Diffusing ideas

Software, noise and building mathematical toys

Robert J. Hardwick

January 16, 2023

Introduction

Diffusing ideas is a book of research exploration and software development which I have written over several years for the interest of mathematically-inclined programmers and computational scientists. It's the output of many interrelated projects 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 have tried to be as ambitious as possible with the level of technical sophistication in these models, I've also written most of the expressions in a way so as to make their intended computations as explicit as possible.

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.[1] 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 my 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] 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 the software and some examples, 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.

So the basic computational idea here is to 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 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.

What about the stochastic term? For example, if we wanted to consider a *Wiener process noise*, we can define W^i_{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 of the increments is

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

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_{t} to the Wiener process like so

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

[1] Or memory at least within some window.

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 'α-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 noise*, where $[B_H]_{\mathsf{t}}$ is a sample from a fractional Brownian motion process with Hurst exponent H. Following Ref. [4], we can simulate this process by first leveraging an extra dimension in the available state space within X to simulate a standard Wiener process $S^0_{\mathsf{t}+1}(X', \mathsf{t}) = W_{\mathsf{t}+1} - W_{\mathsf{t}}$ and then by inserting this into the following computation

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

we are able to reproduce the desired noise in the $i = 1$ dimension of our state space, i.e., $S^1_{\mathsf{t}+1}(X', \mathsf{t}) = [B_H]_{\mathsf{t}+1}$. 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 Γ 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

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

where $[N_\lambda]^i_{\mathsf{t}}$ is a sample from a Poisson process with rate λ. 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 $1/\lambda(\mathsf{t})$

[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).

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 τ 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, τ 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 quite a few possible extensions to the simple Poisson process which introduce additional stochastic processes. *Cox (doubly-stochastic) process noises*, for instance, are basically where we replace the time-dependent rate $\lambda(\mathsf{t})$ with independent samples from some probability distribution. In *compound Poisson process noises*, the count values $[N_\lambda]^i_\mathsf{t}$ are replaced by independent samples $[J_\lambda]^i_\mathsf{t}$ from another probability distribution, i.e.,

$$S^i_{\mathsf{t}+1}(X', \mathsf{t}) = [J_\lambda]^i_{\mathsf{t}+1} - [J_\lambda]^i_\mathsf{t} \,, \tag{1.13}$$

where we note that this can also be time-inhomogenous so that the rate is $\lambda(\mathsf{t})$.

Generalised probabilistic discrete state transitions would take the form

$$F^i_{\mathsf{t}+1}(X', \mathsf{t}) = T^i_{\mathsf{t}+1}(X') \,, \tag{1.14}$$

where $T^i_{\mathsf{t}+1}(X')$ is a generator of the next state to occupy. This generator uses the current state transition probabilities (which are generally conditional on X') at each new step.

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, before ending this subsection, it would be worthwhile going through some simple examples of this kind of process too.

Self-exciting process noises would generally take the form

$$S^i_{\mathsf{t}+1}(X', \mathsf{t}) = N^i_{\mathsf{t}+1}[\mathcal{I}^i_{\mathsf{t}+1}(N', \dots)] - N^i_\mathsf{t}[\mathcal{I}^i_\mathsf{t}(N'', \dots)] \,, \tag{1.15}$$

where the stochastic rate $\mathcal{I}^i_\mathsf{t}(N', \dots)$ now depends on the history of N' explicitly (amongst other potential inputs - see, e.g., Hawkes processes) for each of the state dimensions indexed by i.

1.2 Useful analysis concepts

Before progressing further, it will be helpful to discuss some useful concepts that should help us analyse the system later on. 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* of X. This can be written as

$$p_{\mathsf{t}+1}(X) = \int_{\Omega_\mathsf{t}} \mathrm{d}X' p_\mathsf{t}(X') p_{\mathsf{t}+1}(X|X', \mathsf{t}) \,, \tag{1.16}$$

where we are assuming the state space is continuous in each dimension. But what is Ω_{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.

Let us now define the probability density matrix P, with x-dependent elements $P^i_{\mathsf{t}}(x)$ which just means the probability density of $X^i_{\mathsf{t}} = x$. One can deduce that the latest values of this matrix are updated by a step in time according to

$$P^i_{\mathsf{t}+1}(x) = \int_{\Omega_{\mathsf{t}}} \mathrm{d}X' p_{\mathsf{t}}(X') p_{\mathsf{t}+1}(F^i_{\mathsf{t}+1}|X', \mathsf{t}) \delta(x - F^i_{\mathsf{t}+1}) \,, \tag{1.17}$$

where we have used Eq. (1.1) and a Dirac delta distribution δ.

Note that analogs of Eqs. (1.16) and (1.17) exist for discrete state spaces as well. In the case of Eq. (1.16), for instance, we would simply replace the integral with a sum and the schematic would look something like this

$$p_{\mathsf{t}+1}(X) = \sum_{\Omega_{\mathsf{t}}} p_{\mathsf{t}}(X') p_{\mathsf{t}+1}(X|X') \,, \tag{1.18}$$

where we note that the p's all now refer to *probability mass functions*.

Another useful concept comes from constructing what we shall call 'V states' using the time difference in X like so

$$V^i_{\mathsf{t}+1} = X^i_{\mathsf{t}+1} - X^i_{\mathsf{t}} \,. \tag{1.19}$$

Given Eq. (1.19), we can see that the following relation between X and V is also valid

$$X^i_{\mathsf{t}} = X^i_{\mathsf{s}} + \sum_{\mathsf{t}'=\mathsf{s}+1}^{\mathsf{t}} V^i_{\mathsf{t}'} \,, \tag{1.20}$$

where we have defined $\mathsf{s} < \mathsf{t}$. The resemblance of the tuple $(X_{\mathsf{t}}, V_{\mathsf{t}}, \mathsf{t})$ to a formal 'phase space' will be a useful observation to consider for many systems.

1.3 Summary of desirable features

- using the learnings from the previous sections looking at specific example processes
- 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

1.4 Software design choices

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

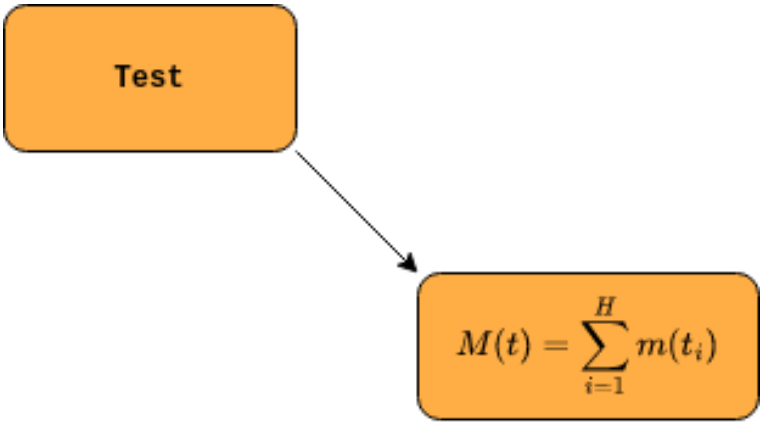

Figure 1.1: Write software design diagrams

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.

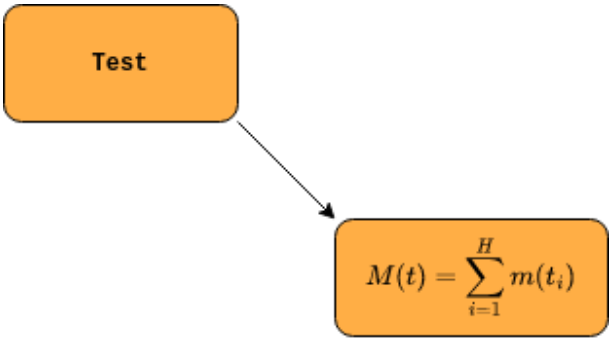

Figure 9.1: Simplified event graph of a rugby union match - replace with drawio.

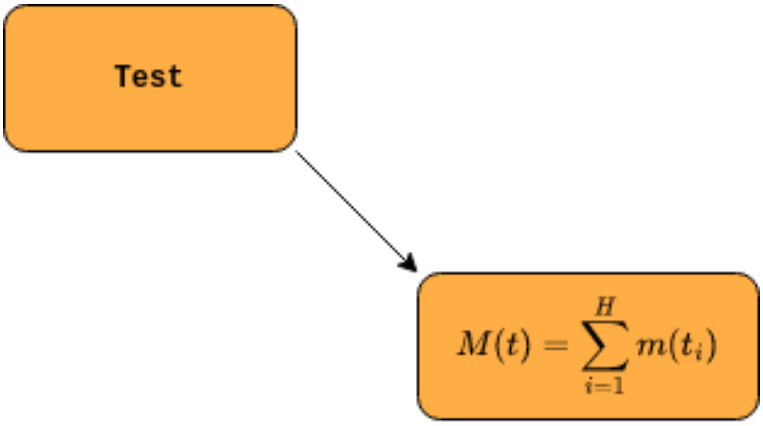

Figure 9.2: Optional model ideas - replace with drawio.

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.