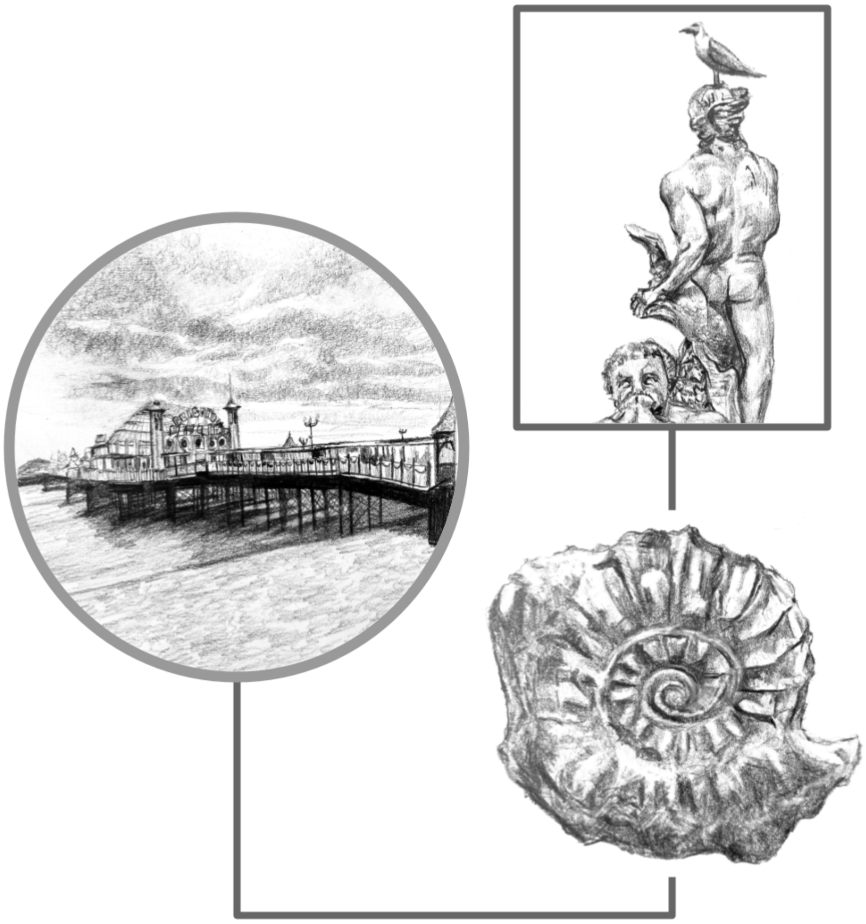

4

Empirical dynamical emulators

Concept. To extend the formalism that we developed in previous chapters to enable the empirical emulation of real-world data in a probabilistic way. This technique should enable a researcher to model complex dynamical trends in the data very well; at the cost of making the abstract interpretation of the model less immediately comprehensible than the statistical inference models in some proceeding chapters. As our generalised framework applies to a wide variety stochastic phenomena, our emulator will be applicable to a great breadth of data modeling problems as well. We will also explore some examples which illustrate how our empirical emulator should be applied in practice and then follow this up with how the code is designed and implemented as part of a new software package called 'learnadex'. For the mathematically-inclined, this chapter will take a detailed look at how our formalism can be extended to focus on probabilistic dynamical process emulation. For the programmers, the software described in this chapter lives in the public Git repository: https://github.com/umbralcalc/learnadex.

4.1 Probabilistic formalism

The key distinction between the methods that we will develop in this chapter and the ones in the proceeding chapters is in their utility when faced with the problem of attempting to model real-world data. In the proceeding chapter, we shall describe some powerful techniques that can be used most effectively when the researcher is aware of the family of models that generated the data. In the present chapter, we will go into the details of how a more 'empirical' approach can be derived for dynamical process modeling in a probabilistic framework which locally adapts the model to the data through time.

While we think that it's worth going into some mathematical detail to give a better sense of where our formalism comes from; we want to emphasise that the framework we discuss here is not especially new to the technical literature. We shall be using well-known techniques, such as Gaussian processes [1], and our overall framework is comparable to that of Empirical Dynamical Modeling (EDM) [2], or perhaps some classic nonparametric local regression techniques such as LOWESS/Savitzky-Golay filtering [3] as well. The novelties here, instead, lie more in the specifics of how we combine some of these ideas together when referencing the stochadex formalism, and how this manifests in designing more generally-applicable software for the user.

Before we are able to develop this empirical emulator, we need to return to the stochadex formalism that we introduced in the first chapter of this book. As we discussed at that point; this formalism is appropriate for sampling from nearly every stochastic phenomenon that one can think of. However, when trying robustly assess how far a model is from accurately describing a set of real-world data, trying to use only generated samples of the model process can be diffcult. Instead, in this section, we are going to extend this formalism to look at how probability theory can help with this data comparison problem in a systematic way.

So, how do we begin? In the first chapter, we defined the general stochastic process with the formula $X^i_{\mathsf{t}+1} = F^i_{\mathsf{t}+1}(X', z, \mathsf{t})$. This equation also has an implicit *master equation* associated to it that 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 t. This can be written as

$$P_{\mathsf{t}+1}(x) = \frac{1}{\mathsf{t}} \sum_{\mathsf{t}'=0}^{\mathsf{t}} \int_{\omega_{\mathsf{t}'}} \mathrm{d}x' P_{\mathsf{t}'}(x') P_{(\mathsf{t}+1)\mathsf{t}'}(x|x') \,, \tag{4.1}$$

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 t' was $x' = X_{\mathsf{t}'}$. This is a very general equation which should almost always apply to any continuous stochastic phenomenon we want to study in due course. To try and understand what this equation is saying we find it's helpful to think of an iterative relationship between probabilities; each of which is connected by their relative conditional probabilities. This kind of thinking is also illustrated in Fig. 4.1. Let's say we also wanted to program what this equation is saying as a function in Go. Using a Monte Carlo approximation for the integral domain, the code might look something like this.

```go
type StateVector  []float64

// returns a random draw of the possible state vectors at this timestep
func RandomPossibleStateVectors(timeStepNumber int) []StateVector {
    // return a slice of randomly-drawn possible state vectors
    // corresponding to the integral domain at this timestep
}

// returns the conditional probability of the state vector at this timestep
// given the value that the state vector had on a previous timestep
func StateVectorConditionalProbability(
    stateVector StateVector,
    timeStepNumber int,
    previousStateVector StateVector,
    previousTimeStepNumber int,
) float64 {
    // return the conditional probability value
}

// returns the probability of the state vector at this timestep
func StateVectorProbability(
    stateVector StateVector,
    timeStepNumber int,
) float64 {
    prob := 0.0
    // loop over all the possible previous timesteps
    for t := 0; t < timeStepNumber; t++ {
        // loop over the randomly-drawn possible state vectors
        // for this previous timestep
        possibleStateVectors := RandomPossibleStateVectors(t)
        for _, possibleStateVector := range possibleStateVectors {
            // note the recursion
            prob += StateVectorProbability(possibleStateVector, t)*
                StateVectorConditionalProbability(
                    stateVector,
                    timeStepNumber,
                    possibleStateVector,
                    t,
                )
        }
        // normalisation for the Monte Carlo integration
        prob /= float64(len(possibleStateVectors))
    }
    // timestep normalisation
    prob /= float64(timeStepNumber)
    return prob
}
```

The factor of $1/\mathsf{t}$ in Eq. (4.1) is a normalisation factor — this just normalises the sum of all probabilities to 1 given that there is a sum over t'. Note that, if the process is defined over continuous time, we would need to replace

$$\frac{1}{\mathsf{t}}\sum_{\mathsf{t}'=0}^{\mathsf{t}} \rightarrow \frac{1}{t(\mathsf{t})}\sum_{\mathsf{t}'=0}^{\mathsf{t}} \delta t(\mathsf{t}')\,. \tag{4.2}$$

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.

What if we wanted the joint distribution of both rows $P_{(\mathsf{t}+1)\mathsf{t}'}(x,x')$? One way to obtain this would be to extend Eq. (4.1) such that both matrix rows are marginalised over separately like so

$$P_{(\mathsf{t}+1)\mathsf{t}'}(x,x') = \\ \frac{1}{(\mathsf{t}'-1)\mathsf{t}}\sum_{\mathsf{t}''=0}^{\mathsf{t}}\sum_{\mathsf{t}'''=0}^{\mathsf{t}'-1}\int_{\omega_{\mathsf{t}''}}\mathrm{d}x''\int_{\omega_{\mathsf{t}'''}}\mathrm{d}x''' P_{\mathsf{t}''\mathsf{t}'''}(x'',x''')P_{(\mathsf{t}+1)\mathsf{t}''}(x|x'')P_{\mathsf{t}'\mathsf{t}'''}(x'|x''')\,. \tag{4.3}$$

Given Eqs. (4.1) and (4.3) it's also possible to work out what the conditional probabilities would look like using the simple relation

$$P_{(\mathsf{t}+1)\mathsf{t}'}(x|x') = \frac{P_{(\mathsf{t}+1)\mathsf{t}'}(x,x')}{P_{\mathsf{t}'}(x')}\,. \tag{4.4}$$

The implicit notation in Eq. (4.1) can hide some staggering complexity. To analyse the system in more detail, we can also do a kind of Kramers-Moyal expansion [4, 5] for each point in time to

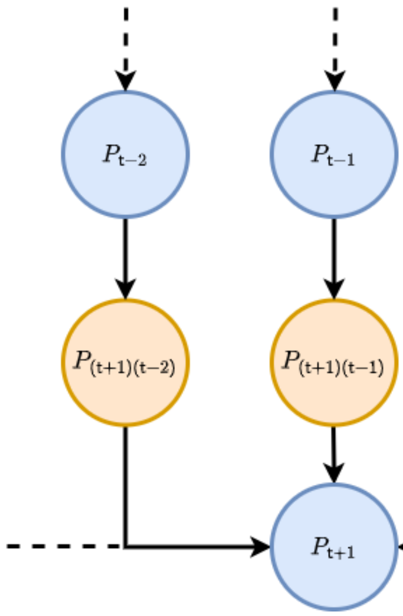

Figure 4.1: Graph representation of Eq. (4.1).

approximate the overall equation like this

$$
\begin{aligned}
P_{t+1}(x) = \frac{1}{t}\sum_{t'=0}^{t} P_{t'}(x) - \frac{1}{t}\sum_{t'=0}^{t}\sum_{i=1}^{d}\frac{\partial}{\partial x^i}\left[\alpha^i_{(t+1)t'}(x)P_{t'}(x)\right] \\
+ \frac{1}{2t}\sum_{t'=0}^{t}\sum_{i=1}^{d}\sum_{j=1}^{d}\frac{\partial}{\partial x^i}\frac{\partial}{\partial x^j}\left[\beta^{ij}_{(t+1)t'}(x)P_{t'}(x)\right] + \dots\,,
\end{aligned}
\tag{4.5}
$$

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

$$
\alpha^i_{(t+1)t'}(x) = \int_{\omega_{t'}} \mathrm{d}x'(x'-x)^i P_{(t+1)t'}(x'|x) \tag{4.6}
$$

$$
\beta^{ij}_{(t+1)t'}(x) = \int_{\omega_{t'}} \mathrm{d}x'(x'-x)^i(x'-x)^j P_{(t+1)t'}(x'|x)\,. \tag{4.7}
$$

So the matrix notation of Eq. (4.1) can indeed hide a very complicated calculation. Truncating the expansion at second-order, Eq. (4.5) tells us that there can be first and second derivatives contributing to the flow of probability to each element of the row $x = X_{t+1}$ 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' $J_{(t+1)t'}(x)$ from t' to $(t+1)$ which illustrates this through the following continuity relation

$$P_{\mathsf{t}+1}(x) - \frac{1}{\mathsf{t}}\sum_{\mathsf{t}'=0}^{\mathsf{t}} P_{\mathsf{t}'}(x) = \frac{1}{\mathsf{t}}\sum_{\mathsf{t}'=0}^{\mathsf{t}} \big[P_{\mathsf{t}+1}(x) - P_{\mathsf{t}'}(x)\big] = -\frac{1}{\mathsf{t}}\sum_{\mathsf{t}'=0}^{\mathsf{t}} J_{(\mathsf{t}+1)\mathsf{t}'}(x)\,. \tag{4.8}$$

By inspection of Eq. (4.5) we can therefore also deduce that

$$J^i_{(\mathsf{t}+1)\mathsf{t}'}(x) = \alpha^i_{(\mathsf{t}+1)\mathsf{t}'}(x)P_{\mathsf{t}'}(x) - \frac{1}{2}\sum_{j=1}^{d}\frac{\partial}{\partial x^j}\big[\beta^{ij}_{(\mathsf{t}+1)\mathsf{t}'}(x)P_{\mathsf{t}'}(x)\big] + \dots\,. \tag{4.9}$$

What would happen if we assumed that α could be defined with just an arbitrary time-dependent function? In this instance, we mean

$$\alpha^i_{(\mathsf{t}+1)\mathsf{t}'}(x) = f^i(\mathsf{t}') - x^i\,, \tag{4.10}$$

where $f(\mathsf{t}') = f'$ is an arbitrary vector-valued function of the timestep. Using this function, we might then try to approximate $\beta^{ij}_{(\mathsf{t}+1)\mathsf{t}'}(x) \simeq \beta^{ij}_{(\mathsf{t}+1)\mathsf{t}'}(f') = 2K(\theta,\mathsf{t}';f')$ with the matrix K, which depends on both the timestep and a set of hyperparameters θ. If we now also assume stationarity of $P_{\mathsf{t}'}(x) = P_{\mathsf{t}''}(x)$ for any t' and t'' such that

$$P_{\mathsf{t}+1}(x) = \frac{1}{\mathsf{t}}\sum_{\mathsf{t}'=0}^{\mathsf{t}} P_{\mathsf{t}'}(x)\,, \tag{4.11}$$

we can solve Eq. (4.5) to obtain the following stationary solution

$$P_{\mathsf{t}'}(x) = \mathsf{MultivariateNormalPDF}[x; f', K(\theta,\mathsf{t}';f')]\,. \tag{4.12}$$

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

It's possible to take this derivation a bit further by expanding Eq. (4.3) 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. (4.4), you can get something that looks like this conditional distribution

$$\begin{aligned} P_{(\mathsf{t}+1)\mathsf{t}'}(x|x') \propto \exp\bigg\{ &-\frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d}(x-f)^i\big[K^{-1}(\theta,\mathsf{t}+1,\mathsf{t}+1;f,f)\big]^{ij}(x-f)^j \\ &+\sum_{i=1}^{d}\sum_{j=1}^{d}(x-f)^i\big[K^{-1}(\theta,\mathsf{t}+1,\mathsf{t}';f,f')\big]^{ij}(x'-f')^j\bigg\}\,, \end{aligned} \tag{4.13}$$

where $f(\mathsf{t}+1) = f$ and $K(\theta,\mathsf{t}+1,\mathsf{t}';f,f')$ is some arbitrary covariance matrix that encodes how the correlation structure varies with the between compared states at two different timesteps and K^{-1} denotes taking its inverse. Note also that the inequality $(\mathsf{t}+1) > \mathsf{t}'$ must always hold.

Eq. (4.13) may look a bit familiar to some readers who like using Gaussian processes from the machine learning literature [1] — this version implies a *generative* model for a future x value (which we've illustrated in Fig 4.2), in contrast to the more standard equation used to *infer* values of f. These are two sides of the same coin though.

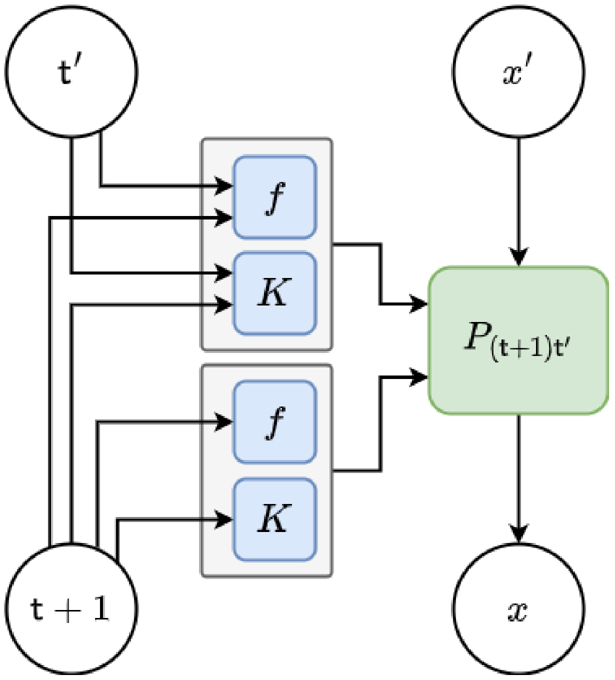

Figure 4.2: Graph representation of the generative process implied by Eq. (4.13).

Written in Go, the conditional probability in Eq. (4.13) could be computed by performing the summations over each element of the state vectors. We've given a rough idea of what this code would look like below.

```go
import "math"

type KernelParams []float64

// returns the f function value for a given timestep
func F(timestepNumber int) []float64 {
    // return value
}

// returns the inverse-K matrix for the input timesteps and
// params (theta in the text)
func inverseK(
    params KernelParams,
    timestepNumber int,
    previousTimeStepNumber int,
) [][]float64 {
    // return value
}

// returns the conditional probability of the state vector at this timestep
// given the value that the state vector had on a previous timestep
func StateVectorConditionalProbability(
    stateVector StateVector,
    timeStepNumber int,
    previousStateVector StateVector,
    previousTimeStepNumber int,
) float64 {
    // set the params from somewhere
    var kernelParams KernelParams

    // get the f-function values for both timesteps
    f := F(timeStepNumber)
    previousF := F(previousTimeStepNumber)

    // get the inverse-K matrices for the latest timestep
    // and also when both timesteps are input
    invK := inverseK(kernelParams, timeStepNumber, timeStepNumber)
    bothInvK := inverseK(
        kernelParams,
        timeStepNumber,
        previousTimeStepNumber,
    )

    // loop over terms to add to the log-probability
    logProbability := 0.0
    for i := range stateVector {
        for j := range stateVector {
            logProbability += -(stateVector[i] - f[i])*
                invK[i][j]*(stateVector[j] - f[j])/2.0
            logProbability += (stateVector[i] - f[i])*
                bothInvK[i][j]*(previousStateVector[j] - previousF[j])
        }
    }
    return math.Exp(logProbability)
}
```

In Eq. (4.13) we are implicitly approximating $\beta^{ij}_{(\mathsf{t}+1)\mathsf{t}'}(x,x') \simeq \beta^{ij}_{(\mathsf{t}+1)\mathsf{t}'}(f,f')$, which can also be thought of as a zeroth-order 'mean-field' expansion (here just a Taylor series in x and x') of the covariance matrix. If we were to continue this approximation up to second order, we would obtain

$$
\begin{aligned}
&K(\theta,\mathsf{t}+1,\mathsf{t}';x,x') \simeq \\
&K + \sum_{i=1}^{d}(x-f)^i\frac{\partial K}{(\partial x)^i} + \sum_{i=1}^{d}(x'-f')^i\frac{\partial K}{(\partial x')^i} + \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d}(x-f)^i\frac{\partial^2 K}{(\partial x)^i(\partial x)^j}(x-f)^j \\
&+ \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d}(x'-f')^i\frac{\partial^2 K}{(\partial x')^i(\partial x')^j}(x'-f')^j + \sum_{i=1}^{d}\sum_{j=1}^{d}(x-f)^i\frac{\partial^2 K}{(\partial x)^i(\partial x')^j}(x'-f')^j\,. \qquad (4.14)
\end{aligned}
$$

where all of the references to 'K' above are evaluated using $x = f$ and $x' = f'$ after any derivatives of the variable have been taken. This kind of derivative approximation of K is known in other contexts as the DALI method [6, 7]. In our case, this derivative approximation manifests as an expansion of K to include terms which extend the powers of x and x' all the way up to fourth-order when applied to Eq. (4.13). The benefit of this expansion in Eq. (4.13) as opposed to approximating the probability distribution through Gram-Charlier A series or Edgeworth series [8], is that in the latter cases it is much more difficult to ensure that the probability distribution is properly normalised.

By extending the procedure which obtains the joint distribution update in Eq. (4.3) and uses Eq. (4.4) to higher and higher orders of joint distribution $P_{(\mathsf{t}+1)\mathsf{t}'\mathsf{t}''}(x,x',x'')$,

$P_{(\mathsf{t}+1)\mathsf{t}'\mathsf{t}''\mathsf{t}'''}(x, x', x'', x''')$, etc. you can derive equations for conditional distributions like $P_{(\mathsf{t}+1)\mathsf{t}'\mathsf{t}''}(x; x', x'')$ which are able to describe higher-order correlations between state vectors. If we apply the DALI expansion of K to include higher-order dependencies on x'' and x''', we obtain this monster expression

$$
\begin{aligned}
&K(\theta, \mathsf{t}+1, \mathsf{t}', \mathsf{t}'', \mathsf{t}'''; x, x', x'', x''') \simeq \\
&K + \sum_{i=1}^{d} (x-f)^i \frac{\partial K}{(\partial x)^i} + \sum_{i=1}^{d} (x'-f')^i \frac{\partial K}{(\partial x')^i} + \sum_{i=1}^{d} (x''-f'')^i \frac{\partial K}{(\partial x'')^i} + \sum_{i=1}^{d} (x'''-f''')^i \frac{\partial K}{(\partial x''')^i} \\
&+ \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d} (x-f)^i \frac{\partial^2 K}{(\partial x)^i (\partial x)^j} (x-f)^j + \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d} (x'-f')^i \frac{\partial^2 K}{(\partial x')^i (\partial x')^j} (x'-f')^j \\
&+ \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d} (x''-f'')^i \frac{\partial^2 K}{(\partial x'')^i (\partial x'')^j} (x''-f'')^j + \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d} (x'''-f''')^i \frac{\partial^2 K}{(\partial x''')^i (\partial x''')^j} (x'''-f''')^j \\
&+ \sum_{i=1}^{d}\sum_{j=1}^{d} (x-f)^i \frac{\partial^2 K}{(\partial x)^i (\partial x')^j} (x'-f')^j + \sum_{i=1}^{d}\sum_{j=1}^{d} (x-f)^i \frac{\partial^2 K}{(\partial x)^i (\partial x'')^j} (x''-f'')^j \\
&+ \sum_{i=1}^{d}\sum_{j=1}^{d} (x-f)^i \frac{\partial^2 K}{(\partial x)^i (\partial x''')^j} (x'''-f''')^j + \sum_{i=1}^{d}\sum_{j=1}^{d} (x'-f')^i \frac{\partial^2 K}{(\partial x')^i (\partial x'')^j} (x''-f'')^j \\
&+ \sum_{i=1}^{d}\sum_{j=1}^{d} (x'-f')^i \frac{\partial^2 K}{(\partial x')^i (\partial x''')^j} (x'''-f''')^j + \sum_{i=1}^{d}\sum_{j=1}^{d} (x''-f'')^i \frac{\partial^2 K}{(\partial x'')^i (\partial x''')^j} (x'''-f''')^j \, ,
\end{aligned}
\tag{4.15}
$$

where, once again, all of the references to 'K' above are evaluated using $x = f$, $x' = f'$, $x'' = f''$ and $x''' = f'''$ after any derivatives of the variable have been taken. Note also in the expression above, we have extended the shorthand notation to include $f(\mathsf{t}'') = f''$ and $f(\mathsf{t}''') = f'''$, where these new timesteps must also fit into the general inequality $(\mathsf{t}+1) > \mathsf{t}' > \mathsf{t}'' > \mathsf{t}'''$.

We've already mentioned that the derivative expansion (or DALI method) used in Eqs. (4.14) and (4.15) is in a different context to where it was employed in Refs. [6, 7]. In our case, the probability distribution that we are trying to approximate contains out-of-time-order correlations in a similar fashion to those studied in quantum mechanics [9, 10] (though we are obviously only talking about a classical statistical analogue in this case). Hence, if K is a matrix which encodes second-order (pairwise) out-of-time order correlations, the derivatives of this matrix in Eq. (4.14) are higher-rank[1] objects which encode higher-order correlations between state vectors at different time points. Despite these expressions appearing as more complex than the second-order version, the methodology for using them in a generative model like Eq. (4.13), or for leveraging them to infer the $f(\mathsf{t})$ function (as is done in Gaussian process inference) can be much the same. We'll get into all how this could be done in the next section.

What other processes can be described by Eq. (4.1)? For Markovian phenomena, the equation no longer depends on timesteps older than the immediately previous one, hence the expression

[1]By 'higher-rank' here we specifically mean objects which have more indices than just a matrix $M^{ij} \rightarrow M^{ijk\cdots}$.

reduces to just

$$P_{\mathsf{t}+1}(x) = \int_{\omega_{\mathsf{t}}} \mathrm{d}x' P_{\mathsf{t}}(x') P_{(\mathsf{t}+1)\mathsf{t}}(x|x')\,. \tag{4.16}$$

It's also easy to show that Eq. (4.5) naturally simplifies into the more usually applied Kramers-Moyal expansion when considering a Markovian process — you just remove the sum over t' and the $1/\mathsf{t}$ normalisation factor.

Note that an analog of Eq. (4.1) 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

$$P_{\mathsf{t}+1}(x) = \frac{1}{\mathsf{t}} \sum_{\mathsf{t}'=0}^{\mathsf{t}} \sum_{\omega_{\mathsf{t}'}} P_{\mathsf{t}'}(x') P_{(\mathsf{t}+1)\mathsf{t}'}(x|x')\,, \tag{4.17}$$

where we note that the P's in the expression above all now refer to *probability mass functions*. Because the state space is now discrete, we cannot immediately intuit an approximative expansion from this expression.

As a brief mathematical aside; if you're really determined to use a similar approach to the one we derived above, you can rewrite it in terms of continuous-valued characteristic functions like this

$$\varphi_{\mathsf{t}+1}(s) = \frac{1}{\mathsf{t}} \sum_{\mathsf{t}'=0}^{\mathsf{t}} \int_{\ell_{\mathsf{t}'}} \mathrm{d}s' \mathcal{C}(s') \varphi_{\mathsf{t}'}(s') \varphi_{(\mathsf{t}+1)\mathsf{t}'}(s|s') \tag{4.18}$$

$$\mathcal{C}(s') = \frac{1}{(2\pi)^d} \sum_{\omega_{\mathsf{t}'}} e^{-i(s'\cdot x')}\,, \tag{4.19}$$

where $\ell_{\mathsf{t}'}$ defines all the continuous values that the vector s' can possibly have at time t'. In the expression above, $\mathcal{C}(s')$ acts like is a kind of comb[2] to map the continuous frequency domain of s' onto the discrete state space of x'. Note also that $\mathcal{C}(s')$ uses the imaginary number i and, to be visually tidier, the dot product notation $a\cdot b$ just means the sum of vector elements: $a\cdot b = \sum_{\forall k} a^k b^k$. In principle, one can perform an approximative expansion on Eq. (4.18) like we did for continuous state spaces. This isn't always the most practical way of analysing the system though.

We have one more important example to discuss and then we can cap off this more mathematical section. In the even-simpler case where x is just a vector of binary 'on' or 'off' states, Eq. (4.17) reduces to

$$P^i_{\mathsf{t}+1} = \frac{1}{\mathsf{t}} \sum_{\mathsf{t}'=0}^{\mathsf{t}} \sum_{j=1}^{d} P^j_{\mathsf{t}'} P^{ij}_{(\mathsf{t}+1)\mathsf{t}'} = \frac{1}{\mathsf{t}} \sum_{\mathsf{t}'=0}^{\mathsf{t}} \sum_{j=1}^{d} \big[P^j_{\mathsf{t}'} A^{ij}_{(\mathsf{t}+1)\mathsf{t}'} + (1 - P^j_{\mathsf{t}'}) B^{ij}_{(\mathsf{t}+1)\mathsf{t}'} \big]\,, \tag{4.20}$$

where $P^i_{\mathsf{t}'}$ now represents the probability that element $x^i = 1$ (is 'on') at time t'. The matrices A and B are defined as conditional probabilities where the previous state in time $P^j_{\mathsf{t}'}$ was either 'on' or 'off', respectively.

In this section, we looked into how the mathematical formalism used in the stochadex could be extended with probability theory. Now that we have more of a sense of how this formalism works, we are ready to move on to designing the algorithms for our emulator. So let's go!

[2] This is very similar to how a 'Dirac comb' works in signal processing [11].

4.2 Emulator algorithms

Plan of action for this section.

- Introduce the concept of retrieving state data through a measurement function $Y^i_{\mathsf{t}+1} = M^i_{\mathsf{t}+1}(X_{\mathsf{t}+1}, z, \mathsf{t})$.
- Parametric algorithm is the only one that will be available up to 4th order expansion. This should just be a straightforward optimisation.
- In order to stabilise the algorithm to sample size fluctuations, a stochastic gradient descent method to optimize would work well with the $\mathrm{E}[\ln \mathcal{L}]$ so the expectation values like $\mathrm{E}[(x - f)(x' - f')(x'' - f'')]$ end up being used directly in the optimisation!

The parametric algorithm simply needs the K-functions to be specified and then an optimiser can run over the data values to simultaneously attempt to obtain θ and f values.

Helpful to write the basic structure of algorithm out in Go.

4.3 Software design

Bibliography

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

[2] G. Sugihara and R. M. May, *Nonlinear forecasting as a way of distinguishing chaos from measurement error in time series*, *Nature* **344** (1990) 734–741.

[3] A. Savitzky and M. J. Golay, *Smoothing and differentiation of data by simplified least squares procedures.*, *Analytical chemistry* **36** (1964) 1627–1639.

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

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

[6] E. Sellentin, M. Quartin and L. Amendola, *Breaking the spell of gaussianity: forecasting with higher order fisher matrices*, *Monthly Notices of the Royal Astronomical Society* **441** (2014) 1831–1840.

[7] "DALI: Non-Gaussian Likelihood Approximations." `https://lnasellentin.github.io/DALI/`.

[8] J. E. Kolassa, *Series approximation methods in statistics*, vol. 88. Springer Science & Business Media, 2006.

[9] K. Hashimoto, K. Murata and R. Yoshii, *Out-of-time-order correlators in quantum mechanics*, *Journal of High Energy Physics* **2017** (2017) 1–31.

[10] I. García-Mata, R. A. Jalabert and D. A. Wisniacki, *Out-of-time-order correlations and quantum chaos*, *Scholarpedia* **18** (2023) 55237.

[11] D. Brandwood, *Fourier transforms in radar and signal processing.* Artech House, 2012.