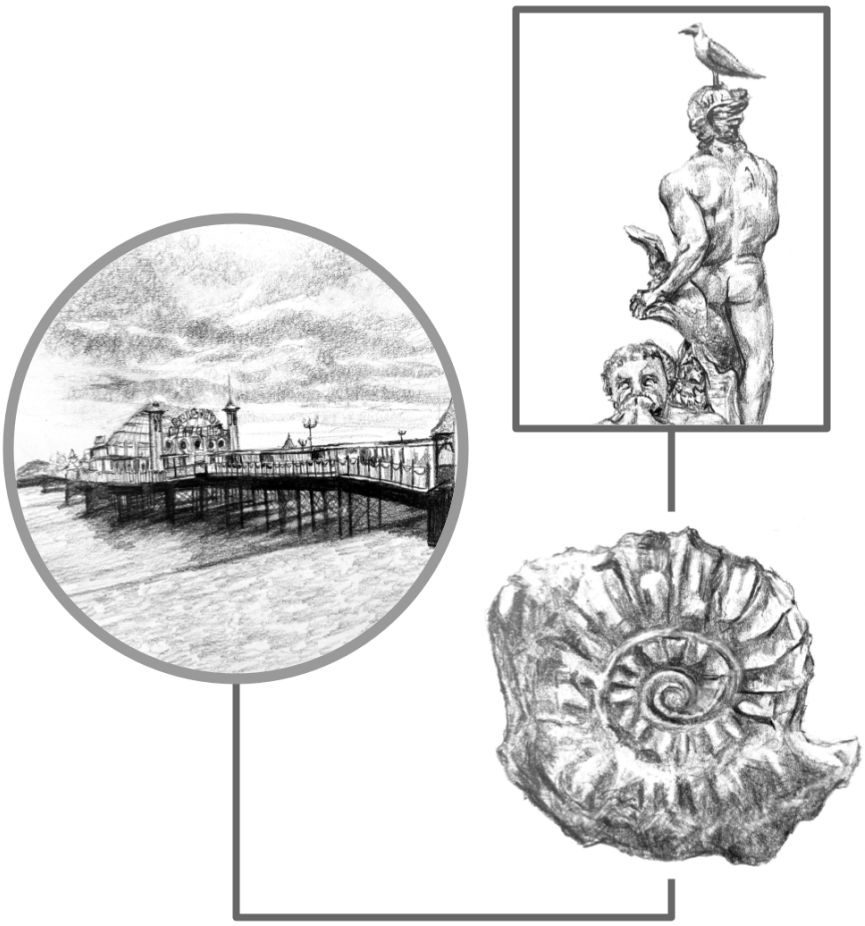

4

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. Our overall framework draws on influences from Empirical Dynamical Modeling (EDM) [1], some classic nonparametric local regression techniques — such as LOWESS/Savitzky-Golay filtering [2] — and also Gaussian processes [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 $X^i_{\mathtt{t}+1} = F^i_{\mathtt{t}+1}(X', z, \mathtt{t})$. This equation also has an implicit *master equation* associated to it that fully describes the time evolution of the *probability density function* $P_{\mathtt{t}+1}(x)$ of the next matrix row being $x = X_{\mathtt{t}+1}$. This can be written as

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

where at the moment we are assuming the state space is continuous in each dimension and $P_{(\mathtt{t}+1)\mathtt{t}'}(x|x')$ is the conditional probability that the matrix row at time $(\mathtt{t}+1)$ will be $x = X_{\mathtt{t}+1}$ given that the row at time $\mathtt{t}'$ was $x' = X_{\mathtt{t}'}$. 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.

```
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

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

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.

If we wanted to compute the mean of the distribution $M_{\mathsf{t}+1}$ in Eq. (4.1), it would be straightforward to just multiply both sides of the expression by x and integrate over $\mathrm{d}x$ in the ω_{t} domain. However, there is another similar expression for the mean that we can derive under certain conditions which will be valuable to us when developing the empirical emulator. If the probability distribution is *stationary* — meaning that $P_{\mathsf{t}'}(x) = P_{\mathsf{t}''}(x)$ for all t' and t'' — it's possible to derive[1]

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

[1] To see that this is true, first note that the joint distribution $P_{(\mathsf{t}+1)\mathsf{t}'}(x, x') = P_{(\mathsf{t}+1)\mathsf{t}'}(x|x')P_{\mathsf{t}'}(x')$. Secondly, note that joint distributions always allow variable swaps trivially like this $P_{(\mathsf{t}+1)\mathsf{t}'}(x, x') = P_{\mathsf{t}'(\mathsf{t}+1)}(x', x)$. Then, lastly, note that stationarity of $P_{\mathsf{t}+1}(x) = P_{\mathsf{t}'}(x)$ means

$$\int \mathrm{d}x \int \mathrm{d}x'\, x\, P_{(\mathsf{t}+1)\mathsf{t}'}(x, x') = \int \mathrm{d}x \int \mathrm{d}x'\, x\, P_{\mathsf{t}'(\mathsf{t}+1)}(x, x') = \int \mathrm{d}x \int \mathrm{d}x'\, x\, P_{(\mathsf{t}+1)\mathsf{t}'}(x', x)\,,$$

where we've used the trivial variable swap to get to the last equality, and the domain references ω in the integrals are implicitly defined.

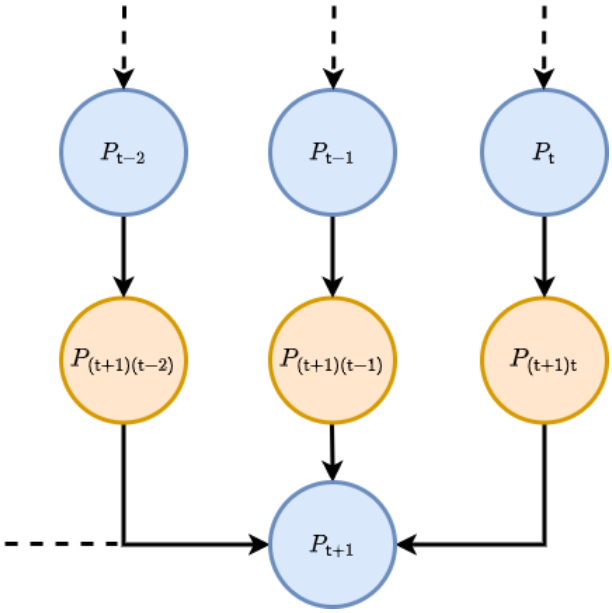

Figure 4.1: Graph representation of Eq. (4.1).

The standard covariance matrix elements can also be computed in a similar fashion

$$\begin{aligned} C^{ij}_{\mathsf{t}+1} &= \int_{\omega_{\mathsf{t}+1}} \mathrm{d}x\,(x - M_{\mathsf{t}+1})^i (x - M_{\mathsf{t}+1})^j P_{\mathsf{t}+1}(x) \\ &= \frac{1}{\mathsf{t}} \sum_{\mathsf{t}'=0}^{\mathsf{t}} \int_{\omega_{\mathsf{t}'}} \mathrm{d}x' \int_{\omega_{\mathsf{t}+1}} \mathrm{d}x\,(x' - M_{\mathsf{t}+1})^i (x' - M_{\mathsf{t}+1})^j P_{\mathsf{t}'}(x') P_{(\mathsf{t}+1)\mathsf{t}'}(x|x')\,. \end{aligned} \tag{4.4}$$

In order to study higher-order out-of-time-order correlations, we can also consider using the statistical moments computed from joint distributions like these

$$P_{(\mathsf{t}+1)\mathsf{t}'}(x, x') = P_{(\mathsf{t}+1)\mathsf{t}'}(x|x') P_{\mathsf{t}'}(x') \tag{4.5}$$

$$P_{(\mathsf{t}+1)\mathsf{t}'\mathsf{t}''}(x, x', x'') = P_{(\mathsf{t}+1)\mathsf{t}'\mathsf{t}''}(x|x', x'') P_{\mathsf{t}'\mathsf{t}''}(x'|x'') P_{\mathsf{t}''}(x'')\,. \tag{4.6}$$

For example, Eq. (4.5) would apply if we wanted to retrieve the out-of-time-order pairwise correlations between x at timestep $\mathsf{t}+1$ and x' at timestep t'. Using this pairwise relationship we can also derive an equation for the out-of-time-order covariance matrix elements

$$C^{ij}_{(\mathsf{t}+1)\mathsf{t}'} = \frac{1}{\mathsf{t}} \sum_{\mathsf{t}'=0}^{\mathsf{t}} \int_{\omega_{\mathsf{t}'}} \mathrm{d}x' \int_{\omega_{\mathsf{t}+1}} \mathrm{d}x\,(x - M_{\mathsf{t}+1})^i (x' - M_{\mathsf{t}'})^j P_{(\mathsf{t}+1)\mathsf{t}'}(x, x')\,. \tag{4.7}$$

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

reduces to just

$$P_{\mathsf{t}+1}(x) = \int_{\omega_\mathsf{t}} \mathrm{d}x' P_\mathsf{t}(x') P_{(\mathsf{t}+1)\mathsf{t}}(x|x') \,. \tag{4.8}$$

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

$$P_{\mathsf{t}+1}(x) = \frac{1}{\mathsf{t}} \sum_{\mathsf{t}'=0}^{\mathsf{t}} \sum_{\omega_{\mathsf{t}'}} P_{\mathsf{t}'}(x') P_{(\mathsf{t}+1)\mathsf{t}'}(x|x') \,, \tag{4.9}$$

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, Eq. (4.9) reduces to

$$P^i_{\mathsf{t}+1} = \frac{1}{\mathsf{t}} \sum_{\mathsf{t}'=0}^{\mathsf{t}} \sum_{j=1}^{d} P^j_{\mathsf{t}'} P^{ij}_{(\mathsf{t}+1)\mathsf{t}'} = \frac{1}{\mathsf{t}} \sum_{\mathsf{t}'=0}^{\mathsf{t}} \sum_{j=1}^{d} \left[P^j_{\mathsf{t}'} A^{ij}_{(\mathsf{t}+1)\mathsf{t}'} + (1 - P^j_{\mathsf{t}'}) B^{ij}_{(\mathsf{t}+1)\mathsf{t}'} \right] , \tag{4.10}$$

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 algorithm for our emulator. So let's go!

4.2 Generalised emulator algorithm

The empirical emulator algorithm that we're going to describe in this section will depend on the stationarity of $P_{\mathsf{t}+1}(x) = P_{\mathsf{t}'}(x)$ such that, e.g., Eq. (4.3) is applicable. But before we think about the various kinds of conditional probability we could use, we need to connect values of x in our emulator to the data from which it will be learning. If the mean is a sufficient statistic for the distribution which describes the data, a choice of, e.g., Exponential, Poisson or Binomial distribution could be used where the mean is estimated directly from the time series using Eq. (4.3), given a conditional probability $P_{(\mathsf{t}+1)\mathsf{t}'}(x|x')$. Extending this idea further to include distributions which also require a variance to be known, e.g., the Normal, Gamma or Negative Binomial distributions could be used where the variance (or covariance between multivariate implementations) could be estimated using Eq. (4.4). These are just a few simple examples of distributions that can link the estimated statistics from Eqs. (4.3) and (4.4) to a time series dataset. However, the algorithmic framework is very general to whatever choice of 'data linking' distribution that you want.

We should probably make what we've just said a little more mathematically concrete. Let's define a data vector y at time $\mathsf{t}+1$, which shares the same length as x, and represents a specific observation of x at this point in a real dataset. At this point in time, we can define a distribution $P_{\mathsf{t}+1}(y; M_{\mathsf{t}+1}, C_{\mathsf{t}+1}, \dots)$ which represents the likelihood of y given the estimated statistics from Eqs. (4.3) and (4.4) (and maybe higher-orders). Note that in order to do this, we need to identify the x' and t' values that are used to estimate, e.g., $M_{\mathsf{t}+1}$ with the past data values which are observed in the dataset time series itself. Now that we have this likelihood, we can immediately

define an overall objective function (an overall log-likelihood) that we might seek to optimise over for a given dataset

$$\ln \mathcal{L} = \sum_{\forall(y',\mathsf{t}')} \ln P_{\mathsf{t}'}(y'; M_{\mathsf{t}'}, C_{\mathsf{t}'}, \dots)\,, \tag{4.11}$$

where $\forall(y', \mathsf{t}')$ symbolises all tuples of datapoint and timestep, respectively.

In order to specify what $P_{(\mathsf{t}+1)\mathsf{t}'}(x|x')$ is, it's quite commmon to define a number of new hyperparameters. For example; one clear, and generally applicable, option for the conditional probability is that of a Gaussian process

$$P_{(\mathsf{t}+1)\mathsf{t}'}(x|x') = \frac{\exp\left\{-\frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d}[x - f(\mathsf{t}+1)]^i \left[K^{-1}(\theta, \mathsf{t}+1, \mathsf{t}')\right]^{ij}[x' - f(\mathsf{t}')]^j\right\}}{|2\pi K(\theta, \mathsf{t}+1, \mathsf{t}')|^{\frac{1}{2}}}\,, \tag{4.12}$$

where K is a covariance matrix; f is a function in time and θ represents a set of hyperparameters which are both typically learned by optimisation of the corresponding likelihood function of the data; and $|A|$ denotes computing the determinant of matrix A. By optimising Eq. (4.11) with respect to all of these hyperparameters which are defined with $P_{(\mathsf{t}+1)\mathsf{t}'}(x|x')$, we are effectively adopting what is known as an 'empirical Bayes' approach [3, 4].

Talk about other kernels?

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

4.3 Software design

Diagrams would help here.

Bibliography

[1] 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.

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

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

[4] D. J. MacKay, *Information theory, inference and learning algorithms*. Cambridge university press, 2003.