Modeling Temporal Data as Continuous Functions with Stochastic Process Diffusion

Marin Biloš [1 2] Kashif Rasul [2] Anderson Schneider [2] Yuriy Nevmyvaka [2] Stephan Günnemann [1]

Abstract

Temporal data such as time series can be viewed as discretized measurements of the underlying function. To build a generative model for such data we have to model the stochastic process that governs it. We propose a solution by defining the denoising diffusion model in the function space which also allows us to naturally handle irregularly-sampled observations. The forward process gradually adds noise to functions, preserving their continuity, while the learned reverse process removes the noise and returns functions as new samples. To this end, we define suitable noise sources and introduce novel denoising and score-matching models. We show how our method can be used for multivariate probabilistic forecasting and imputation, and how our model can be interpreted as a neural process.

1. Introduction

Time series data is collected from measurements of some real-world system that evolves via some complex unknown dynamics. The sampling rate is often arbitrary and non-uniform, producing irregularly-sampled time series. Therefore, we can make an assumption that time series follows some underlying continuous function; consider, e.g., the temperature or load of a system over time. Although values are observed as separate events, we know that temperature always exists and its evolution over time is smooth, not jittery. This kind of data can be found in many domains, from medical, industrial to financial applications.

Previously, different approaches for modeling irregular data have been proposed, including neural ordinary and stochastic differential equations (Chen et al., 2018; Li et al., 2020), neural processes (Garnelo et al., 2018), normalizing flows (Deng et al., 2020), etc. As it turns out, capturing the true generative process proves difficult, especially with the inherent stochasticity of the data.

[1]Technical University of Munich, Germany [2]Machine Learning Research, Morgan Stanley, United States. Correspondence to: Marin Biloš <marin.bilos@tum.de>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

Recently, denoising diffusion models have shown great promise in modeling very complicated data distributions such as those in the image domain (Ho et al., 2020; Song et al., 2021). The approach consists of first gradually adding the noise to data, until it becomes pure noise, corresponding to some base distribution. At the same time, the model is trained to reverse this process. To generate a new data point, we start with an initial noisy value sampled from the base distribution; then the model gradually denoises it to reach a sample from the learned data distribution. We define this more rigorously in Section 2.

In this work, we expand on this general framework to define the diffusion for data measured in continuous time by treating it as a discretization of some continuous function. That is, instead of adding noise to each data point independently, we add the noise to the whole function while preserving its continuity. In Section 3, we show that this can be done by using stochastic processes as noise generators. We additionally show that the final noisy function will also correspond to a sample from a known stochastic process. Next, we specify the transition probabilities in the forward noising process, the evidence bound on the likelihood used in the training, and the new sampling procedure, for both the fixed-step and SDE-based diffusion approaches.

Figure 1 shows an illustration of our approach. Data is observed as a set of (irregularly-sampled) points that correspond to some underlying function. By adding noise to this function we reach the prior stochastic process. At the same time, the model can reverse this process, allowing us to generate new function samples.

In Section 4 we describe different use cases that we tackle with our model while highlighting the benefits over previous approaches. For instance, we use conditioning, to output the distribution over future values, i.e., for multivariate probabilistic forecasting. Since we define the distribution over functions we can also view our model as a neural process (Garnelo et al., 2018), allowing us to estimate missing points from the observed. In Section 5 we empirically show that our model outperforms the baselines on all tasks.

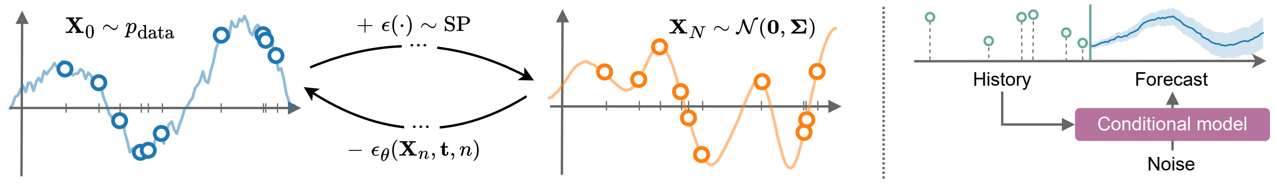
$\mathbf{X}_0 \sim p_{\text{data}}$
$+\ \epsilon(\cdot) \sim \text{SP}$
$-\ \epsilon_\theta(\mathbf{X}_n, \mathbf{t}, n)$
$\mathbf{X}_N \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma})$
History
Forecast
Conditional model
Noise

Figure 1. (Left) We add noise from a stochastic process (SP) to the *whole* time series at once. The model $\boldsymbol{\epsilon}_\theta$ learns to reverse this process. (Right) We can use this approach to, e.g., forecast with uncertainty.

2. Background

Generally, given training data $\{\boldsymbol{x}_i\}$, with $\boldsymbol{x}_i \in \mathbb{R}^d$, the goal of generative modeling is to learn the probability density function $p(\boldsymbol{x})$ and be able to generate new samples from this learned distribution. Diffusion models achieve both of these goals by learning to reverse some fixed process that adds noise to the data. In the following, we present a brief overview of the two ways to define diffusion; in Section 2.1 the noise is added across N increasing scales (Ho et al., 2020), which is then taken to the limit in Section 2.2 using a stochastic differential equation (SDE) (Song et al., 2021).

2.1. Fixed-step diffusion

Sohl-Dickstein et al. (2015); Ho et al. (2020) propose the denoising diffusion probabilistic model (DDPM) which gradually adds *fixed* Gaussian noise to the observed data point $\boldsymbol{x}_0$ via known scales β_n to define a sequence of progressively noisier values $\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N$. The final noisy output $\boldsymbol{x}_N \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ carries no information about the original data point. The sequence of positive noise (variance) scales $\beta_1, \dots, \beta_N$ has to be increasing such that the first noisy output $\boldsymbol{x}_1$ is close to the original data $\boldsymbol{x}_0$, and the final value $\boldsymbol{x}_N$ is pure noise. The goal is then to learn to reverse this process.

As diffusion forms a Markov chain, the transition between any two consecutive points is defined with a conditional probability $q(\boldsymbol{x}_n|\boldsymbol{x}_{n-1}) = \mathcal{N}(\sqrt{1-\beta_n}\boldsymbol{x}_{n-1}, \beta_n \boldsymbol{I})$. Since the transition kernel is Gaussian, the value at any step n can be sampled directly from $\boldsymbol{x}_0$. Let $\alpha_n = 1 - \beta_n$ and $\bar{\alpha}_n = \prod_{k=1}^n \alpha_k$, then we can write:

$$q(\boldsymbol{x}_n|\boldsymbol{x}_0) = \mathcal{N}(\sqrt{\bar{\alpha}_n}\boldsymbol{x}_0, (1-\bar{\alpha}_n)\boldsymbol{I}). \tag{1}$$

Further, the probability of any intermediate value $\boldsymbol{x}_{n-1}$ given its successor $\boldsymbol{x}_n$ and initial $\boldsymbol{x}_0$ is

$$q(\boldsymbol{x}_{n-1}|\boldsymbol{x}_n, \boldsymbol{x}_0) = \mathcal{N}(\tilde{\boldsymbol{\mu}}_n, \tilde{\beta}_n \boldsymbol{I}), \tag{2}$$

$$\text{where:} \quad \tilde{\boldsymbol{\mu}}_n = \frac{\sqrt{\bar{\alpha}_{n-1}}\beta_n}{1-\bar{\alpha}_n}\boldsymbol{x}_0 + \frac{\sqrt{\alpha_n}(1-\bar{\alpha}_{n-1})}{1-\bar{\alpha}_n}\boldsymbol{x}_n, \qquad \tilde{\beta}_n = \frac{1-\bar{\alpha}_{n-1}}{1-\bar{\alpha}_n}\beta_n.$$

The generative model learns the reverse process. To this end, Sohl-Dickstein et al. (2015) set $p(\boldsymbol{x}_{n-1}|\boldsymbol{x}_n) = \mathcal{N}(\boldsymbol{\mu}_\theta(\boldsymbol{x}_n, n), \beta_n \boldsymbol{I})$, and parameterized $\boldsymbol{\mu}_\theta$ with a neural network. The training objective is to maximize the evidence lower bound, $\log p(\boldsymbol{x}_0) \geq$

$$\begin{aligned}&\mathbb{E}_q[\log p(\boldsymbol{x}_0|\boldsymbol{x}_1) - D_{\mathrm{KL}}(q(\boldsymbol{x}_N|\boldsymbol{x}_0)\|p(\boldsymbol{x}_N)) \\ &\quad - \textstyle\sum_{n>1} D_{\mathrm{KL}}(q(\boldsymbol{x}_{n-1}|\boldsymbol{x}_n, \boldsymbol{x}_0)\|p(\boldsymbol{x}_{n-1}|\boldsymbol{x}_n))].\end{aligned} \tag{3}$$

In practice, however, the approach by Ho et al. (2020) is to reparameterize $\boldsymbol{\mu}_\theta$ and predict the noise $\boldsymbol{\epsilon}$ that was added to $\boldsymbol{x}_0$, using a neural network $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_n, n)$, and minimize the simplified loss function:

$$\mathcal{L} = \mathbb{E}_{\boldsymbol{\epsilon}, n}\left[\|\boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_n}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_n}\boldsymbol{\epsilon}, n) - \boldsymbol{\epsilon}\|_2^2\right], \tag{4}$$

where the expectation is over $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and $n \sim \mathcal{U}(\{0, \dots, N\})$. To generate new data, the first step is to sample a point from the final distribution $\boldsymbol{x}_N \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and then iteratively denoise it using the above model ($\boldsymbol{x}_N \mapsto \boldsymbol{x}_{N-1} \mapsto \cdots \mapsto \boldsymbol{x}_0$) to get a sample from the data distribution. To summarize, the forward process adds the noise $\boldsymbol{\epsilon}$ to the input $\boldsymbol{x}_0$, at different scales, to produce $\boldsymbol{x}_n$. The learned model inverts this, i.e., predicts $\boldsymbol{\epsilon}$ from $\boldsymbol{x}_n$.

2.2. Score-based SDE diffusion

Instead of taking a finite number of diffusion steps as in Section 2.1, Song et al. (2021) introduce a continuous diffusion of vector valued data, $\boldsymbol{x}_0 \mapsto \boldsymbol{x}_s$ where $s \in [0, S]$ is now a continuous variable. The forward process can be elegantly defined with an SDE:

$$\mathrm{d}\boldsymbol{x}_s = f(\boldsymbol{x}_s, s)\mathrm{d}s + g(s)\mathrm{d}W_s, \tag{5}$$

where W is a standard Wiener process. The variable s is the continuous analogue of the discrete steps implying that the input gets noisier during the SDE evolution. The final value $\boldsymbol{x}_S \sim p(\boldsymbol{x}_S)$ will follow some predefined distribution, as in Section 2.1. For the forward SDE in Equation 5 there exist a corresponding reverse SDE (Anderson, 1982):

$$\mathrm{d}\boldsymbol{x}_s = [f(\boldsymbol{x}_s, s) - g(s)^2 \nabla_{\boldsymbol{x}_s} \log p(\boldsymbol{x}_s)]\mathrm{d}s + g(s)\mathrm{d}W_s,$$

where $\nabla_{\boldsymbol{x}_s} \log p(\boldsymbol{x}_s)$ is the score function. Solving the above SDE from S to 0, given initial condition $\boldsymbol{x}_S \sim p(\boldsymbol{x}_S)$, returns a sample from the data distribution. The generative model's goal is to learn the score function via a neural network $\psi_{\boldsymbol{\theta}}(\boldsymbol{x}_s, s)$, by minimizing:

$$\mathcal{L} = \mathbb{E}_{\boldsymbol{x}_s, s}\left[\|\psi_{\boldsymbol{\theta}}(\boldsymbol{x}_s, s) - \nabla_{\boldsymbol{x}_s} \log p(\boldsymbol{x}_s)\|_2^2\right], \quad (6)$$

with $\boldsymbol{x}_s \sim \mathrm{SDE}(\boldsymbol{x}_0)$ and $s \sim \mathcal{U}(0, S)$. Song et al. (2021) define the continuous equivalent to DDPM forward process as the following SDE:

$$\mathrm{d}\boldsymbol{x}_s = -\frac{1}{2}\beta(s)\boldsymbol{x}_s \mathrm{d}s + \sqrt{\beta(s)}\mathrm{d}W_s, \quad (7)$$

where $\beta(s)$ and S are chosen in such a way that ensures the final noise distribution is unit normal, $\boldsymbol{x}_S \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. Given this specific parameterization, one can easily derive the transition probability $q(\boldsymbol{x}_s|\boldsymbol{x}_0)$ and calculate the exact score in closed-form (see Section 3.3 and Appendix A.3).

2.3. Extensions

Generative modeling with diffusion recently gained traction as it provides good sampling quality on image generation (Dhariwal & Nichol, 2021; Ramesh et al., 2022; Rombach et al., 2021) and became the state-of-the-art method replacing GANs (Goodfellow et al., 2020). The modeling power translates to other tasks as well, so it has been used in, e.g., modeling waveforms (Kong et al., 2021) and time series forecasting (Rasul et al., 2021a), but also generating discrete data such as text (Austin et al., 2021) and molecules (Anand & Achim, 2022; Lee & Kim, 2022). In this work we tackle a different task—generating continuous functions. Many of the advances over the original diffusion focused on improving the sampling speed (Chung et al., 2022; Jolicoeur-Martineau et al., 2021; Lyu et al., 2022), while others implement the noise scheduling for better modeling capacity (Nichol & Dhariwal, 2021b; Kingma et al., 2021). This area of research is orthogonal to our proposed method as we can easily implement any of the techniques that improve general diffusion, to make our method perform faster or have better sampling quality.

3. Diffusion for time series data

In contrast to the previous section which deals with data points that are represented by vectors, we are interested in generative modeling for time series data. We represent the data as a time-indexed sequence of points observed across M timestamps: $\boldsymbol{X} = (\boldsymbol{x}(t_0), \dots, \boldsymbol{x}(t_{M-1}))$, $t_i \in \boldsymbol{t} \subset [0, T]$. The observations can be equally spaced but this formulation encompasses irregularly-sampled data as well. We assume that each observed time series comes from its corresponding underlying continuous function $\boldsymbol{x}(\cdot)$.

Our approach can be viewed as modeling the distribution "$p(\boldsymbol{x}(\cdot))$" over functions instead of vectors, which amounts to learning the stochastic process. We review stochastic processes in more detail in Section 4.2. To preserve continuity, we cannot apply the ideas from Section 2 directly, unless we assume measurements are independent of each other. One issue of adding independent noise in the diffusion arises because it produces discontinuous samples.

3.1. Stochastic processes as noise sources for diffusion

Instead of defining the diffusion by adding some scaled noise vector $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ to a data vector $\boldsymbol{x}$, we add a noise *function* (stochastic process) $\boldsymbol{\epsilon}(\cdot)$ to the underlying data function $\boldsymbol{x}(\cdot)$. The only restriction on $\boldsymbol{\epsilon}(\cdot)$ is that it has to be continuous so that the output remains continuous as well, which clearly rules out stochastic processes where time is indexed by a *finite* set, e.g., $\boldsymbol{\epsilon}(\boldsymbol{t}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. However, using a normal distribution proved to be very convenient in Section 2 as it allowed for closed-form formulations of various terms, especially the loss. This is due to the nice properties that Gaussian random variables have.

Therefore, our goal is to define $\boldsymbol{\epsilon}(\cdot)$ which will satisfy the continuity property while giving us tractable training and sampling. Note that t refers to the time of the observation and $\boldsymbol{\epsilon}(t)$ is the noise at t, in contrast to the previous section where *time-like* variables n and s referred to the noise scale.

We could consider obtaining the noise from a standard Wiener process $\boldsymbol{\epsilon}(t) = W_t$. A clear disadvantage of this approach is that variance grows with time. Additionally, the distribution of W_0 is degenerate as we never add any noise. This can be solved in an ad hoc manner by shifting the whole time series similar to Deng et al. (2020).

Instead, in the following, we present two *stationary* stochastic processes that add the same amount of noise regardless of the time of the observation. Note that the noise is *correlated* in the time dimension, hence the use of the stochastic process. An additional nice property of these processes is that they reduce to the diffusion from Section 2 in the trivial case of time series with only one element.

Let us shortly restrict the discussion to univariate time series $\boldsymbol{X} \in \mathbb{R}^M$ and producing noise $\boldsymbol{\epsilon}(\boldsymbol{t}) \in \mathbb{R}^M$. We present the general approach at the end of this section.

A) Gaussian process prior. Given a set of M time points $\boldsymbol{t}$, we propose sampling $\boldsymbol{\epsilon}(\boldsymbol{t})$ from a Gaussian process $\mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$ where each element of the covariance matrix is specified with a kernel $\boldsymbol{\Sigma}_{ij} = k(t_i, t_j)$, where $t_i, t_j \in \boldsymbol{t}$. This produces *smooth* noise functions $\boldsymbol{\epsilon}(\cdot)$ that can be evaluated at any t. To define a stationary process, we have to use a stationary kernel; we will use a radial basis function $k(t_i, t_j) = \exp(-\gamma(t_i - t_j)^2)$. Adjusting the parameter γ (or $\sigma = 1/\gamma$) lets us vary the flatness of the noise curves. Given a set of time points $\boldsymbol{t}$, we can easily sample from this process by first computing the covariance $\boldsymbol{\Sigma}(\boldsymbol{t})$ and then sample from the multivariate normal distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$.

B) Ornstein-Uhlenbeck diffusion. An alternative noise distribution is a stationary OU process that is specified as a solution to the following SDE:

$$\mathrm{d}\epsilon_t = -\gamma\epsilon_t \mathrm{d}t + \mathrm{d}W_t, \tag{8}$$

where W_t is the standard Wiener process and we use the initial condition $\epsilon_0 \sim \mathcal{N}(0, 1)$. We can obtain samples from OU process easily by sampling from a time-changed and scaled Wiener process: $\exp(-\gamma t)W_{\exp(2\gamma t)}$. The covariance can be calculated as $\mathbf{\Sigma}_{ij} = \exp(-\gamma|t_i - t_j|)$. The OU process is a special case of a Gaussian process with a Matérn kernel ($\nu = 0.5$) (Rasmussen & Williams, 2005, p. 86). We discuss different sampling techniques for OU process and their trade-offs in Appendix A.4.

In the end, both the GP and OU processes are defined with a multivariate normal distribution over a finite collection of points, where the covariance is calculated using the times of the observations. As opposed to the methods from Section 2, we use correlated noise in the forward process. Our approach allows us to produce continuous functions as samples and will prove to be a natural way to do forecasting and imputation.

Multivariate time series. In our work, we consider multivariate time series which means we observe an evolution of a d-dimensional vector over time. In the forward diffusion process, we treat the data as d individual univariate time series and add the noise to them independently. This is equivalent to using block-diagonal covariance matrix of size $(Md) \times (Md)$ with $\mathbf{\Sigma}$ repeated on the diagonal. This is in line with the previous works where, e.g., independent noise is added to individual pixels in an image.

Note that this does not mean we do not model correlations between dimensions. As we will see in the following section, the reverse process takes a complete multivariate time series and captures these correlations. This is again similar to image synthesis—although forward process is independent over pixels, the reverse process captures the whole image. The difference in our approach is that we also enforce the continuity across the time dimension, which means our model is guaranteed to produce continuous samples.

3.2. Discrete stochastic process diffusion (DSPD)

We apply the discrete diffusion framework to the time series setting. Note, *discrete* refers to the number of diffusion steps (Section 2.1), i.e., we still model continuous functions. Reusing the notation from before, $\boldsymbol{X}_0$ denotes the input data and $\boldsymbol{X}_n = (\boldsymbol{x}_n(t_0), \dots, \boldsymbol{x}_n(t_{M-1}))$ is the noisy output after n diffusion steps. In contrast to the classical DDPM (Ho et al., 2020) where one adds independent Gaussian noise to data, we now add the noise from a stochastic process. In particular, given the times of the observations, we can compute the covariance $\mathbf{\Sigma}$ and sample noise $\boldsymbol{\epsilon}(\cdot)$ from a GP or OU process as defined in Section 3.1. We write the transition kernel and the posterior as:

$$q(\boldsymbol{X}_n|\boldsymbol{X}_0) = \mathcal{N}(\sqrt{\bar{\alpha}_n}\boldsymbol{X}_0, (1-\bar{\alpha}_n)\mathbf{\Sigma}), \tag{9}$$

$$q(\boldsymbol{X}_{n-1}|\boldsymbol{X}_n, \boldsymbol{X}_0) = \mathcal{N}(\tilde{\boldsymbol{\mu}}_n, \tilde{\beta}_n\mathbf{\Sigma}). \tag{10}$$

We provide a full derivation in Appendix A.1. Even though we are now able to sample functions instead of single points, the distributions are still similar to the previous case with the only change occurring in the covariance. This nice result will be useful later to define the loss which is analogous to Equation 4.

We define the generative model as the reverse process $p(\boldsymbol{X}_{n-1}|\boldsymbol{X}_n) = \mathcal{N}(\boldsymbol{\mu}_\theta(\boldsymbol{X}_n, \boldsymbol{t}, n), \beta_n\mathbf{\Sigma})$, similar to Ho et al. (2020), keeping the time-dependent covariance $\mathbf{\Sigma}$. The key difference is that the model now takes the full time series consisting of noisy observations $\boldsymbol{X}_n$ with their timestamps $\boldsymbol{t}$ in order to predict the noise $\boldsymbol{\epsilon}$ which has the same size as $\boldsymbol{X}_n$. The architecture, therefore, has to be a type of a time series encoder-decoder.

Since each distribution that appears in the ELBO (Equation 3) is multivariate normal, the loss can be calculated in closed-form. In Appendix A.2 we show the full derivation. Further, we show how we can reparameterize the model such that the covariance $\mathbf{\Sigma}$ disappears from the final loss. In particular, if our model predicts the noise that was added to the original data we can simplify the loss to only compute the squared difference between the predicted and true noise, similar to Equation 4:

$$\mathcal{L} = \mathbb{E}_{\boldsymbol{\epsilon},n}\left[\|\boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_n}\boldsymbol{X}_0 + \sqrt{1-\bar{\alpha}_n}\boldsymbol{\epsilon}, \boldsymbol{t}, n) - \boldsymbol{\epsilon}\|_2^2\right]. \tag{11}$$

Finally, in order to sample, the initial noise has to come from a stochastic process instead of an independent normal distribution. The same is the case for the noise that is used in the intermediate steps of the Langevin dynamics. We show the implementation of training (Algorithm 1) and sampling (Algorithm 2) for Gaussian process diffusion in Appendix A.5. The Ornstein-Uhlenbeck case is analogous—we simply change the noise source.

3.3. Continuous stochastic process diffusion (CSPD)

Similarly to the previous section, we can extend the continuous diffusion framework to use the noise coming from a Gaussian or OU process. Now, the noise scales $\beta(s)$ are continuous in the diffusion time s, see Section 2.2. Given a factorized covariance matrix $\mathbf{\Sigma} = \boldsymbol{L}\boldsymbol{L}^T$, we modify the variance preserving diffusion SDE (Song et al., 2021):

$$\mathrm{d}\boldsymbol{X}_s = -\frac{1}{2}\beta(s)\boldsymbol{X}_s\mathrm{d}s + \sqrt{\beta(s)}\boldsymbol{L}\mathrm{d}W_s, \tag{12}$$

which gives us the following transition probability:

$$q(\boldsymbol{X}_s|\boldsymbol{X}_0) = \mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{\mathbf{\Sigma}}), \tag{13}$$

with:

$$\begin{aligned} \tilde{\boldsymbol{\mu}} &= \boldsymbol{X}_0 e^{-\frac{1}{2}\int_0^s \beta(s)\mathrm{d}s} \\ \tilde{\boldsymbol{\Sigma}} &= \boldsymbol{\Sigma}\left(1 - e^{-\int_0^s \beta(s)\mathrm{d}s}\right). \end{aligned} \tag{14}$$

This result is derived using Equation 5.51 from Särkkä & Solin (2019), similar to an analogous result in Song et al. (2021). We discuss this in more detail in Appendix A.3. Since this probability is normal, the value of the score function can be computed in closed-form:

$$\nabla_{\boldsymbol{X}_s} \log q(\boldsymbol{X}_s | \boldsymbol{X}_0) = -\tilde{\boldsymbol{\Sigma}}^{-1}(\boldsymbol{X}_s - \tilde{\boldsymbol{\mu}}), \tag{15}$$

which we can use to optimize the same objective as in Equation 6. Our neural network $\boldsymbol{\epsilon}_\theta(\boldsymbol{X}_s, \boldsymbol{t}, s)$ will take in the full time series, together with the observation times $\boldsymbol{t}$ and the diffusion time s, and predict the values of the score function. As it turns out, we can again use the reparameterization in which we predict the noise, whilst the score is only calculated when sampling new realizations. That is, we represent the score as $\boldsymbol{L}\tilde{\boldsymbol{\epsilon}}/\sigma^2$, where $\sigma^2 = 1 - \exp(-\int_0^s \beta(s)\mathrm{d}s)$ (Equation 15) and $\tilde{\boldsymbol{\epsilon}}$ is the noise coming from an independent normal distribution.

3.4. Related work

Recently, Kerrigan et al. (2023) proposed a similar approach for modeling functions with diffusion by defining a Gaussian measure on Hilbert spaces. This formalizes some of our ideas using the results from measure theory. Another related work (Dutordoir et al., 2022) views diffusion on functions as neural processes, similar to our formulation in Section 4.2. A concurrent work (Phillips et al., 2022) uses KL decomposition to approximate the data in spectral space and then learns a standard diffusion model in this space. On the other hand, our method is well suited for irregular time series data, e.g., it naturally offers conditioning on observed data and performing imputation.

4. Applications

To train a generative model, it must learn to reverse the forward diffusion process by predicting the noise that was added to the clean data. The input to the model is the time series $(\boldsymbol{X}_0, \boldsymbol{t})$ along with the diffusion step n or diffusion time s, and the output is of the same size as $\boldsymbol{X}_0$. If additional inputs are available, we can also model the conditional distribution; e.g., we often have covariates for each time point of $\boldsymbol{t}$. We can also condition the generation on the past observations which essentially defines a probabilistic forecaster (Section 4.1) or condition only on the observed values which defines a neural process (Section 4.2) or an imputation model (Section 4.3).

4.1. Forecasting multivariate time series

Forecasting is answering what is going to happen, given what we have seen, and as such is the most prominent task in time series analysis. Probabilistic forecasting adds the layer of (aleatoric) uncertainty on top of that and returns the confidence intervals which is often a requirement for deploying models in real world settings. The neural forecasters are usually encoder-decoders, where the history of observations $(\boldsymbol{X}^H, \boldsymbol{t}^H)$ is represented with a single vector $\boldsymbol{z}$ and the decoder outputs the distribution of the future values $\boldsymbol{X}^F$ given $\boldsymbol{z}$ at time points $\boldsymbol{t}^F$. Previous works relied on specifying the parameters of the output distribution, e.g., via a diagonal covariance (Salinas et al., 2020), its low-rank approximation (Salinas et al., 2019a), normalizing flows (de Bézenac et al., 2020; Rasul et al., 2021b), or GANs (Koochali et al., 2021).

Recently, Rasul et al. (2021a) introduced a diffusion-based forecasting model to learn the conditional probability $p(\boldsymbol{X}^F | \boldsymbol{X}^H)$, where $\boldsymbol{X}^H = (\boldsymbol{x}(t_0), \dots, \boldsymbol{x}(t_{M-1}))$ is a history window of size M sampled randomly from the full training data. They specify the distribution $p(\boldsymbol{x}(t_M)|\boldsymbol{X}^H)$ using a conditional DDPM model. The forward process adds independent Gaussian noise to $\boldsymbol{x}(t_M)$ the same way as in DDPM. However, the reverse denoising model is conditioned on the history $\boldsymbol{X}^H$ which is represented with a fixed sized vector $\boldsymbol{z}$. After training is completed, the predictions are made in the following way: (1) $\boldsymbol{X}^H$ is encoded with an RNN to obtain $\boldsymbol{z}$; (2) the initial noisy value is sampled $\boldsymbol{x}_N(t_M) \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$; and (3) denoising is performed using the sampling algorithm from Ho et al. (2020) but conditioned on $\boldsymbol{z}$ to obtain $\boldsymbol{x}_0(t_M)$. The final denoised value is the forecast and sampling multiple values allows computing empirical confidence intervals of interest.

In Rasul et al. (2021a), the timestamps are always discrete and the prediction is autoregressive, i.e., the values are produced *one by one*. Our diffusion framework offers these key improvements: (a) the predictions can be made at *any* future time point, i.e., in continuous time, not discrete steps; and (b) we can predict *multiple* values in parallel which scales better on modern hardware.

In our case, the prediction $\boldsymbol{X}^F$ will not be a single vector but a sequence $(\boldsymbol{x}(t_M), \dots, \boldsymbol{x}(t_{M+K}))$ of size K, where K can vary in size. This type of data is naturally handled by our stochastic process diffusion as defined in Section 3. Note that the predicted values are also not conditionally independent but we model the interactions between them in the denoising model $\boldsymbol{\epsilon}_\theta$.

We design $\boldsymbol{\epsilon}_\theta$ in the following way. Previous observations are represented with an RNN to obtain $\boldsymbol{z}$ and condition the reverse process. We propose an architecture similar to the TimeGrad model (Rasul et al., 2021a; Kong et al., 2021),

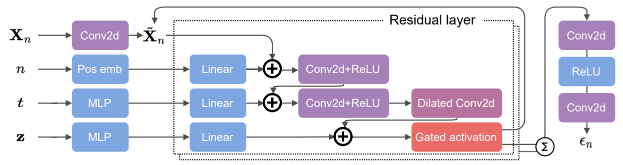

Figure 2. Overview of the forecasting model. The inputs are the noisy time series $\boldsymbol{X}_n$, diffusion steps n, observation times $\boldsymbol{t}$, and the history vector $\boldsymbol{z}$. The output is the predicted noise value $\boldsymbol{\epsilon}_n$. We use two-dimensional convolutions where *height* and *width* correspond to feature and time dimensions.

but not autoregressive as it outputs all the values simultaneously. Figure 2 shows an overview. The model inputs noisy future values $\boldsymbol{X}_n^F$, diffusion step n, future timestamps $\boldsymbol{t}$ and the encoded history $\boldsymbol{z}$. In contrast to previous works, we use 2D convolution where the extra dimension corresponds to the time dimension.

After training a DSPD-GP (Section 3.2), we can forecast:

1. Encode the history $\boldsymbol{X}^H$ with an RNN to get $\boldsymbol{z}$,
2. Sample initial prediction $\boldsymbol{X}_N^F$ from a GP prior,
3. Denoise using $\boldsymbol{\epsilon}_\theta(\boldsymbol{X}_n^F, \boldsymbol{t}, n, \boldsymbol{z})$ with Algorithm 2.

Instead of an RNN we can also use transformers (Vaswani et al., 2017) but we wanted to keep the architecture similar to Rasul et al. (2021a) and showcase the novel stochastic process-based diffusion.

4.2. Diffusion process as a neural process

So far we have used stochastic processes as noise sources to generate continuous functions. We can view such a model as a stochastic process as well. Stochastic process is defined as a collection of random variables $\{X(t)\}_t$ indexed over some set $\mathcal{T}$, in our case $\mathcal{T} \subseteq \mathbb{R}$. We usually care about the finite sequences of points since this is what we encounter in our data. In that case, the model that defines some probability measure p is a stochastic process if it satisfies consistency conditions, as defined in the Kolmogorov extension theorem (Oksendal, 2013). Crucially, the model has to be permutation equivariant, i.e., the order of the incoming points should not matter.

Based on this, neural processes (Garnelo et al., 2018) are a class of latent variable models that define a stochastic process with neural networks. Given a set of data points (a dataset), the model outputs the probability distribution over the functions that generated this dataset. That is, for different datasets, the model will define different stochastic processes. Due to this behavior, neural processes bear a resemblance to the Gaussian processes but can also be viewed as a meta learning model (Hospedales et al., 2021).

Let $\boldsymbol{X}^A$ denote the observed data, in our case, a time series, and let $\boldsymbol{X}^B$ be the unobserved data at the time points $\boldsymbol{t}^B$. Garnelo et al. (2018) construct the encoder-decoder model that uses an amortized variational inference for training (Kingma & Welling, 2014). The encoder takes in a set of observed points $(\boldsymbol{X}^A, \boldsymbol{t}^A)$ and outputs the distribution over the latent variable $q(\boldsymbol{z})$. The decoder takes in the sampled latent vector $\boldsymbol{z}$ and the query time points $\boldsymbol{t}^B$ and predicts the values of the unobserved points $\boldsymbol{X}^B$. In order to produce permutation equivariant measure, it is crucial that the encoder is permutation invariant, i.e., the input order does not alter the result. Then the probability of $\boldsymbol{X}^B$ is conditionally independent given $\boldsymbol{z}$ (De Finetti, 1937). This is easy to achieve using, e.g., deep sets (Zaheer et al., 2017).

Since our approach samples functions, we can condition the generation on an input dataset $(\boldsymbol{X}^A, \boldsymbol{t}^A)$ in order to create our version of a neural process, based purely on the diffusion framework. The encoder will be a deterministic neural network that outputs the latent vector $\boldsymbol{z}$, contrary to Garnelo et al. (2018) which outputs the distribution. Similar to Section 4.1, the diffusion is conditioned on $\boldsymbol{z}$ and we can output samples for any query $\boldsymbol{t}^B$. For example, if we again take DSPD-GP model, we sample as follows:

1. Permutation invariant encode $(\boldsymbol{X}^A, \boldsymbol{t}^A)$ to get $\boldsymbol{z}$,
2. Sample initial points $\boldsymbol{X}_N^B$ at $\boldsymbol{t}^B$ from a GP prior,
3. Denoise using $\boldsymbol{\epsilon}_\theta(\boldsymbol{X}_n^B, \boldsymbol{t}^B, n, \boldsymbol{z})$ with Algorithm 2.

Therefore, we capture the distribution $p(\boldsymbol{X}^B | \boldsymbol{X}^A)$ directly.

We achieve equivariance using a transformer-like model $\boldsymbol{\epsilon}_\theta$ (Vaswani et al., 2017) that uses a learnable RBF kernel for a similarity function. The architecture is described in more detail in Appendix B.3. During training, we adopt the approach of feeding in data such that we learn $p(\boldsymbol{X}^A \cup \boldsymbol{X}^B | \boldsymbol{X}^A)$ which helps our model output high certainty around $\boldsymbol{t}^A$, see Garnelo et al. (2018).

In the end, our model sees many observed-unobserved pairs corresponding to different true underlying processes. The model learns to represent the observed points $\boldsymbol{X}^A$ such that the denoising process corresponds to the correct distribution, given $\boldsymbol{X}^A$. After training is completed, we take a time series $\boldsymbol{X}^A$ and output samples at any set of query time points $\boldsymbol{t}^B$. We can view such an approach as an interpolation or imputation model that fills-in the missing values across time. The main appeal is the ability to capture different stochastic processes within a single model.

A similar idea by Dutordoir et al. (2022) proposes using diffusion as an alternative to Gaussian processes, however, it uses an independent noise, therefore, it does not guarantee producing continuous functions.

4.3. Probabilistic time series imputation

The previous section considered interpolating in time. Now, we look into filling-in the missing values across the observation dimensions, i.e., the imputation of the vectors. Each element $\boldsymbol{x}(t_i)$ of the time series $\boldsymbol{X}$ is assigned a mask $\boldsymbol{m}$ of the same dimension that indicates whether the j-th value $x^{(j)}$ of the vector $\boldsymbol{x}(t_i)$ has been observed ($m^{(j)} = 1$) or if it is missing ($m^{(j)} = 0$).

Given observed $\boldsymbol{X}^A$ and missing points $\boldsymbol{X}^B$, Tashiro et al. (2021) propose a model that learns a conditional distribution $p(\boldsymbol{X}^B|\boldsymbol{X}^A)$. The model is built upon a diffusion framework and the reverse process is conditioned on $\boldsymbol{X}^A$, similar to that in Section 4.2. We extend this by introducing noise from a stochastic process, as presented above. The learnable model remains the same but we introduce the correlated noise in the loss and sampling. We posit that continuous noise process, as an inductive bias for the irregular time series, is a more natural choice.

5. Experiments

5.1. Probabilistic modeling

We start by investigating pure generative capabilities of our model, i.e., unconditional generation of time series.[1]

Baselines. Previously, neural ODEs (Chen et al., 2018) were introduced as a way to capture the irregularly sampled time series since they can naturally handle the continuous time. As such, they can be seen as a building block that can also be used alongside our method to devise different denoising networks. Rubanova et al. (2019) construct an encoder-decoder architecture based on neural ODEs which resembles the variational autoencoder (Kingma & Welling, 2014). The time series is, thus, modeled in a latent space by sampling a random vector which is propagated with an ODE. Neural SDEs (Li et al., 2020) extend this by adding noise in every solver step but they either do not produce noisy-enough samples (Li et al., 2020) or use an adversarial objective which is difficult to train (Kidger et al., 2021). Finally, continuous-time flow process (CTFP) (Deng et al., 2020) uses normalizing flows (Kobyzev et al., 2020) to generate the time series by sampling the initial noise from the stochastic process and transform it with an invertible function to obtain the sample from the target distribution. Although this allows exact likelihood training, the method cannot capture some processes (Deng et al., 2021) and is often augmented to be trained as a VAE.

Data. We generate 6 synthetic datasets, each with 10000 samples, that involve stochastic processes, dynamical and chaotic systems. CIR (Cox-Ingersoll-Ross) is the stochastic differential equation often used in finance, Lorenz is a chaotic system in three dimensions, OU is generated using a specific parameterization of Equation 8, Predator-prey and Sink are two-dimensional dynamical systems, and Sine is generated as a mixture of random sine waves. Full details on generation are included in Appendix B.1.

[1] https://github.com/morganstanley/MSML/tree/main/papers/Stochastic_Process_Diffusion

Table 1. Accuracy of the discriminator trained to distinguish real data and model samples (closer to 0.5 is better).

	CTFP	Latent ODE	DSPD-GP (Our)
CIR	0.998±0.001	1.0±0.0	**0.511±0.028**
Lorenz	0.995±0.006	0.998±0.002	**0.513±0.028**
OU	0.783±0.076	**0.512±0.033**	**0.505±0.045**
Predator-prey	0.789±0.023	0.958±0.021	**0.585±0.022**
Sine	0.981±0.01	1.0±0.0	**0.525±0.009**
Sink	0.726±0.138	0.907±0.039	**0.513±0.01**

Ablation. We test our DSPD and CSPD with independent Gaussian noise and noise from a stochastic process (GP and OU) on the above described datasets. We first check whether using a model that captures interactions across time (e.g., RNN or transformer) outperforms the model that treats each data point in the time series independently. Table 6 (Appendix B.1) shows we need to model the interaction across time, as expected.

Now, we check if having a stochastic process noise is better than the independent Gaussian noise, i.e., we compare our method to Ho et al. (2020); Song et al. (2021). Table 5 (Appendix B.1) shows that using a stochastic process achieves lower negative log-likelihood. We report the results only for CSPD model as it allows likelihood evaluation, whereas DSPD returns ELBO. The results for ELBO are similar. The gap between stochastic and independent noise is especially evident on datasets where we need to generate complicated samples. The difference is less visible in *noisy* datasets, such as CIR, but our method shows much better performance when generating smooth curves such as Lorenz. Finally, Figure 3 demonstrates the quality of the samples.

Results. Since likelihood cannot be evaluated for all models and is not the best indication of the final sampling quality, we report the discriminative score. That is, we quantitatively compare the generative power of our model with the established baselines for irregular time series modeling, namely, latent ODEs (Rubanova et al., 2019) and CTFP (Deng et al., 2020) (details in Appendix B.1) by comparing the quality of the samples. In short, after training a single generative model we sample new data from it. The original and generated data is then used to train a new model that learns to discriminate between them. We then report the performance of a discriminator on the held-out test set. If the discriminator cannot be trained, i.e., its prediction is not better than a random guess, we say the generative model captures the true distribution.

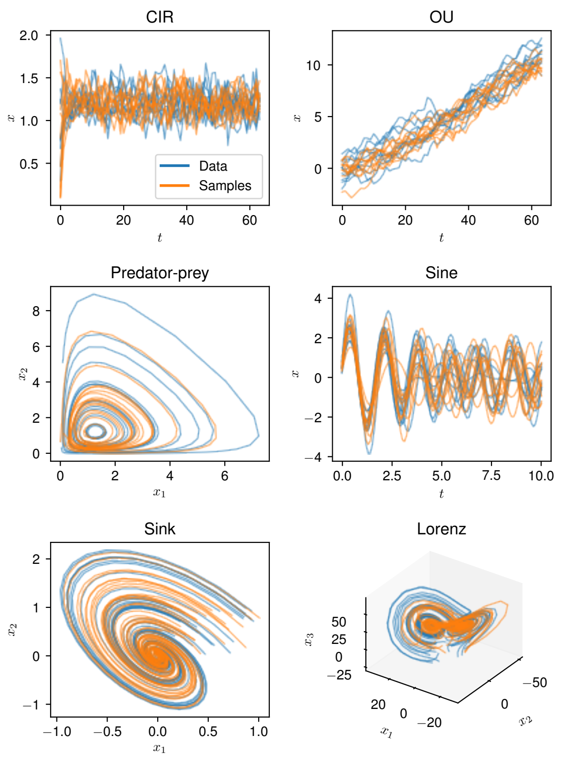

Figure 3. Real data (in blue) and samples from our model (in orange) based on diffusion with Gaussian process noise.

Table 1 compares our model with the baselines and demonstrates that we produce samples that are indistinguishable to a powerful transformer-based discriminator. The same does not hold for the competing methods.

Note that we also implemented the latent SDE model (Li et al., 2020) and we observe that latent ODE outperforms it, which is why we do not include it in the main results.

We notice differences in sampling times for different methods. In particular, CTFP is the fastest followed by our method and neural ODEs, which have similar runtime. Even though diffusion requires evaluating the denoising network N times, ODE approaches oversample in the time dimension. Another difference is that neural ODEs slow down as they learn more complex dynamics since adaptive solver takes more steps. Same can be true for continuous version of diffusion models. The exact times depend on the architecture and hyperparameters as well as other design choices. For example, we present different ways to sample from OU process in Appendix A.4. Depending on the choice we can trade-off having low memory impact (option 2) or ability to parallelize (option 3). We would like to highlight that, for all datasets in this paper, using stochastic process noise did not significantly impact performance compared to independent noise.

Table 2. NRMSE (top rows) and energy score (bottom rows) on real-world forecasting data, averaged over five runs.

	TimeGrad	Ours
Electricity	0.064±0.007	**0.045±0.002**
	8425±613	**7079±164**
Exchange	0.013±0.003	**0.012±0.001**
	0.057±0.002	**0.031±0.002**
Solar	0.799±0.096	**0.757±0.026**
	150±17	166±12

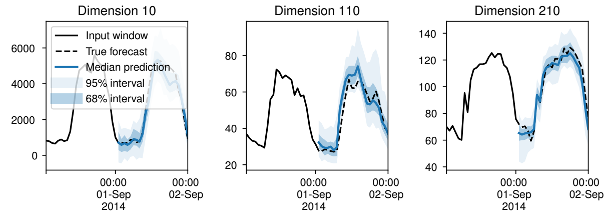

Figure 4. Forecast and uncertainty intervals on Electricity.

5.2. Forecasting

We test our model as defined in Section 4.1 and Figure 2 against TimeGrad (Rasul et al., 2021b) on three established real-world datasets: Electricity, Exchange and Solar (Lai et al., 2018). Due to the limitations of the CRPS-sum metric (Koochali et al., 2022), we report the NRMSE and the energy score (Gneiting & Raftery, 2007) averaged over five runs, but we note that the rank of the model's performance does not change when using other metrics as well. For completeness, we include other time series baselines such as Gaussian process forecaster (Salinas et al., 2019b) in Table 7 (Appendix B.2). Table 2 shows that our method outperforms TimeGrad even though we predict over the complete forecast horizon at once, and Figure 4 demonstrates the prediction quality alongside the uncertainty estimate.

5.3. Neural process

We construct a dataset where each time series $\boldsymbol{X}$ comes from a different stochastic process, by sampling from Gaussian processes with varying kernel parameters and time series lengths. This is a standard training setting in neural process literature (Garnelo et al., 2018). In our denoising network, we modify the attention-like layer to make it stationary (see Appendix B.3) and train as described in Section 4.2. Due to the use of tanh activations in the final layers, combined with its stationary, our model extrapolates well, i.e., when tanh saturates the mean and the variance fall to zero-one. This is the same behaviour we see in the GP with an RBF kernel, for example. The quantile loss of the unobserved data under the true GP model is

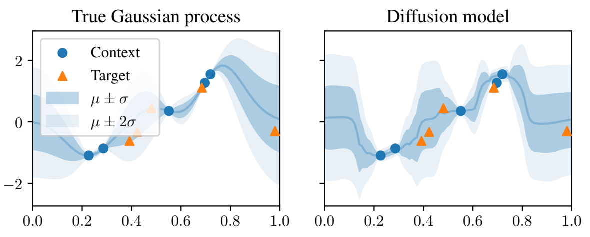

Figure 5. Sampled curves given a set of points.

Table 3. Imputation RMSE on Physionet with varying amounts of missingness. See Appendix B.4 for more results.

Missing	CSDI	DSPD-GP (Our)
10%	0.520±0.055	**0.498±0.036**
50%	**0.644±0.024**	**0.644±0.029**
90%	0.818±0.02	**0.815±0.019**

0.845 while we achieve 0.737 which indicates we capture the true process, which can also be seen in Figure 5. We remark that the attentive neural process (Kim et al., 2019) does not produce the correct uncertainty.

Finally, in Figure 6 we show how model behaves across different kernels. The noise process is connected to the final sample *smoothness* but not the marginal distribution which are always correctly captured.

5.4. Imputation

We compare to the CSDI (Tashiro et al., 2021), introduced in Section 4.3, on an imputation task. To this end, we use exactly the same training setup, including the random seeds and model architecture, but change the noise source to a Gaussian process. Following Tashiro et al. (2021), we use Physionet dataset (Silva et al., 2012) which is a collection of medical time series collected at an hourly rate. It already contains missing values but for testing purposes, we choose varying degrees of missingness and report the results on the test set. We update the loss and sampling accordingly, as in Section 3. Table 3 shows that we outperform the original CSDI model even though we only changed the noise, and the dataset we used has regular time sampling. In Appendix B.4 we provide more details, including the Wilcoxon one-sided signed-rank test (Conover, 1999) that shows our results have statistical significance.

6. Discussion

In this paper, we introduced a novel generative model for continuous functions. It can also be viewed as a neural stochastic process or a generative model for solutions to stochastic differential equations. We also demonstrate how

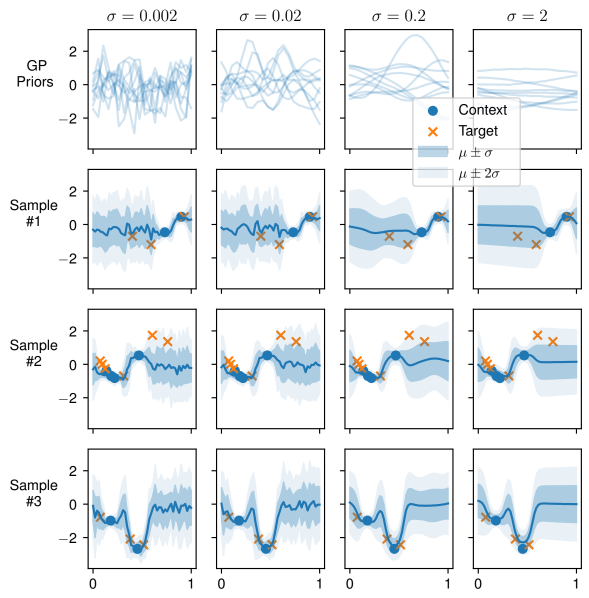

Figure 6. Neural process with Gaussian process diffusion, fitted on GP synthetic data. Columns correspond to different values of the kernel parameter $\sigma = 1/\gamma$. The first row shows samples from the GP prior. As we can see, the higher the value of σ the smoother the process will be. This is also reflected in the samples from the model. We show the same for OU process in Figure 7.

it can be used in conventional (both regular and irregularly-sampled) time series tasks such as forecasting, interpolation, and imputation. In the experiments we showed that the improvements over the previous works come from using the stochastic process as the noise source; and using the model that takes in the whole time series at once. The results demonstrate the practical utility of our method and validate our motivation.

Future work. We used bare bones diffusion without extensive tuning to demonstrate the modeling potential and make a fair comparison to other methods. However, it should be straightforward to improve upon our models by implementing recent advances (e.g., Nichol & Dhariwal, 2021a; Ramos et al., 2022). In case we have a large number of points, we can consider replacing the current sampling strategies with more scalable variants, such as switching to a sparse Gaussian processes (Quiñonero-Candela & Rasmussen, 2005). Finally, we can also apply the presented methods to other areas outside the time series domain, such as modeling point clouds or images, as we have demonstrated that our method is competitive on regular grids.

Acknowledgements

M.B. completed part of the work while interning at Morgan Stanley. M.B. and S.G. are supported by the German Federal Ministry of Education and Research (BMBF), grant no. 01IS18036B.

References

Anand, N. and Achim, T. Protein structure and sequence generation with equivariant denoising diffusion probabilistic models. *arXiv preprint arXiv:2205.15019*, 2022. 3

Anderson, B. D. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12(3):313–326, 1982. 2

Austin, J., Johnson, D. D., Ho, J., Tarlow, D., and van den Berg, R. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. 3

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. 1, 7, 16

Chung, H., Sim, B., and Ye, J. C. Come-closer-diffuse-faster: Accelerating conditional diffusion models for inverse problems through stochastic contraction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022. 3

Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014. 17

Conover, W. J. *Practical nonparametric statistics*, volume 350. john wiley & sons, 1999. 9, 18

de Bézenac, E., Rangapuram, S. S., Benidis, K., Bohlke-Schneider, M., Kurle, R., Stella, L., Hasson, H., Gallinari, P., and Januschowski, T. Normalizing kalman filters for multivariate time series analysis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. 5

De Finetti, B. La prévision: ses lois logiques, ses sources subjectives. In *Annales de l'institut Henri Poincaré*, volume 7, pp. 1–68, 1937. 6

Deng, R., Chang, B., Brubaker, M. A., Mori, G., and Lehrmann, A. Modeling continuous stochastic processes with dynamic normalizing flows. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. 1, 3, 7, 16

Deng, R., Brubaker, M. A., Mori, G., and Lehrmann, A. Continuous latent process flows. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. 7

Dhariwal, P. and Nichol, A. Q. Diffusion models beat gans on image synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. 3

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using real NVP. In *International Conference on Learning Representations (ICLR)*, 2017. 16

Dutordoir, V., Saul, A., Ghahramani, Z., and Simpson, F. Neural diffusion processes. *arXiv preprint arXiv:2206.03992*, 2022. 5, 6

Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S., and Teh, Y. W. Neural processes. In *ICML 2018 workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018. 1, 6, 8

Gneiting, T. and Raftery, A. E. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007. 8

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020. 3

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. 1, 2, 4, 5, 7, 13, 14

Hospedales, T., Antoniou, A., Micaelli, P., and Storkey, A. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44 (9):5149–5169, 2021. 6

Jolicoeur-Martineau, A., Li, K., Piché-Taillefer, R., Kachman, T., and Mitliagkas, I. Gotta go fast when generating data with score-based models. *arXiv preprint arXiv:2105.14080*, 2021. 3

Kerrigan, G., Ley, J., and Smyth, P. Diffusion generative models in infinite dimensions. *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2023. 5

Kidger, P., Foster, J., Li, X., and Lyons, T. J. Neural sdes as infinite-dimensional gans. In *International Conference on Machine Learning (ICML)*, 2021. 7

Kim, H., Mnih, A., Schwarz, J., Garnelo, M., Eslami, S. M. A., Rosenbaum, D., Vinyals, O., and Teh, Y. W. Attentive neural processes. In *International Conference on Learning Representations, (ICLR)*, 2019. 9

Kingma, D., Salimans, T., Poole, B., and Ho, J. Variational diffusion models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. 3

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*, 2014. 6, 7

Kobyzev, I., Prince, S. J., and Brubaker, M. A. Normalizing flows: An introduction and review of current methods. *IEEE transactions on pattern analysis and machine intelligence*, 43(11):3964–3979, 2020. 7, 16

Kong, Z., Ping, W., Huang, J., Zhao, K., and Catanzaro, B. Diffwave: A versatile diffusion model for audio synthesis. In *International Conference on Learning Representations (ICLR)*, 2021. 3, 5

Koochali, A., Dengel, A., and Ahmed, S. If you like it, GAN it—probabilistic multivariate times series forecast with GAN. *Engineering Proceedings*, 5(1), 2021. 5

Koochali, A., Schichtel, P., Dengel, A., and Ahmed, S. Random noise vs. state-of-the-art probabilistic forecasting methods: A case study on crps-sum discrimination ability. *Applied Sciences*, 12(10):5104, 2022. 8

Lai, G., Chang, W., Yang, Y., and Liu, H. Modeling long- and short-term temporal patterns with deep neural networks. In *ACM SIGIR Conference on Research & Development in Information Retrieval*, 2018. 8

Lee, J. S. and Kim, P. M. Proteinsgm: Score-based generative modeling for de novo protein design. *bioRxiv*, 2022. 3

Li, X., Wong, T.-K. L., Chen, R. T. Q., and Duvenaud, D. Scalable gradients for stochastic differential equations. In *International Conference on Artificial Intelligence and Statistics*, 2020. 1, 7, 8

Lyu, Z., Xu, X., Yang, C., Lin, D., and Dai, B. Accelerating diffusion models via early stop of the diffusion process. *arXiv preprint arXiv:2205.12524*, 2022. 3

Nichol, A. Q. and Dhariwal, P. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning (ICML)*, 2021a. 9

Nichol, A. Q. and Dhariwal, P. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning (ICML)*, 2021b. 3

Oksendal, B. *Stochastic differential equations: an introduction with applications*. Springer Science & Business Media, 2013. 6

Phillips, A., Seror, T., Hutchinson, M., De Bortoli, V., Doucet, A., and Mathieu, E. Spectral diffusion processes. *arXiv preprint arXiv:2209.14125*, 2022. 5

Quiñonero-Candela, J. and Rasmussen, C. E. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6(65): 1939–1959, 2005. 9

Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., and Chen, M. Hierarchical text-conditional image generation with CLIP latents. *arXiv preprint arXiv:2204.06125*, 2022. 3

Ramos, A. G. C. P., Mehrotra, A., Lane, N. D., and Bhattacharya, S. Conditioning sequence-to-sequence networks with learned activations. In *International Conference on Learning Representations (ICLR)*, 2022. 9

Rasmussen, C. E. and Williams, C. K. I. *Gaussian Processes for Machine Learning*. The MIT Press, 2005. 4

Rasul, K., Seward, C., Schuster, I., and Vollgraf, R. Autoregressive denoising diffusion models for multivariate probabilistic time series forecasting. In *International Conference on Machine Learning (ICML)*, 2021a. 3, 5, 6

Rasul, K., Sheikh, A.-S., Schuster, I., Bergmann, U. M., and Vollgraf, R. Multivariate probabilistic time series forecasting via conditioned normalizing flows. In *International Conference on Learning Representations (ICLR)*, 2021b. 5, 8

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. *arXiv preprint arXiv:2112.10752*, 2021. 3

Rubanova, Y., Chen, R. T. Q., and Duvenaud, D. K. Latent ordinary differential equations for irregularly-sampled time series. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019. 7

Salinas, D., Bohlke-Schneider, M., Callot, L., Medico, R., and Gasthaus, J. High-dimensional multivariate forecasting with low-rank Gaussian Copula Processes. In *Advances in Neural Information Processing Systems (NeurIPS)*. 2019a. 5

Salinas, D., Bohlke-Schneider, M., Callot, L., Medico, R., and Gasthaus, J. High-dimensional multivariate forecasting with low-rank gaussian copula processes. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019b. 8, 17

Salinas, D., Flunkert, V., Gasthaus, J., and Januschowski, T. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3):1181–1191, 2020. 5

Silva, I., Moody, G., Scott, D. J., Celi, L. A., and Mark, R. G. Predicting in-hospital mortality of icu patients: The physionet/computing in cardiology challenge 2012. In *2012 Computing in Cardiology*. IEEE, 2012. 9

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. Deep Unsupervised Learning using Nonequilibrium Thermodynamics. In *International Conference on Machine Learning (ICML)*, 2015. 2

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations (ICLR)*, 2021. 1, 2, 3, 4, 5, 7, 14, 15, 17

Särkkä, S. and Solin, A. *Applied Stochastic Differential Equations*. Institute of Mathematical Statistics Textbooks. Cambridge University Press, 2019. 5, 14

Tashiro, Y., Song, J., Song, Y., and Ermon, S. Csdi: Conditional score-based diffusion models for probabilistic time series imputation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. 7, 9, 18

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. 6, 16

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. Deep sets. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. 6

A. Derivations

A.1. Discrete diffusion posterior probability

We extend Ho et al. (2020) by using full covariance $\boldsymbol{\Sigma}(\boldsymbol{t})$ to define the noise distribution across time $\boldsymbol{t}$. If $\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^T$ and keeping the same definitions from Section 2.1 for β_n, α_n, and $\bar{\alpha}_n$, we can write:

$$\boldsymbol{X}_n = \sqrt{1-\beta_n}\boldsymbol{X}_{n-1} + \sqrt{\beta_n}\boldsymbol{L}\boldsymbol{\epsilon}, \tag{16}$$

$$\boldsymbol{X}_n = \sqrt{\bar{\alpha}_n}\boldsymbol{X}_0 + \sqrt{1-\bar{\alpha}_n}\boldsymbol{L}\boldsymbol{\epsilon}, \tag{17}$$

with $\boldsymbol{\epsilon} \in \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. This corresponds to the following transition distributions:

$$q(\boldsymbol{X}_n|\boldsymbol{X}_{n-1}) = \mathcal{N}(\sqrt{1-\beta_n}\boldsymbol{X}_{n-1}, \beta_n\boldsymbol{\Sigma}), \tag{18}$$

$$q(\boldsymbol{X}_n|\boldsymbol{X}_0) = \mathcal{N}(\sqrt{\bar{\alpha}_n}\boldsymbol{X}_0, (1-\bar{\alpha}_n)\boldsymbol{\Sigma}). \tag{19}$$

We are interested in $q(\boldsymbol{X}_{n-1}|\boldsymbol{X}_n, \boldsymbol{X}_0) \propto q(\boldsymbol{X}_n|\boldsymbol{X}_{n-1})q(\boldsymbol{X}_{n-1}|\boldsymbol{X}_0)$. Since both distributions on the right-hand side are normal, the result will be normal as well. We can write the resulting distribution as $\mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\Sigma}})$, where:

$$\begin{aligned}
\tilde{\boldsymbol{\mu}} &= \boldsymbol{R}(\boldsymbol{X}_n - \boldsymbol{A}\boldsymbol{\mu}_1) + \boldsymbol{\mu}_1 \\
\tilde{\boldsymbol{\Sigma}} &= \boldsymbol{\Sigma}_1 - \boldsymbol{R}\boldsymbol{A}\boldsymbol{\Sigma}_1^T \\
\boldsymbol{R} &= \boldsymbol{\Sigma}_1\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{\Sigma}_1\boldsymbol{A}^T + \boldsymbol{\Sigma}_2)^{-1},
\end{aligned}$$

with $\boldsymbol{A} = \sqrt{1-\beta_n}\boldsymbol{I}$, $\boldsymbol{\mu}_1 = \sqrt{\bar{\alpha}_{n-1}}\boldsymbol{X}_0$, $\boldsymbol{\Sigma}_1 = (1-\bar{\alpha}_{n-1})\boldsymbol{\Sigma}$, and $\boldsymbol{\Sigma}_2 = \beta_n\boldsymbol{\Sigma}$. We can now write:

$$\begin{aligned}
\boldsymbol{R} &= (1-\bar{\alpha}_{n-1})\boldsymbol{\Sigma}\sqrt{1-\beta_n}\left(\sqrt{1-\beta_n}(1-\bar{\alpha}_{n-1})\boldsymbol{\Sigma}\sqrt{1-\beta_n} + \beta_n\boldsymbol{\Sigma}\right)^{-1} \\
&= \frac{(1-\bar{\alpha}_{n-1})\sqrt{\alpha_n}}{\alpha_n(1-\bar{\alpha}_{n-1}) + 1 - \alpha_n}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^{-1} \\
&= \frac{1-\bar{\alpha}_{n-1}}{1-\bar{\alpha}_n}\sqrt{\alpha_n},
\end{aligned}$$

and from there:

$$\begin{aligned}
\tilde{\boldsymbol{\mu}} &= \frac{1-\bar{\alpha}_{n-1}}{1-\bar{\alpha}_n}\sqrt{\alpha_n}\left(\boldsymbol{X}_n - \sqrt{1-\beta_n}\sqrt{\bar{\alpha}_{n-1}}\boldsymbol{X}_0\right) + \sqrt{\bar{\alpha}_{n-1}}\boldsymbol{X}_0 \\
&= \frac{1-\bar{\alpha}_{n-1}}{1-\bar{\alpha}_n}\sqrt{\alpha_n}\boldsymbol{X}_n + \sqrt{\bar{\alpha}_{n-1}}\left(1 - \frac{1-\bar{\alpha}_{n-1}}{1-\bar{\alpha}_n}\alpha_n\right)\boldsymbol{X}_0 \\
&= \frac{1-\bar{\alpha}_{n-1}}{1-\bar{\alpha}_n}\sqrt{\alpha_n}\boldsymbol{X}_n + \frac{\sqrt{\bar{\alpha}_{n-1}}}{1-\bar{\alpha}_n}\beta_n\boldsymbol{X}_0,
\end{aligned} \tag{20}$$

and using the fact that $\boldsymbol{\Sigma}$ is a symmetric matrix:

$$\begin{aligned}
\tilde{\boldsymbol{\Sigma}} &= (1-\bar{\alpha}_{n-1})\boldsymbol{\Sigma} - \frac{1-\bar{\alpha}_{n-1}}{1-\bar{\alpha}_n}\sqrt{\alpha_n}\sqrt{1-\beta_n}(1-\bar{\alpha}_{n-1})\boldsymbol{\Sigma}^T \\
&= \left(1-\bar{\alpha}_{n-1} - \frac{1-\bar{\alpha}_{n-1}}{1-\bar{\alpha}_n}\alpha_n(1-\bar{\alpha}_{n-1})\right)\boldsymbol{\Sigma} \\
&= \frac{1-\bar{\alpha}_{n-1}}{1-\bar{\alpha}_n}\beta_n\boldsymbol{\Sigma}.
\end{aligned} \tag{21}$$

Therefore, the only difference to the derivation in Ho et al. (2020) is the $\boldsymbol{\Sigma}(\boldsymbol{t})$ instead of the identity matrix $\boldsymbol{I}$ in the covariance.

A.2. Discrete diffusion loss

We use the evidence lower bound from Equation 3. The distribution $q(\boldsymbol{X}_{n-1}|\boldsymbol{X}_n, \boldsymbol{X}_0)$ is defined as $\mathcal{N}(\tilde{\boldsymbol{\mu}}, C_1\boldsymbol{\Sigma})$, where C_1 is some constant (Equations 20 and 21). Similar to Ho et al. (2020), we start with the parameterization for the reverse process $p(\boldsymbol{X}_{n-1}|\boldsymbol{X}_n) = \mathcal{N}(\boldsymbol{\mu}_\theta(\boldsymbol{X}_n, \boldsymbol{t}, n), \beta_n \boldsymbol{\Sigma})$, where:

$$\boldsymbol{\mu}_\theta(\boldsymbol{X}_n, \boldsymbol{t}, n) = \frac{1}{\sqrt{\alpha_n}} \left(\boldsymbol{X}_n - \frac{\beta_n}{\sqrt{1-\bar{\alpha}_n}} \boldsymbol{\epsilon}_\theta(\boldsymbol{X}_n, \boldsymbol{t}, n) \right).$$

Then the KL-divergence is between two normal distributions so we can write the following, where C_2 is a term that does not depend on the parameters θ:

$$\begin{aligned} D_{\mathrm{KL}}[q(\boldsymbol{X}_{n-1}|\boldsymbol{X}_n, \boldsymbol{X}_0) \parallel p(\boldsymbol{X}_{n-1}|\boldsymbol{X}_n)] &= D_{\mathrm{KL}}[\mathcal{N}(\tilde{\boldsymbol{\mu}}, C_1 \boldsymbol{\Sigma}) \parallel \mathcal{N}(\boldsymbol{\mu}_\theta(\boldsymbol{X}_n, \boldsymbol{t}, n), \beta_n \boldsymbol{\Sigma})] \\ &= \frac{1}{2}(\tilde{\boldsymbol{\mu}} - \boldsymbol{\mu}_\theta)^T \boldsymbol{\Sigma}^{-1} (\tilde{\boldsymbol{\mu}} - \boldsymbol{\mu}_\theta) + C_2. \end{aligned}$$

Ho et al. (2020) show that their loss can be simplified to Equation 4 given their particular parameterization. Recall that we obtain noise by computing $\boldsymbol{L}\tilde{\boldsymbol{\epsilon}}$, where $\tilde{\boldsymbol{\epsilon}}$ is unit normal and $\boldsymbol{L}$ is the lower triangular matrix from the Cholesky decomposition of the covariance $\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^T$.

Therefore, we can factorize $\boldsymbol{L}$ from the bracket containing the difference of two means to get:

$$D_{\mathrm{KL}}[q(\boldsymbol{X}_{n-1}|\boldsymbol{X}_n, \boldsymbol{X}_0) \parallel p(\boldsymbol{X}_{n-1}|\boldsymbol{X}_n)] = (\boldsymbol{L}\boldsymbol{a})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{L}\boldsymbol{a}) = \boldsymbol{a}^T \boldsymbol{L}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{L}\boldsymbol{a},$$

where we write $\boldsymbol{a}$ as a shorthand for the term depending on $\boldsymbol{X}_0$ and unit normal noise $\tilde{\boldsymbol{\epsilon}}$. The term $\boldsymbol{L}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{L}$ evaluates to identity and we are again left with the same loss as in Ho et al. (2020). That is, we can use the same trick to simplify the loss to be the mean squared error between the true noise and the predicted noise, which leads to faster evaluation and better results. Note that in the above notation, we have a set of observations $\boldsymbol{X}$ for times $\boldsymbol{t}$ that we feed into the model $\boldsymbol{\epsilon}_\theta$ to predict a set of noise values $\boldsymbol{\epsilon}(t)$, $t \in \boldsymbol{t}$, whereas, previous works predicted the noise for each data point independently.

A.3. Continuous diffusion transition probability

Given an SDE in Equation 12 we want to compute the change in the variance $\tilde{\boldsymbol{\Sigma}}_s$, where s denotes the diffusion time. The derivation is similar to that in Song et al. (2021). We start with the Equation 5.51 from Särkkä & Solin (2019):

$$\frac{\mathrm{d}\tilde{\boldsymbol{\Sigma}}_s}{\mathrm{d}s} = \mathbb{E}[f(\boldsymbol{X}_s, s)(\boldsymbol{X}_s - \boldsymbol{\mu})^T] + \mathbb{E}[(\boldsymbol{X}_s - \boldsymbol{\mu}) f(\boldsymbol{X}_s, s)^T] + \mathbb{E}[\boldsymbol{L}(\boldsymbol{X}_s, s)\boldsymbol{Q}\boldsymbol{L}(\boldsymbol{X}_s, s)^T],$$

where f is the drift, $\boldsymbol{L}$ is the SDE diffusion term and $\boldsymbol{Q}$ is the diffusion matrix. From here, the only difference to Song et al. (2021) is in the last term; they obtain $\beta(s)\boldsymbol{I}$ while we have a full covariance matrix from the stochastic process: $\beta(s)\boldsymbol{\Sigma}$. Therefore, we only need to slightly modify the result:

$$\frac{\mathrm{d}\boldsymbol{\Sigma}_s}{\mathrm{d}s} = \beta(s)(\boldsymbol{\Sigma} - \tilde{\boldsymbol{\Sigma}}_s),$$

which will give us the covariance of the transition probability as in Equation 13. The derivation for the mean is unchanged as our drift term is the same as in Song et al. (2021).

A.4. Sampling from an Ornstein-Uhlenbeck process

In the following, we discuss three different approaches to sampling noise $\boldsymbol{\epsilon}(\cdot)$ from an OU process defined by γ at time points $t_0, \dots, t_{M-1}$.

1. **Modified Wiener.** As we already mentioned in Section 3.1, we can use a time-changed and scaled Wiener process: $e^{-\gamma t} W_{e^{2\gamma t}}$. Sampling from a Wiener process is straightforward: given a set of time increments $\Delta t_0, \dots, \Delta t_{M-1}$, we sample M points independently from $\mathcal{N}(0, \Delta t_i)$ and cumulatively sum all the samples. The time changed process first needs to reparameterize the time values. The issue arises when applying the exponential for large t which leads to numerical instability. This can be mitigated by re-scaling t.

2. **Discretized SDE.** A numerically stable approach involves *solving* the OU SDE in fixed steps. The point at $t = 0$, $\boldsymbol{\epsilon}(0)$ is sampled from unit Gaussian. After that, each point is obtained based on the previous, i.e., i-th point $\boldsymbol{\epsilon}(t_i)$ is calculated as $\boldsymbol{\epsilon}(t_i) = c\boldsymbol{\epsilon}(t_{i-1}) + \sqrt{1-c^2}z$, where $c = \exp(-\gamma(t_i - t_{i-1}))$ and $z \sim \mathcal{N}(0, 1)$. This is an iterative procedure but is quite fast and stable.

3. **Multivariate normal.** Finally, we can treat the process as a multivariate normal distribution with mean zero and covariance $\boldsymbol{\Sigma}_{ij}(t_i, t_j) = \exp(-\gamma|t_i - t_j|)$. Given a set of time points $\boldsymbol{t}$ it is easy to obtain the covariance matrix $\boldsymbol{\Sigma}$ and its factorization $\boldsymbol{L}^T\boldsymbol{L}$. To sample, we first draw $\tilde{\boldsymbol{\epsilon}} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and then $\boldsymbol{\epsilon} = \boldsymbol{L}\tilde{\boldsymbol{\epsilon}}$. Since our model performs best if it predicts $\tilde{\boldsymbol{\epsilon}}$, we opted for this particular sampling approach. If $\boldsymbol{t}$ is not changing, $\boldsymbol{L}$ can be computed once and the performance impact will be minimal. Also when sampling new realizations, $\boldsymbol{L}$ has to be computed only once, before the sampling loop (see Algorithm 2).

A.5. Algorithms

In Algorithms 1 and 2 we provide the pseudocode for training the model and sampling new data, for DSPD-GP model. Analogously for OU, we would replace the noise source using the third algorithm from Appendix A.4. For the score-based model we compute the mean squared error between the predicted and true conditional score function and the sampling uses either ODE or SDE solver, just like in Song et al. (2021).

Algorithm 1 Training (DSPD-GP diffusion)

```
1: while not converged do
2:     $\boldsymbol{X}_0, \boldsymbol{t} \sim p_{\text{data}}(\boldsymbol{X}, \boldsymbol{t})$
3:     $\boldsymbol{\Sigma} = k(\boldsymbol{t}, \boldsymbol{t}^T)$
4:     $\boldsymbol{L} = \text{Cholesky}(\boldsymbol{\Sigma})$
5:     $\tilde{\boldsymbol{\epsilon}} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$
6:     $n \sim \mathcal{U}(\{1, \dots, N\})$
7:     $\boldsymbol{X}_n = \sqrt{\alpha_n}\boldsymbol{X}_0 + \sqrt{1-\alpha_n}\boldsymbol{L}\tilde{\boldsymbol{\epsilon}}$
8:     Take gradient step on
9:         $\nabla_\theta ||\tilde{\boldsymbol{\epsilon}} - \boldsymbol{\epsilon}_\theta(\boldsymbol{X}_n, \boldsymbol{t}, n)||_2^2$
10: end while
```

Algorithm 2 Sampling (DSPD-GP diffusion)

```
1: input: $\boldsymbol{t} = (t_0, \dots, t_{M-1})$
2: $\boldsymbol{\Sigma} = k(\boldsymbol{t}, \boldsymbol{t}^T)$; $\boldsymbol{L} = \text{Cholesky}(\boldsymbol{\Sigma})$
3: $\boldsymbol{X}_N \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma})$
4: for $n = N, \dots, 1$ do
5:     $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma})$
6:     $\boldsymbol{X}_{n-1} = \frac{1}{\sqrt{\alpha_n}}\left(\boldsymbol{X}_n - \frac{1-\alpha_n}{\sqrt{1-\bar{\alpha}_n}}\boldsymbol{L}\boldsymbol{\epsilon}_\theta(\boldsymbol{X}_n, \boldsymbol{t}, n)\right) + \beta_n\boldsymbol{z}$
7: end for
8: return $\boldsymbol{X}_0$
```

B. Experimental details

B.1. Probabilistic modeling

B.1.1. Datasets

The properties of the open datasets used in the forecasting experiment are detailed in Table 4. Additionally, we generate 6 synthetic datasets, each with 10000 samples, that involve stochastic processes, dynamical and chaotic systems.

1. CIR (Cox-Ingersoll-Ross SDE) is the stochastic differential equations defined by:

$$\mathrm{d}x = a(b - x)\mathrm{d}t + \sigma\sqrt{x}\mathrm{d}W_t,$$

   where we set $a = 1$, $b = 1.2$, $\sigma = 0.2$ and sample $x_0 \sim \mathcal{N}(0, 1)$ but only take the positive values, otherwise the $\sqrt{x}$ term is undefined. We solve for $t \in \{1, \dots, 64\}$.

2. Lorenz is a chaotic system in three dimensions. It is governed by the following equations:

$$\begin{aligned} \dot{x} &= \sigma(y - x), \\ \dot{y} &= \rho x - y - xz, \\ \dot{z} &= xy - \beta z, \end{aligned}$$

   where $\rho = 28$, $\sigma = 10$, $\beta = 2.667$, and t is sampled 100 times, uniformly on $[0, 2]$, and $x, y, z \sim \mathcal{N}(\boldsymbol{0}, 100\boldsymbol{I})$.

3. Ornstein-Uhlenbeck is defined as:

$$\mathrm{d}x = (\mu t - \theta x)\mathrm{d}t + \sigma \mathrm{d}W_t,$$

   with $\mu = 0.02$, $\theta = 0.1$ and $\sigma = 0.4$. We sample time the same way as for CIR.

Table 4. Multivariate dimension, domain, frequency, total training time steps, and prediction length properties of the training datasets used in the forecasting experiments.

Dataset	Dim. d	Dom.	Freq.	Time steps	Pred. steps
Exchange	8	$\mathbb{R}^+$	day	$6,071$	30
Solar	137	$\mathbb{R}^+$	hour	$7,009$	24
Electricity	370	$\mathbb{R}^+$	hour	$5,833$	24

4. Predator-prey is a 2D dynamical system defined with an ODE:

$$\dot{x} = 2/3x - 2/3xy,$$
$$\dot{y} = xy - y.$$

5. Sine dataset is generated as a mixture of 5 random sine waves $a\sin(bx + c)$, where $a \sim \mathcal{N}(3, 1)$, $b \sim \mathcal{N}(0, 0.25)$, and $c \sim \mathcal{N}(0, 1)$.

6. Sink is again a dynamical system, governed by:

$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = \begin{bmatrix} -4 & 10 \\ -3 & 2 \end{bmatrix} \boldsymbol{x},$$

with $\boldsymbol{x}_0 \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.

B.1.2. CTFP

We implement continuous-time flow process (Deng et al., 2020) which is a normalizing flow model for stochastic processes. That is, there is a predefined base distribution $p(\boldsymbol{z})$ and a series of invertible transformations f such that we can generate samples $\boldsymbol{x} = f(\boldsymbol{z})$, and evaluate the density in closed-form by computing $\boldsymbol{z} = f^{-1}(\boldsymbol{x})$ and using the change of variables formula. For more details on normalizing flows, see Kobyzev et al. (2020). The novel idea in CTFP is to change the base density to a stochastic process, i.e., a Wiener process, to obtain the distribution over the functions, similar to our work. In our case, we do not use invertible functions but learn to inverse the noising process, and additionally, we add noise at multiple levels instead only in the beginning. In the experiments, we define a CTFP model as a 12-layer real NVP architecture (Dinh et al., 2017) with 2 hidden layers in each layer's MLP.

B.1.3. Latent ODE

Latent ODE is a variational autoencoder architecture, with an encoder that represents the complete time series as a single vector following $q(\boldsymbol{z})$, and a decoder that produces the samples at observation times t_i, $\boldsymbol{z}(t_i) = f(\boldsymbol{z}), \boldsymbol{z} \sim q(\boldsymbol{z})$. The final step is projection to a data space $\boldsymbol{q}(t_i) \mapsto \boldsymbol{x}(t_i)$. The key idea is to use the neural ordinary differential equation (Chen et al., 2018) to define the evolution of the latent variable $\boldsymbol{z}(\cdot)$, thus, have a probabilistic model of the function. This is different from our approach as it models the function in a latent space, with a single source of randomness at the beginning of the time series. That is, the random value is sampled at $t = 0$ and the time series is determined from there onward, whereas our method samples random values on the whole interval $[0, T]$ and does so multiple times (for N diffusion steps) until we get the new realization. In the experiments, we use a two layer neural network for the neural ODE, and another two layer network for projection to the data space.

B.1.4. Our models

We use two models, one is a simple feedforward network, and the second is an RNN-based model. We also use a simple transformer-based model (Vaswani et al., 2017) that achieves similar results to an RNN. The model takes in the time series $\boldsymbol{X}$, times of the observations $\boldsymbol{t}$ and the diffusion step n or diffusion time s. The output is the same size as $\boldsymbol{X}$. The feedforward model embeds the time and the diffusion step with a positional encoding (Vaswani et al., 2017) and passes it together with $\boldsymbol{X}$ through the multilayer neural network. Here, there is no interaction between the points along the time dimension. The model, however, has the capacity to learn transformation based on time of observation. The second model

Table 5. Negative log-likelihood on synthetic data (lower is better) shows OU/GP is mostly better than independent noise.

	CIR	Lorenz	OU	Predator-prey	Sine	Sink
Song et al. (2021)	-0.4769±0.0249	1.5162±0.3861	0.5105±0.0088	-3.4643±0.1039	-1.3338±0.0863	-5.6637±0.1839
CSPD-GP	-0.4766±0.0224	-3.4893±8.2181	0.5202±0.0255	-9.4478±0.2466	-3.4878±1.3467	-11.4179±0.3627
CSPD-OU	-0.4688±0.0178	-6.6707±0.175	0.5239±0.0639	-7.0098±1.4929	-3.5324±0.6466	-9.5349±1.3183

Table 6. Accuracy of the discriminator trained on samples from a diffusion model. Values around 0.5 indicate the discriminative model cannot distinguish the model samples and real data. Values closer to 1 indicate the generative model is not capturing the data distribution.

		CIR	Lorenz	OU	Predator-prey	Sine	Sink
RNN-based model							
DSPD	Gauss	0.5245±0.0252	0.512±0.0212	0.568±0.051	0.5275±0.0383	0.5565±0.0353	0.526±0.0085
	GP	0.5115±0.0282	0.5135±0.0288	0.5055±0.0458	0.5855±0.0219	0.5255±0.009	0.513±0.0103
	OU	0.514±0.0737	0.6095±0.0964	0.5605±0.0581	0.5865±0.053	0.507±0.11	0.6255±0.1672
CSPD	Gauss	0.644±0.0373	0.5015±0.0243	0.6105±0.0153	0.548±0.0751	0.611±0.0516	0.5495±0.0313
	GP	0.5795±0.0541	0.674±0.0739	0.5025±0.0622	0.607±0.0538	0.5575±0.0376	0.5345±0.0201
	OU	0.4535±0.165	0.715±0.0884	0.5255±0.011	0.5835±0.0723	0.556±0.118	0.5795±0.0173
Feedforward model							
DSPD	Gauss	0.624±0.0438	0.713±0.1798	0.5275±0.0371	1.0±0.0	0.7875±0.0585	0.9695±0.0302
	GP	0.558±0.0611	0.894±0.212	0.5535±0.1152	0.7565±0.1362	0.735±0.2146	0.784±0.2281
	OU	1.0±0.0	1.0±0.0	1.0±0.0	1.0±0.0	1.0±0.0	1.0±0.0
CSPD	Gauss	0.537±0.0458	0.959±0.0808	0.5155±0.0165	0.9995±0.001	0.6335±0.0765	0.9095±0.1306
	GP	0.645±0.1034	1.0±0.0	0.507±0.0264	0.894±0.212	0.894±0.212	0.88±0.088
	OU	0.984±0.032	1.0±0.0	0.9905±0.019	1.0±0.0	1.0±0.0	1.0±0.0

is RNN based, that is, we pass the same concatenated input as before to a 2-layer bidirectional GRU (Chung et al., 2014) and use a single linear layer to project to the output dimension. Table 6 shows that it is important to have interactions in the time dimension, regardless of the noise source, because otherwise we only learn the marginal distribution and the quality of the samples suffers.

B.2. Multivariate probabilistic forecasting

Table 7. CRPS-sum results on forecasting task. Values for non-diffusion baselines are taken from Salinas et al. (2019b).

	Electricity	Exchange	Solar
LSTM	0.025±0.001	0.008±0.001	0.391±0.017
LSTM-Copula	0.064±0.008	0.007±0.000	0.319±0.011
GP	0.947±0.016	0.011±0.001	0.828±0.010
GP-Copula	0.024±0.002	0.007±0.000	0.337±0.024
TimeGrad	0.036±0.002	0.009±0.001	0.389±0.041
Our	0.027±0.001	0.007±0.001	0.371±0.034

B.3. Neural process

B.3.1. Dataset

We sample points from a Gaussian process to obtain a single time series. In the end, we have 8000 time series and 2000 test time series. We sample the number of time points from a Poisson distribution with $\lambda = 10$ but restrict the values to always be above 5 and below 50. The time points are sampled uniformly on $[0, 1]$. The observations are sampled from a multivariate normal distribution with mean zero and covariance obtained from an RBF kernel. The σ value in the kernel is uniformly sampled in $[0.01, 0.05]$ for each time series independently. Half of the sampled points are treated as unobserved while the rest are used as a context in the model.

B.3.2. Model

The denoising model takes in $\boldsymbol{X}^A$ (observed points) as a conditioning variable and $\boldsymbol{X}_n^B$ (target points) as the noisy input. We first run a learnable RBF kernel $k(\boldsymbol{t}^A, \boldsymbol{t}^B)$ to obtain a similarity matrix $\boldsymbol{K}$ between the observed and unobserved time points. We project $\boldsymbol{X}^A$ with a neural network by transforming each point independently to obtain $\boldsymbol{Z}$, and then obtain the latent variable of the same time dimension size as $\boldsymbol{X}^B$ by multiplying $\boldsymbol{K}$ and $\boldsymbol{Z}$. We then use $\boldsymbol{Z}$ as a conditioning vector and add it to projected $\boldsymbol{X}^B$, transform with a multilayer network, and obtain the output.

B.3.3. Additional results

We test the hypothesis that using a stochastic process with similar properties to the data will lead to better performance. The difference to the neural process setup in Section 5 is that we fix the synthetic GP to always have $\sigma = 0.05$. As can be seen from Figures 6 and 7, the marginal distribution will be equal regardless of which process and which kernel parameter we use. On the other hand, when we look at path probability $p(\boldsymbol{X})$, we notice better results when the noise process matches data properties (as was also shown in Table 5 and 6). That means, while our model can reverse the process well, the qualitative properties of the sampled curves will be different. In particular, the curves will be *rougher* with increasing γ in OU and *smoother* with increasing σ in GP.

B.4. CSDI imputation

The imputation experiment presented in Sections 4.3 and 5 uses the original CSDI model (Tashiro et al., 2021) and only changes the noise to include the stochastic process source. In this case, the time points at which we evaluate the stochastic process are regular which does not reflect the true nature of the Physionet dataset. Here, we change the setup such that the measurements keep the actual time that has passed instead of rounding to the nearest hour. This is still in favour of the original paper as it only takes one measurement per hour and discards others if they are present. The model from Tashiro et al. (2021) remains the same and we replace the independent normal noise with the GP noise with $\sigma \in \{0.005, 0.01, 0.02\}$.

We run each experimental setup 10 times with different data maskings (see Tashiro et al. (2021) for more details) and report the results in Table 8. We perform the Wilcoxon one-sided signed-rank test (Conover, 1999) and reject the null hypothesis that the expected RMSE values are the same when $p < 0.05$. As we can see, higher values of σ produce better results which makes sense since $\sigma = 0.005$ is, informally, closer to independent Gaussian sampling than $\sigma = 0.02$, which has stronger temporal dependency between the samples. We suspect 10%-missing case does not produce significant results due to noise. Using higher σ does not further improve the results.

Table 8. Imputation results averaged over 10 runs and p-value of Wilcoxon one-sided test.

Missingness:		10%		50%		90%	
Metrics:		RMSE	p-value	RMSE	p-value	RMSE	p-value
CSDI (baseline)		0.603±0.274	–	0.658±0.060	–	0.839±0.043	–
	0.005	0.541±0.085	0.125	0.647±0.049	0.116	0.824±0.032	0.188
$\sigma =$	0.01	0.575±0.195	0.125	0.640±0.050	**0.001**	0.823±0.028	**0.032**
	0.02	0.515±0.039	0.326	0.636±0.050	**0.001**	0.811±0.032	**0.001**

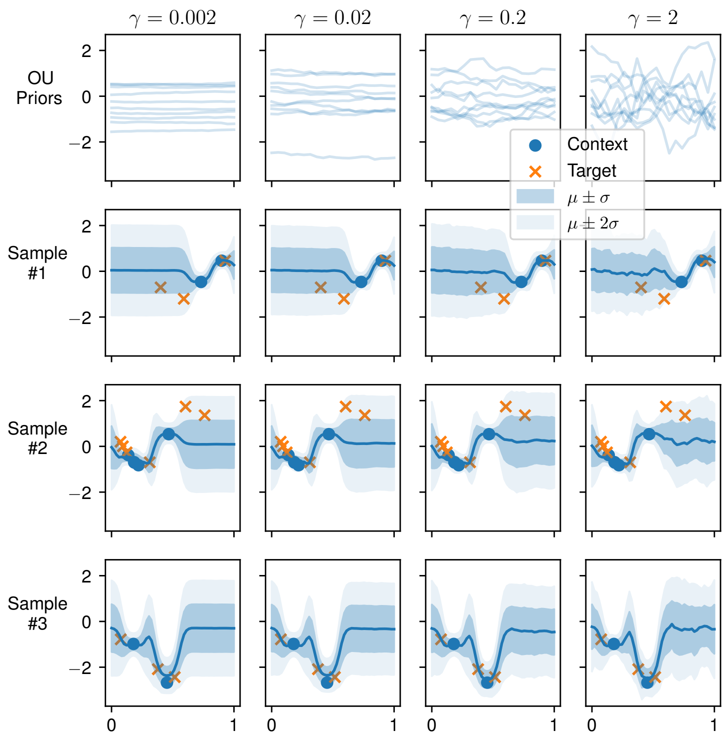

Figure 7. Same setting as Figure 6 but for the Ornstein-Uhlenbeck process. Here, increasing the kernel parameter γ now decreases the smoothness. All of the models perfectly capture the marginal distribution.