}(x) \right] + \dots \,, \end{aligned} \tag{1.25}$$

in which we have assumed that the state space is $d$-dimensional. In this expansion, we also needed to define these new integrals

$$\alpha^i_{(\mathsf{t}+1)\mathsf{t}'}(x) = \int_{\omega_{\mathsf{t}'}} \mathrm{d}x' (x'-x)^i P_{(\mathsf{t}+1)\mathsf{t}'}(x'|x) \tag{1.26}$$

$$\beta^{ij}_{(\mathsf{t}+1)\mathsf{t}'}(x) = \int_{\omega_{\mathsf{t}'}} \mathrm{d}x' (x'-x)^i (x'-x)^j P_{(\mathsf{t}+1)\mathsf{t}'}(x'|x)\,. \tag{1.27}$$

So the matrix notation of Eq. (1.21) can indeed hide a very complicated calculation. Truncating the expansion at second-order, Eq. (1.25) tells us that there can be first and second derivatives contributing to the flow of probability to each element of the row $x = X_{\mathsf{t}+1}$ which depend on every element of the matrix $X'$. The probability does indeed *flow*, in fact. We can define a quantity known as the 'probability current' $J_{(\mathsf{t}+1)\mathsf{t}'}(x)$ from $\mathsf{t}'$ to $(\mathsf{t}+1)$ which illustrates this through the following continuity relation

$$P_{\mathsf{t}+1}(x) - \frac{1}{\mathsf{t}}\sum_{\mathsf{t}'=0}^{\mathsf{t}} P_{\mathsf{t}'}(x) = -\frac{1}{\mathsf{t}}\sum_{\mathsf{t}'=0}^{\mathsf{t}} J_{(\mathsf{t}+1)\mathsf{t}'}(x)\,. \tag{1.28}$$

By inspection of Eq. (1.25) we can therefore also deduce that

$$J^i_{(\mathsf{t}+1)\mathsf{t}'}(x) = \alpha^i_{(\mathsf{t}+1)\mathsf{t}'}(x) P_{\mathsf{t}'}(x) - \frac{1}{2}\sum_{j=1}^{d} \frac{\partial}{\partial x^j}\left[\beta^{ij}_{(\mathsf{t}+1)\mathsf{t}'}(x) P_{\mathsf{t}'}(x)\right] + \dots\,. \tag{1.29}$$

What would happen if we assumed that $\alpha$ and $\beta$ were just arbitrary time-dependent functions? For example, let's make the following assumptions

$$\alpha^i_{(\mathsf{t}+1)\mathsf{t}'}(x) = \mu^i(\mathsf{t}') - x^i \tag{1.30}$$

$$\beta^{ij}_{(\mathsf{t}+1)\mathsf{t}'}(x) = 2\Sigma^{ij}(\theta, \mathsf{t}')\,, \tag{1.31}$$

where $\mu(\mathsf{t}')$ is an arbitrary vector-valued function of the timestep and $\Sigma(\theta, \mathsf{t}')$ is an arbitrary matrix (often known as the 'diffusion tensor') which depends on both the timestep and a set of hyperparameters $\theta$. If we now also assume stationarity of $P_{\mathsf{t}'}(x) = P_{\mathsf{t}''}(x)$ for any $\mathsf{t}'$ and $\mathsf{t}''$ such that

$$P_{\mathsf{t}+1}(x) = \frac{1}{\mathsf{t}}\sum_{\mathsf{t}'=0}^{\mathsf{t}} P_{\mathsf{t}'}(x)\,, \tag{1.32}$$

we can solve Eq. (1.25) to obtain the following stationary solution

$$P_{\mathsf{t}'}(x) = \mathsf{MultivariateNormalPDF}[x; \mu(\mathsf{t}'), \Sigma(\theta, \mathsf{t}')]\,. \tag{1.33}$$

I hid a bit of the detail in that last step; the solution also required the identification that the flow of probability between timesteps vanishes uniquely for each and every $\mathsf{t}'$ such that $J_{(\mathsf{t}+1)\mathsf{t}'}(x) = 0$.

It's possible to take this derivation a bit further by expanding Eq. (1.23) in a similar fashion, truncating it to second-order, assuming only time-dependent terms and then solving it in the stationary limit. By plugging this solution (and its corresponding marginal distribution equivalent)

into Eq. (1.24), it's possible to get something that looks like this conditional distribution

$$P_{(\mathsf{t}+1)\mathsf{t}'}(x|x') \propto \exp\Big\{ -\frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d}\big[x - f(\mathsf{t}+1)\big]^i [K^{-1}(\theta, \mathsf{t}+1, \mathsf{t}+1)]^{ij}\big[x - f(\mathsf{t}+1)\big]^j + \sum_{i=1}^{d}\sum_{j=1}^{d}\big[x - f(\mathsf{t}+1)\big]^i [K^{-1}(\theta, \mathsf{t}+1, \mathsf{t}')]^{ij}\big[x' - f(\mathsf{t}')\big]^j \Big\}, \tag{1.34}$$

where $K(\theta, \mathsf{t}+1, \mathsf{t}')$ is some arbitrary covariance matrix that encodes how the correlation structure varies with the between compared states at two different timesteps and $K^{-1}(\theta, \mathsf{t}+1, \mathsf{t}')$ denotes taking its inverse. Eq. (1.34) may look a bit familiar to some readers who like using Gaussian processes from the machine learning literature [10] — this version is a *generative* model for a future $x$ value, in contrast to the more standard equation used to *infer* values of $f$. These are two sides of the same coin though.

What other processes can be described by Eq. (1.21)? For Markovian phenomena, the equation no longer depends on timesteps older than the immediately previous one, hence the expression reduces to just

$$P_{\mathsf{t}+1}(x) = \int_{\omega_\mathsf{t}} \mathrm{d}x' P_\mathsf{t}(x') P_{(\mathsf{t}+1)\mathsf{t}}(x|x')\,. \tag{1.35}$$

It's also easy to show that Eq. (1.25) naturally simplifies into the more usually applied Kramers-Moyal expansion when considering a Markovian process — you just remove the sum over $\mathsf{t}'$ and the $1/\mathsf{t}$ normalisation factor.

Note that an analog of Eq. (1.21) exists for discrete state spaces as well. We just need to replace the integral with a sum and the schematic would look something like this

$$P_{\mathsf{t}+1}(x) = \frac{1}{\mathsf{t}}\sum_{\mathsf{t}'=0}^{\mathsf{t}}\sum_{\omega_{\mathsf{t}'}} P_{\mathsf{t}'}(x') P_{(\mathsf{t}+1)\mathsf{t}'}(x|x')\,, \tag{1.36}$$

where we note that the $P$'s in the expression above all now refer to *probability mass functions*. In the even-simpler case where $x$ is just a vector of binary 'on' or 'off' states, we just have

$$P^i_{\mathsf{t}+1} = \frac{1}{\mathsf{t}}\sum_{\mathsf{t}'=0}^{\mathsf{t}}\sum_{j=1}^{d} P^j_{\mathsf{t}'} P^{ij}_{(\mathsf{t}+1)\mathsf{t}'}\,, \tag{1.37}$$

where $P^i_{\mathsf{t}'}$ now represents the probability that element $x^i = 1$ (is 'on') at time $\mathsf{t}'$.

## 1.3 Data types and desirable features

So I've proposed a computational formalism and done a bit of analysis on it to demonstrate that it can cope with a variety of different stochastic phenomena. Now I think we're ready to summarise what we want the stochadex software package to be able to do. If we begin with the obvious first set of criteria; we want to be able to freely configure the iteration function $F$ of Eq. (1.1) and the timestep function $t$ of Eq. (1.2) so that any process we want can be described. The point at which

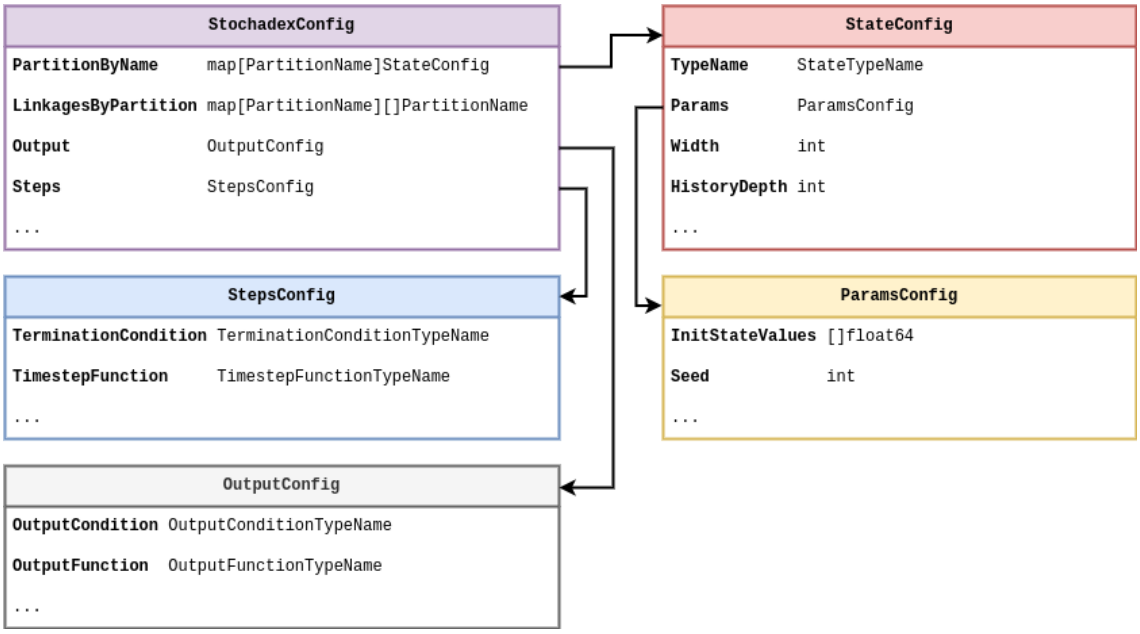


Figure 1.3: A relational summary of the core data types in the stochadex.

a simulation stops can also depend on some algorithm termination condition which the user should be able to specify up-front.

Once someone has written the code to create these functions for the stochadex, I want to then be able to recall them in future only with configuration files while maintaining the possibility of changing their simulation run parameters. This flexibility should facilitate our uses for the simulation later in the book, and from this perspective it also makes sense that the parameters should include the random seed and initial state value.

The state history matrix $X$ should be configurable in terms of its number of rows — what we'll call the 'state width' — and its number of columns — what we'll call the 'state history depth'. If we were to keep increasing the state width up to millions of elements or more, it's likely that on most machines the algorithm performance would grind to a halt when trying to iterate over the resulting $X$ within a single thread. While we don't really want to be talking about the algorithm or its performance yet, we can still pre-empt the requirement that $X$ should represented in computer memory by a set of partitioned matrices which are all capable of communicating to one-another downstream. In this paradigm, I'd like the user to be able to configure which state partitions are able to communicate with each other without having to write any new code.

For convenience, it seems sensible to also make the outputs from stochadex runs configurable. A user should be able to change the form of output that they want through, e.g., some specified function of $X$ at the time of outputting data. The times that the stochadex should output this data can also be decided by some user-specified condition so that the frequency of output is fully configurable as well.

In summary, I've put together a schematic of data types and their relationships in Fig. (1.3). In this diagram there is some indication of the data type that I propose to store each piece information

in (in Go syntax), and the diagram as a whole should serve as a useful guide in developing the structure of configuration files for the stochadex.

## 1.4 Algorithm design

- In order to simulate Eq. (1.1), we need an interative algorithm which reapplies a user-specified function to the continually-updated history.
- multithreading should be used for state partitioning
- multiprocessing should be used for independent ensembles
- above formalism is so general that it can do anything - so while it shall serve as a useful guide and reference point, it would be good here to go through more of the specific desirable components we want to have access to in the software itself
- it might not always be convenient to have the windowed histories stored as S but some other varying quantity which can be used to construct S? take fractional brownian motion as an example of this! hence, need to provide more possible input histories into S
- want the timestep to have either exponentially-sampled lengths or fixed lengths in time
- formalism already isn't explicit about the choice of deterministic integrator in time
- but also want to be able to choose the stochastic integrator in continuous processes (Itô or Stratonovich?)
- enable correlated noise terms at the sample generator level
- configurable setup of simulations with just yamls + a single .go file defining the terms

Ideally, the stochadex sampler should be designed to try and maintain a balance between performance and flexibility of utilisation.

## 1.5 Implementation details

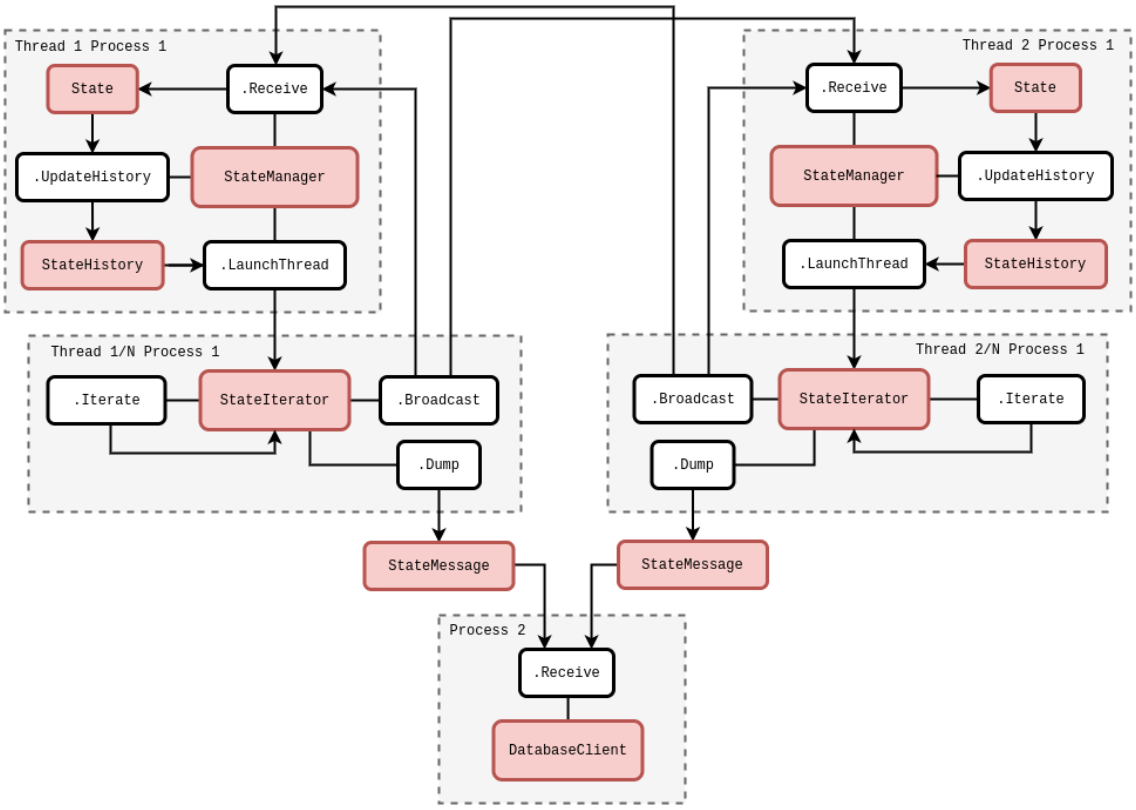


Figure 1.4: Write software design diagrams

2

# Simulating a financial market

**Concept.** The idea here is to use the Q-Hawkes processes and the Bouchaud work to come up with some interesting simulations of financial markets.

## 2.1 Introducing Q-Hawkes processes

3

# Quantum jumps on generic networks

**Concept.** The idea is to follow this sort of thing here to simulate the Lindblad equation over an arbitrary network of entangled states.

## 3.1 The Lindblad equation

# Part 2. How do we then learn/identify the answer to Part 1 from real-world data?

4

# Inferring dynamical 2D maps

**Concept.** The idea here is

## 4.1 Adapting the stochadex formalism

5

# Learning from ants on curved surfaces

**Concept.** The idea here is

## 5.1 Diffusive limits for ant interactions

6

# Hydrodynamic ensembles from input data

**Concept.** The idea here is

## 6.1 The Boltzmann/Navier-Stokes equations

7

# Generalised statistical inference tools

**Concept.** The idea here is to extend the stochadex with tools for very generalised statistical inference (ABC algorithms and the like) that will work in nearly every situation. Probably need to exploit the phase space analogy of the formalism.

## 7.1 Likelihood-free methods

# Part 3. How do we simulate a general set of control policies to interact with the answer to Part 1?

8

# Interacting with systems in general

**Concept.** The idea here is

## 8.1 Parameterising general interactions

9

# Managing a Rugby match

**Concept.** The idea here is

## 9.1 Introduction

Since the basic game engine will run using the stochadex sampler, the novelties in this project are all in the design of the rugby match model itself. And, in this instance, I'm not especially keen on spending a lot of time doing detailed data analysis to come up with the most realistic values for the parameters that are dreamed up here. Even though this would also be interesting.

One could do this data analysis, for instance, by scraping player-level performance data from one of the excellent websites that collect live commentary data such as rugbypass.com or espn.co.uk/rugby.

This game is primarily a way of testing out the interface of the stochadex for other users to build projects with. This should help to both iron out some of the kinks in the design, as well as prioritise adding some more convenience methods for event-based modelling into its code base.

## 9.2 Designing the event simulation engine

We need to begin by specifying an appropriate event space to live in when simulating a rugby match. It is important at this level that events are defined in quite broadly applicable terms, as it will define the state space available to our stochastic sampler and hence the simulated game will never be allowed to exist outside of it. So, in order to capture the fully detailed range of events that are possible in a real-world match, we will need to be a little imaginative in how we define certain gameplay elements when we move through the space.

The diagrams below sum up what should hopefully work as a decent initial approximation while providing a little context with specific examples of play action.

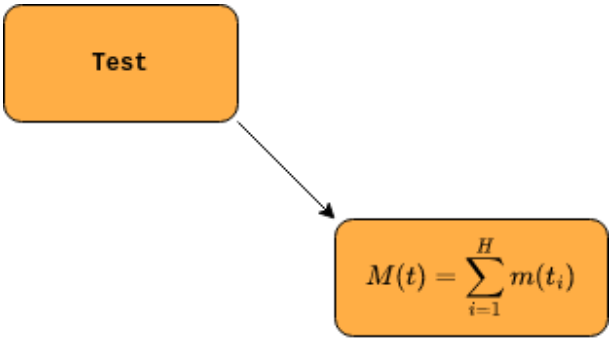


Figure 9.1: Simplified event graph of a rugby union match - replace with drawio.

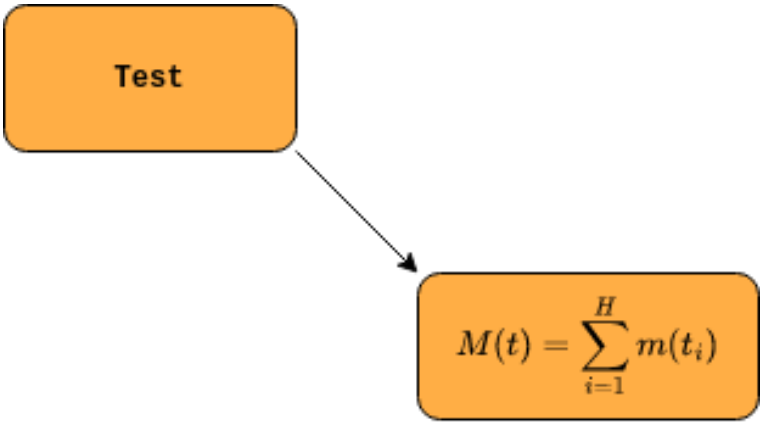


Figure 9.2: Optional model ideas - replace with drawio.

## 9.3 Linking to player attributes

## 9.4 Deciding on gameplay actions

## 9.5 Writing the game itself

10

# Influencing house prices

**Concept.** The idea here is

# Part 4. How do we then optimise the answer to Part 3 to achieve a specified control objective?

11

# Optimising actions for control objectives

**Concept.** The idea controlhere is

12

# Resource allocation for epidemics

**Concept.** The idea here is to limit the spread of some abstract epidemic through the correct time-dependent resource allocation.

13

# Quantum system control

**Concept.** The idea here is to follow stuff along these lines here.

14

# Other models

**Concept.** The idea here is

# Bibliography


[1] N. G. Van Kampen, *Stochastic processes in physics and chemistry*, vol. 1. Elsevier, 1992.

[2] H. Risken, *Fokker-planck equation*, in *The Fokker-Planck Equation*, pp. 63–95. Springer, 1996.

[3] L. Rogers and D. Williams, *Diffusions, Markov Processes and Martingales 2: Ito Calculus*, vol. 1, pp. xiv+480. 04, 2000. 10.1017/CBO9781107590120.

[4] L. Decreusefond et al., *Stochastic analysis of the fractional brownian motion*, *Potential analysis* **10** (1999) 177–214.

[5] D. T. Gillespie, *Exact stochastic simulation of coupled chemical reactions*, *The journal of physical chemistry* **81** (1977) 2340–2361.

[6] J. Neyman and E. L. Scott, *Statistical approach to problems of cosmology*, *Journal of the Royal Statistical Society: Series B (Methodological)* **20** (1958) 1–29.

[7] A. G. Hawkes, *Spectra of some self-exciting and mutually exciting point processes*, *Biometrika* **58** (1971) 83–90.

[8] H. A. Kramers, *Brownian motion in a field of force and the diffusion model of chemical reactions*, *Physica* **7** (1940) 284–304.

[9] J. Moyal, *Stochastic processes and statistical physics*, *Journal of the Royal Statistical Society. Series B (Methodological)* **11** (1949) 150–210.

[10] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.