SIAM J. SCI. COMPUT.
Vol. 41, No. 4, pp. A2485–A2509

# MACHINE LEARNING OF SPACE-FRACTIONAL DIFFERENTIAL EQUATIONS[*]

MAMIKON GULIAN[†], MAZIAR RAISSI[†], PARIS PERDIKARIS[‡], AND GEORGE KARNIADAKIS[†]

**Abstract.** Data-driven discovery of "hidden physics"—i.e., machine learning of differential equation models underlying observed data—has recently been approached by embedding the discovery problem into a Gaussian process regression of spatial data, treating and discovering unknown equation parameters as hyperparameters of a "physics informed" Gaussian process kernel. This kernel includes the parametrized differential operators applied to a prior covariance kernel. We extend this framework to the data-driven discovery of linear *space-fractional* differential equations. The methodology is compatible with a wide variety of space-fractional operators in $\mathbb{R}^d$ and stationary covariance kernels, including the Matérn class, and allows for optimizing the Matérn parameter during training. Since fractional derivatives are typically not given by closed-form analytic expressions, the main challenges to be addressed are a user-friendly, general way to set up fractional-order derivatives of covariance kernels, together with feasible and robust numerical methods for such implementations. Making use of the simple Fourier-space representation of space-fractional derivatives in $\mathbb{R}^d$, we provide a unified set of integral formulas for the resulting Gaussian process kernels. The shift property of the Fourier transform results in formulas involving $d$-dimensional integrals that can be efficiently treated using generalized Gauss–Laguerre quadrature. The implementation of fractional derivatives has several benefits. First, the method allows for discovering models involving fractional-order PDEs for systems characterized by heavy tails or anomalous diffusion, while bypassing the analytical difficulty of fractional calculus. Data sets exhibiting such features are of increasing prevalence in physical and financial domains. Second, a single fractional-order archetype allows for a derivative term of arbitrary order to be learned, with the order itself being a parameter in the regression. As a result, even when used for discovering integer-order equations, the proposed method has several benefits compared to previous works on data-driven discovery of differential equations; the user is not required to assume a "dictionary" of derivatives of various orders and directly controls the parsimony of the models being discovered. We illustrate our method on several examples, including fractional-order interpolation of advection-diffusion and modeling relative stock performance in the S&P 500 with $\alpha$-stable motion via a fractional diffusion equation.

**Key words.** Gaussian processes, Matérn kernel, fractional diffusion, anomalous diffusion, stable process

**AMS subject classifications.** 35R11, 65N21, 62M10, 62F15, 60G15, 60G52

**DOI.** 10.1137/18M1204991

## 1. Introduction.

A novel use of machine learning, which has potential both for modeling with large or high-frequency data sets as well as advancing fundamental science, is the discovery of governing differential equations from data. Unlike highly specialized algorithms used to refine existing models, these novel methods are distinguished by comparatively limited assumptions and the ability to produce various

[*]Submitted to the journal's Methods and Algorithms for Scientific Computing section August 3, 2018; accepted for publication (in revised form) May 13, 2019; published electronically August 1, 2019.
https://doi.org/10.1137/18M1204991

**Funding:** This work was supported by DARPA EQUiPS grant N66001-15-2-4055, MURI/ARO grant W911NF-15-1-0562, AFOSR grant FA9550-17-1-001, and the NSF Graduate Research Fellowship Program.

[†]Department of Applied Mathematics, Brown University, Providence, RI 02912 (mamikon_gulian@brown.edu, maziar_raissi@brown.edu, George_Karniadakis@brown.edu).
[‡]Department of Mechanical Engineering and Applied Mechanics, School of Engineering and Applied Science, University of Pennsylvania, Philadelphia, PA 19104 (pgp@seas.upenn.edu).

types of equations from a wide variety of data.

We now give a very brief (and incomplete) overview of a few proposed algorithms. Key differences between various works include the techniques used to generate candidate equations and the technique used to select equations. A fundamental problem that all these work address is that, if such algorithms are to mimic a human scientist, while generating accurate models, they must also avoid or penalize spurious "overfitted" models. In [34], a symbolic regression method was developed to learn conservation laws of physical systems. The laws, which could be nonlinear, were created using genetic programming and evaluated concurrently on predicative ability and parsimony (number of terms) in order to prevent overfitting. Earlier, in [2] in a dynamical system context, the equations were evaluated using "probing" tests while the overfitting was addressed using a separate "snipping" process. In [5], a more parametric method for learning nonlinear autonomous systems was developed in which the candidate equation is build from a linear combination of elements from a user-defined "dictionary." Parsimony translated to sparsity in the dictionary elements, so sparse regression was used to determine the coefficients. In [32], a similar approach was used to generate nonlinear partial differential evolution equations.

Building on an earlier work [26] where Gaussian process regression was used to infer solutions of equations, a Gaussian process framework was developed in [27] for parametric learning of linear differential equations, of the form

$$\mathcal{L}^{p_1,\dots,p_k} u = f \tag{1.1}$$

given small data on $f$ and $u$. Here $\mathcal{L}^{p_1,\dots,p_k}$ refers to a linear differential operator with parameters $p_1,\dots,p_k$; for example, $\mathcal{L}^{p_1,\dots,p_k}$ could be a linear combination of $k$ differential operators with coefficients $p_1,\dots,p_k$. The method placed a Gaussian process on the function $u$ and used linearity of $\mathcal{L}^{p_1,\dots,p_k}$ to obtain a joint Gaussian process on $(u, f)$ in which the unknown parameters $p_1,\dots,p_k$ were subsumed as hyperparameters (this is reviewed in detail in section 2). This allowed the use of standard continuous optimization techniques to optimize the negative log-marginal likelihood, which effects an automatic trade-off between data fit and model complexity, and thereby find $p_1,\dots,p_k$. Thus, in this approach, the problem of identifying the differential equation is "embedded" into the problem of interpolation/regression of data. Moreover, the Gaussian process method only requires computation of the forward action of $\mathcal{L}^{\boldsymbol{p}}$ on the covariance kernel, rather than the solution of the differential equation. However, the method also required the user to select a "dictionary" of terms and assume a parametric form of the equation.

The inclusion of fractional-order operators in this framework, where the order itself is a parameter, allows for a single fractional-order operator to interpolate between derivatives of all orders. This allows the user to directly control the parsimony, by fitting a specified number of fractional derivatives to data, without making assumptions on the orders of derivatives to be discovered. Therefore, building on a basic example of a fractional-order operator treated in [27], we significantly extend the framework to treat other space-fractional differential operators and covariance kernels. The main problem that must be addressed is the efficient computation of the action of the unknown (fractional) linear operator $\mathcal{L}^{p_1,\dots,p_k}$ on covariance kernels.

At the same time, fractional-order derivatives are far more than a tool to interpolate between integer-order derivatives and facilitate data-driven discovery of PDEs using continuous optimization techniques. The advances in the present article further improve the ability to discover fractional-order partial differential equations (FPDEs) from real-world data. It is now well understood that FPDEs have a profound connection with anomalous diffusions and systems driven by heavy-tailed processes [18],

[19]. While such heavy-tailed data abounds in the fields of, e.g., hydrology, finance, and plasma physics, FPDEs are currently underutilized as tools to model macroscopic properties of such systems. This can be attributed to the analytic difficulties of deriving FPDE models; not only is there an additional parameter, but the involved formulas and nonlocal nature of fractional-order derivatives make them significantly less attractive for specialists in other fields to work with.

Machine learning is a natural tool for ameliorating this issue. As proof of this concept, we point to the work [21], which employed a multifidelity Gaussian process method to discover the fractional order of the fractional advection-dispersion equation governing underground tritium transport through a heterogeneous medium at the macrodispersion experimental site at Columbus Air Force Base. This resulted in improved and highly efficient fitting of variable-order fractional equation to data. Along more theoretical lines, [10] explored the determination, using Gaussian processes, of the fractional order of an elliptic problem involving the spectral fractional Laplacian on a bounded domain with Neumann boundary condition. In addition, the authors also proved well-posedness of the inverse Bayesian formulation of this problem in the sense of [36]. Our work differs from [21] in that we do not repeatedly solve the forward problem and from [10] in that we do not place a prior on the fractional order or on other parameters; rather, the parameters are inferred by placing a prior and training on the solution and right-hand side (RHS) of the equation. This allows for more flexibility with regard to the form of the equation and the inclusion of additional parameters.

In the aforementioned Gaussian process framework of [27], it is possible to treat time-derivatives by considering the data and equation in space-time. Time may be treated as another dimension in the covariance kernel. An alternative method is suggested by the work of [24], in which learning of evolution equations is based on the numerical differentiation of Gaussian processes. There, the data is given at different "snapshots" in time which are used to discretize the time-derivative. In this way, even nonlinear time-dependent equations can be discovered using correspondence between nonlinear terms and specific linearizations of the discretized system. The training is similar to that of [27], and we extend this to fractional operators as well.

This work is organized as follows. In section 2, we review the Gaussian process framework of [27] and [24]. In section 3, we review the Matérn family of covariance kernels and space-fractional operators and present new formulas for space-fractional derivatives of such covariance kernels. These covariance kernels can be more suited to rough data in certain applications. The inclusion of these new formulas, which can be efficiently treated using generalized Gauss–Laguerre quadrature, allows the discovery of various fractional equations in a unified way. In section 4 we present basic synthetic examples to illustrate the methodology, and in section 5 we apply the methodology to the discovery and interpolation of integer-order advection and diffusion using the fractional-order framework. This includes an interesting problem of interpolating two-term advection-diffusion using a single fractional-order term, and an exploration of user-selected parsimony. After reviewing the relation between $\alpha$-stable processes and fractional diffusion in section 6 and studying a synthetic example, in section 7 we apply the methodology to the modeling of relative stock performance (Intel to S&P 500) by $\alpha$-stable processes via an associated fractional-order diffusion equation.

## 2. The Gaussian process framework.

We review the framework developed by [27] for parametric learning of linear differential equations, of the form

$$\mathcal{L}_{\boldsymbol{x}}^{\boldsymbol{p}} u = \mathcal{L}_{\boldsymbol{x}}^{p_1,\dots,p_k} u = f \tag{2.1}$$

given data on $f$ and $u$. Here, $\mathcal{L}_{\boldsymbol{x}}^{p_1,\dots,p_k}$ is a linear operator with unknown parameters $p_1,\dots p_k$. Throughout the article, we use boldface characters (such as $\boldsymbol{x}$) to denote vector-valued variables and capital boldface characters (such as $\boldsymbol{X}$) to denote data vectors.

Assume $u(\boldsymbol{x})$ to be a Gaussian process with mean 0 and covariance function $k_{uu}$ $(\boldsymbol{x},\boldsymbol{x'};\boldsymbol{\theta})$ with hyperparameters $\boldsymbol{\theta}$:

$$u(\boldsymbol{x}) \sim \mathcal{GP}\left(0, k_{uu}\left(\boldsymbol{x},\boldsymbol{x'};\boldsymbol{\theta}\right)\right). \tag{2.2}$$

We shall be vague about the form of the covariance kernel until section 3; for now, it suffices to say that it describes how the correlation between the values of $u$ at two points $\boldsymbol{x}$ and $\boldsymbol{x'}$ falls off with $|\boldsymbol{x}-\boldsymbol{x'}|$ or otherwise behaves with the two points, and that it must be a symmetric, positive semidefinite function [31]. Then, the linear transformation $f = \mathcal{L}^{\boldsymbol{p}}u$ of the Gaussian process $u$ implies a Gaussian process for $f(\boldsymbol{x})$ (see [31, section 9.4]),

$$f(\boldsymbol{x}) \sim \mathcal{GP}(0, k_{ff}(\boldsymbol{x},\boldsymbol{x'};\boldsymbol{\theta},\boldsymbol{p})), \tag{2.3}$$

with covariance kernel

$$k_{ff}(\boldsymbol{x},\boldsymbol{x'};\boldsymbol{\theta},\boldsymbol{p}) = \mathcal{L}_{\boldsymbol{x}}^{\boldsymbol{p}}\mathcal{L}_{\boldsymbol{x'}}^{\boldsymbol{p}}k_{uu}(\boldsymbol{x},\boldsymbol{x'};\boldsymbol{\theta}). \tag{2.4}$$

Moreover, the covariance between $u(\boldsymbol{x})$ and $f(\boldsymbol{x}')$, and between $f(\boldsymbol{x})$ and $u(\boldsymbol{x}')$ is

$$\begin{aligned} k_{uf}(\boldsymbol{x},\boldsymbol{x'};\boldsymbol{\theta},\boldsymbol{p}) &= \mathcal{L}_{\boldsymbol{x'}}^{\boldsymbol{p}}k_{uu}(\boldsymbol{x},\boldsymbol{x'};\theta),\\ k_{fu}(\boldsymbol{x},\boldsymbol{x'};\boldsymbol{\theta},\boldsymbol{p}) &= \mathcal{L}_{\boldsymbol{x}}^{\boldsymbol{p}}k_{uu}(\boldsymbol{x},\boldsymbol{x'};\theta), \end{aligned} \tag{2.5}$$

respectively. By symmetry of $k_{uu}$,

$$k_{fu}(\boldsymbol{x},\boldsymbol{x'};\boldsymbol{\theta},\boldsymbol{p}) = k_{uf}^T(\boldsymbol{x},\boldsymbol{x'};\boldsymbol{\theta},\boldsymbol{p}) \overset{\text{def}}{=} k_{uf}(\boldsymbol{x'},\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{p}). \tag{2.6}$$

The hyperparameters $(\boldsymbol{\theta},\boldsymbol{p})$ of the joint Gaussian process

$$\begin{bmatrix} u(\boldsymbol{x}) \\ f(\boldsymbol{x}) \end{bmatrix} \sim \mathcal{GP}\left(\boldsymbol{0}, \begin{bmatrix} k_{uu}(\boldsymbol{x},\boldsymbol{x'};\boldsymbol{\theta}) & k_{uf}(\boldsymbol{x},\boldsymbol{x'};\boldsymbol{\theta},\boldsymbol{p}) \\ k_{fu}(\boldsymbol{x},\boldsymbol{x'};\boldsymbol{\theta},\boldsymbol{p}) & k_{ff}(\boldsymbol{x},\boldsymbol{x'};\boldsymbol{\theta},\boldsymbol{p}) \end{bmatrix}\right) \tag{2.7}$$

are then learned by training on the data $\boldsymbol{Y}_u$ of $u$ given at points $\boldsymbol{X}_u$ and $\boldsymbol{Y}_f$ of $f$ given at points $\boldsymbol{X}_f$. This is done with a quasi-Newton optimizer L-BFGS to minimize the negative log marginal likelihood [31]:

$$\begin{aligned} \mathcal{NLML}\left(\boldsymbol{\theta},\boldsymbol{p},\sigma_{n_u}^2,\sigma_{n_f}^2\right) &= -\log p\left(\boldsymbol{Y}|\boldsymbol{\theta},\boldsymbol{p},\sigma_{n_u}^2,\sigma_{n_f}^2\right)\\ &= \frac{1}{2}\boldsymbol{Y}^T\boldsymbol{K}^{-1}\boldsymbol{Y} + \frac{1}{2}\log|\boldsymbol{K}| + \frac{N}{2}\log(2\pi), \end{aligned} \tag{2.8}$$

where $\boldsymbol{Y} = \left[\begin{smallmatrix} \boldsymbol{Y}_u \\ \boldsymbol{Y}_f \end{smallmatrix}\right]$, $p(\boldsymbol{Y}|\boldsymbol{\theta},\boldsymbol{p},\sigma_{n_u}^2,\sigma_{n_f}^2) = \mathcal{N}(\boldsymbol{0},\boldsymbol{K})$, and $\boldsymbol{K}$ is given by

$$\boldsymbol{K} = \begin{bmatrix} k_{uu}(\boldsymbol{X}_u,\boldsymbol{X}_u;\boldsymbol{\theta}) + \sigma_{n_u}^2\boldsymbol{I}_{n_u} & k_{uf}(\boldsymbol{X}_u,\boldsymbol{X}_f;\boldsymbol{\theta},\boldsymbol{p}) \\ k_{fu}(\boldsymbol{X}_f,\boldsymbol{X}_u;\boldsymbol{\theta},\boldsymbol{p}) & k_{ff}(\boldsymbol{X}_f,\boldsymbol{X}_f;\boldsymbol{\theta},\boldsymbol{p}) + \sigma_{n_f}^2\boldsymbol{I}_{n_f} \end{bmatrix}. \tag{2.9}$$

The additional noise parameters $\sigma_{n_u}^2$ and $\sigma_{n_f}^2$ are included to learn uncorrelated noise in the data; their inclusion above corresponds to the assumption that

$$\begin{aligned} \boldsymbol{Y}_u &= u(\boldsymbol{X}_u) + \boldsymbol{\epsilon}_u,\\ \boldsymbol{Y}_f &= f(\boldsymbol{X}_f) + \boldsymbol{\epsilon}_f \end{aligned} \tag{2.10}$$

with $\boldsymbol{\epsilon}_u \sim \mathcal{N}(\boldsymbol{0},\sigma_{n_u}^2\boldsymbol{I}_{n_u})$ and independently $\boldsymbol{\epsilon}_f \sim \mathcal{N}(\boldsymbol{0},\sigma_{n_f}^2\boldsymbol{I}_{n_f})$.

Next we review the time-stepping Gaussian process method of [24] for learning linear (in our case) equations of the form

$$u_t + \mathcal{L}^{p_1,\dots,p_k} u = 0, \;\; x \in \mathbb{R}^d, \;\; t \in [0,T]. \tag{2.11}$$

For our purposes, we consider two "snapshots" $\{\boldsymbol{x}^{n-1}, \boldsymbol{u}^{n-1}\}$ and $\{\boldsymbol{x}^n, \boldsymbol{u}^n\}$ of the system at two times $t^{n-1}$ and $t^n$, respectively, such that

$$t^n - t^{n-1} = \Delta t \ll 1. \tag{2.12}$$

We perform, in the case of two snapshots, a backward Euler discretization

$$u^n + \Delta t \mathcal{L}^{p_1,\dots,p_k} u^n = u^{n-1}. \tag{2.13}$$

Then we assume a Gaussian process, but for $u^n$:

$$u^n(\boldsymbol{x}) \sim \mathcal{GP}(0, k(\boldsymbol{x}, \boldsymbol{x'}, \boldsymbol{\theta})). \tag{2.14}$$

As before, the linearity of $\mathcal{L}^{p_1,\dots,p_k}$ leads to a Gaussian process for $u^{n-1}$. We obtain the joint Gaussian process

$$\begin{bmatrix} u^n(\boldsymbol{x}) \\ u^{n-1}(\boldsymbol{x}) \end{bmatrix} \sim \mathcal{GP}\left(\boldsymbol{0}, \begin{bmatrix} k^{n,n}(\boldsymbol{x}, \boldsymbol{x'}; \boldsymbol{\theta}) & k^{n,n-1}(\boldsymbol{x}, \boldsymbol{x'}; \boldsymbol{\theta}, \boldsymbol{p}) \\ k^{n-1,n}(\boldsymbol{x}, \boldsymbol{x'}; \boldsymbol{\theta}, \boldsymbol{p}) & k^{n-1,n-1}(\boldsymbol{x}, \boldsymbol{x'}; \boldsymbol{\theta}, \boldsymbol{p}) \end{bmatrix}\right), \tag{2.15}$$

where, denoting the identity operator by Id,

$$\begin{aligned} k^{n,n} &= k, & k^{n,n-1} &= (\mathrm{Id} + \Delta t \mathcal{L}^{\boldsymbol{p}}_{\boldsymbol{x'}})k, \\ k^{n-1,n} &= (\mathrm{Id} + \Delta t \mathcal{L}^{\boldsymbol{p}}_{\boldsymbol{x}})k, & k^{n-1,n-1} &= (\mathrm{Id} + \Delta t \mathcal{L}^{\boldsymbol{p}}_{\boldsymbol{x}})(\mathrm{Id} + \Delta t \mathcal{L}^{\boldsymbol{p}}_{\boldsymbol{x'}})k. \end{aligned} \tag{2.16}$$

Equation (2.15) can be compared to (2.7), and (2.16) to (2.4) and (2.5). The setups are very similar, and again, $\boldsymbol{p}$ has been merged into the hypermarameters of this joint Gaussian process. Given data at spatial points $\boldsymbol{X}^n$ and $\boldsymbol{X}^{n-1}$ for the functions $u^n$ and $u^{n-1}$, represented by the vectors $\boldsymbol{U}^n$ and $\boldsymbol{U}^{n-1}$, respectively, the new hyperparameters $(\boldsymbol{\theta}, \boldsymbol{p})$ are trained by employing the same quasi-Newton optimizer L-BFGS as before to minimize the $\mathcal{NLML}$ given by (2.8). In this case, $\boldsymbol{Y} = [\begin{smallmatrix} \boldsymbol{U}^n \\ \boldsymbol{U}^{n-1} \end{smallmatrix}]$, $p(\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{p}, \sigma^2) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{K})$, and $\boldsymbol{K}$ is given by

$$\boldsymbol{K} = \begin{bmatrix} k^{n,n}(\boldsymbol{X}^n, \boldsymbol{X}^n) & k^{n,n-1}(\boldsymbol{X}^n, \boldsymbol{X}^{n-1}) \\ k^{n-1,n}(\boldsymbol{X}^{n-1}, \boldsymbol{X}^n) & k^{n-1,n-1}(\boldsymbol{X}^{n-1}, \boldsymbol{X}^{n-1}) \end{bmatrix} + \sigma_n^2 \boldsymbol{I}. \tag{2.17}$$

Here, $\sigma_n^2$ is an additional parameter to learn noise in the data, under the assumption

$$\begin{aligned} \boldsymbol{u}^n &= u^n(\boldsymbol{X}^n) + \boldsymbol{\epsilon}^n, \\ \boldsymbol{u}^{n-1} &= u^{n-1}(\boldsymbol{X}^{n-1}) + \boldsymbol{\epsilon}^{n-1} \end{aligned} \tag{2.18}$$

with $\boldsymbol{\epsilon}^n \sim \mathcal{N}(0, \sigma_n^2 \boldsymbol{I})$ and $\boldsymbol{\epsilon}^{n-1} \sim \mathcal{N}(0, \sigma_n^2 \boldsymbol{I})$ being independent.

## 3. Fractional derivatives of covariance kernels.

Many properties of a Gaussian process are determined by the choice of covariance kernel $k(\boldsymbol{x}, \boldsymbol{x'})$. In particular, the covariance kernel encodes an assumption about the smoothness of the field that being interpolated. Stein [35] writes "properties of spatial interpolants depend strongly on the local behavior of the random field. In practice, this local behavior is not known and must be estimated from the same data that will be used to do the interpolation. This state of affairs strongly suggests that it is critical to select models for the covariance structures that include at least one member whose local behavior accurately reflects the actual local behavior of the spatially varying quantity under study." Matérn kernels $M_\nu$ (defined below), a family of stationary kernels which includes the exponential kernel for $\nu = 1/2$, and the squared-exponential kernel in the limit $\nu \to \infty$, have been widely used for this reason. A Gaussian process with Matérn covariance kernel $M_\nu$ is $n$-times mean square differentiable for $n < \nu$. We have developed a computational methodology that allows for Matérn kernels of arbitrary real order $\nu > 0$ to be used in our Gaussian process, namely, in (2.2) and (2.14). In fact, the parameter $\nu$ itself can be treated and optimized as a hyperparameter of the Gaussian process, just as the equation parameters were in section 2. We employ such an algorithm to explore the effect of the parameter $\nu$ when working with rough time series histogram data in section 6.

The main problem that arises when using fractional operators is the computation of their action on kernels such as the Matérn class, as required by (2.4) and (2.5) for the time-independent case and (2.16) for the time-dependent case. This cannot be done analytically and requires a numerical approach, in contrast to the works [27] and [24] where (standard) differential operators applied to kernels were obtained symbolically in closed form using Mathematica. Moreover, unlike standard derivatives, fractional derivatives, whether in $\mathbb{R}^d$, $\mathbb{R}^+$ or on bounded subsets, are nonlocal operators typically defined by singular integrals or eigenfunction expansions that are difficult and expensive to discretize [14], [18]. However, space-fractional derivatives in $\mathbb{R}^d$ enjoy representations as Fourier multiplier operators. In other words, they are equivalent to multiplication by a function $m(\boldsymbol{\xi})$ in frequency space. This representation suggests a computational method that avoids any singular integral operators or the solution of extension problems in $\mathbb{R}^{d+1}$. The downside to Fourier methods is that, if used to compute the fractional derivative of a function $u$ on $\mathbb{R}^d$, they may require quadrature of a $2d$ (forward and inverse) Fourier integral. Thus, if one wishes to compute the fractional derivative $\mathcal{L}$ of a covariance kernel $k(\boldsymbol{x}, \boldsymbol{y})$, as in (2.4), (2.5), or (2.16), this may entail $2d$ quadrature for $\mathcal{L}_{\boldsymbol{x}} k$ and $\mathcal{L}_{\boldsymbol{y}} k$ and $4d$ quadrature for $\mathcal{L}_{\boldsymbol{x}}\mathcal{L}_{\boldsymbol{y}} k$. These dimensions for quadrature can be cut in half provided the (forward) Fourier transforms of these kernels were known analytically. Moreover, if the covariance kernel $k_{uu}$ is stationary, we see in Theorem 3.1 below that $\mathcal{L}_{\boldsymbol{x}}\mathcal{L}_{\boldsymbol{y}} k$ can further be reduced from a $2d$-dimensional integral to $d$-dimensional one.

Thus, the entire problem of kernel computation is reduced to $d$-dimensional quadrature if the following three conditions are satisfied:

1. *The spacial differential operator $\mathcal{L}$ is a Fourier multiplier operator*:

$$\mathcal{F}\{\mathcal{L}f\}(\boldsymbol{\xi}) = m(\boldsymbol{\xi}) \cdot \mathcal{F}\{f\}(\boldsymbol{\xi}). \tag{3.1}$$

This is true for a variety of fractional space derivatives:

$$\begin{aligned} \text{Fractional Laplacian}: &\ \mathcal{F}\left\{(-\Delta)^{\alpha/2} f\right\}(\boldsymbol{\xi}) = |\boldsymbol{\xi}|^\alpha \mathcal{F}\{f\}(\boldsymbol{\xi}), \\ \text{Left-sided Riemann–Louiville}: &\ \mathcal{F}\left\{{}^{RL}_{-\infty}D^\alpha_{\boldsymbol{x}} f\right\}(\boldsymbol{\xi}) = (-i\boldsymbol{\xi})^\alpha \mathcal{F}\{f\}(\boldsymbol{\xi}), \\ \text{Right-sided Riemann–Louiville}: &\ \mathcal{F}\left\{{}^{RL}_{\boldsymbol{x}}D^\alpha_{\infty} f\right\}(\boldsymbol{\xi}) = (i\boldsymbol{\xi})^\alpha \mathcal{F}\{f\}(\boldsymbol{\xi}). \end{aligned} \tag{3.2}$$

Here, and throughout this article, we use the Fourier transform convention

$$\mathcal{F}f = \frac{1}{(2\pi)^{d/2}} \int_{\mathbb{R}^d} e^{-i\boldsymbol{\xi}\cdot\boldsymbol{x}} f(\boldsymbol{x}) d\boldsymbol{x}, \quad \mathcal{F}^{-1}\hat{f} = \frac{1}{(2\pi)^{d/2}} \int_{\mathbb{R}^d} e^{i\boldsymbol{\xi}\cdot\boldsymbol{x}} \hat{f}(\boldsymbol{\xi}) d\boldsymbol{\xi}. \tag{3.3}$$

2. *The covariance kernel $k$ is stationary*:

$$k(\boldsymbol{x},\boldsymbol{y}) = K(\boldsymbol{x}-\boldsymbol{y}). \tag{3.4}$$

This is true of the squared-exponential kernel in one dimension

$$G(\sigma,\theta;x,y) = \sigma^2 \exp\left(-\frac{1}{2}\frac{(x-y)^2}{\theta^2}\right) \tag{3.5}$$

as well as the frequently used multivariate squared-exponential kernels formed by multiplication

$$G^{\times}(\sigma,\theta_1,\theta_2,\ldots,\theta_d;\boldsymbol{x},\boldsymbol{y}) = \sigma^2 \prod_{i=1}^{d} \exp\left(-\frac{1}{2}\frac{(x_i-y_i)^2}{\theta_i^2}\right) \tag{3.6}$$

or addition

$$G^{+}(\sigma,\theta_1,\theta_2,\ldots,\theta_d;\boldsymbol{x},\boldsymbol{y}) = \sigma^2 \sum_{i=1}^{d} \exp\left(-\frac{1}{2}\frac{(x_i-y_i)^2}{\theta_i^2}\right) \tag{3.7}$$

of the one-dimensional kernel. The same is true for the Matérn kernels $M_\nu$, which have one-dimensional form

$$M_\nu(\sigma,\theta;x,y) = \frac{\sigma^2 2^{1-\nu}}{\Gamma(\nu)}\left(\frac{\sqrt{2\nu}}{\theta}(x-y)\right)^{\nu} K_\nu\left(\frac{\sqrt{2\nu}}{\theta}(x-y)\right) \tag{3.8}$$

and the corresponding multidimensional kernels

$$\begin{aligned} M^{\times}_{\nu_1,\ldots,\nu_d}(\boldsymbol{x},\boldsymbol{y}) &= \prod_{i=1}^{d} M_{\nu_i}(x_i-y_i),\\ M^{+}_{\nu_1,\ldots,\nu_d}(\boldsymbol{x},\boldsymbol{y}) &= \sum_{i=1}^{d} M_{\nu_i}(x_i-y_i). \end{aligned} \tag{3.9}$$

The notation $K_\nu$ refers to the modified Bessel function, which is potentially confusing but will not be an issue as we will focus on Fourier representation of $M_\nu$ in what follows.

3. *The (forward) Fourier transform $\hat{K}(\boldsymbol{\xi}) = \mathcal{F}\{K\}(\boldsymbol{\xi})$ of the stationary covariance kernel $K$ is known analytically.* This is satisfied by the squared exponential kernel $G(\sigma,\theta;x)$ and the Matérn kernel[1] $M_\nu(\sigma,\theta;x)$:

(3.10)

$$\text{Squared-exponential kernel: } \mathcal{F}\{G\}(\xi) = \theta\sigma^2 e^{-\frac{\theta^2}{2}\xi^2}$$

$$\text{Matérn kernel: } \mathcal{F}\{M_\nu\}(\xi) = \frac{\theta\sigma^2\Gamma(\nu+1/2)}{\sqrt{\nu}\Gamma(\nu)}\left(1+\frac{\theta^2\xi^2}{2\nu}\right)^{-(\nu+1/2)}.$$

The same is true for the multidimensional kernels $G^+$ and $M^+_\nu$ by linearity of the Fourier transform and for $G^\times$ and $M^\times_\nu$ by Fubini's theorem.

[1]In the machine learning literature, authors such as [31] describe this Fourier transform as the *spectral density* in the context of Bochner's theorem on stationary kernels and write it in the equivalent form, up to Fourier transform convention: $\hat{M}_\nu(\xi) = \sigma^2\frac{\sqrt{2}\Gamma(\nu+1/2)(2\nu)^\nu}{\Gamma(\nu)\theta^{2\nu}}(\frac{2\nu}{\theta^2}+\xi^2)^{-(\nu+1/2)}$.

Theorem 3.1. *Suppose conditions* (1)–(3) *on the covariance kernel $k$ and the operator $\mathcal{L}$ and are satisfied. Then the fractional derivatives of the covariance kernel $\mathcal{L}_{\boldsymbol{x}}k$, $\mathcal{L}_{\boldsymbol{y}}k = [\mathcal{L}_{\boldsymbol{x}}k]^T$, and $\mathcal{L}_{\boldsymbol{y}}\mathcal{L}_{\boldsymbol{x}}k$ can be computed by d-dimensional integrals*

$$\left\{\begin{aligned} \mathcal{L}_{\boldsymbol{x}}k &= \frac{1}{\sqrt{2\pi}}\int_{\mathbb{R}^d} e^{i\langle \boldsymbol{x}-\boldsymbol{y},\boldsymbol{\xi}\rangle} m(\boldsymbol{\xi})\hat{K}(\boldsymbol{\xi})d\boldsymbol{\xi}, \\ \mathcal{L}_{\boldsymbol{y}}\mathcal{L}_{\boldsymbol{x}}k &= \frac{1}{\sqrt{2\pi}}\int_{\mathbb{R}^d} e^{i\langle \boldsymbol{x}-\boldsymbol{y},\boldsymbol{\xi}\rangle} m(\boldsymbol{\xi})m(-\boldsymbol{\xi})\hat{K}(\boldsymbol{\xi})d\boldsymbol{\xi}. \end{aligned}\right. \tag{3.11}$$

For a proof of this theorem, see Appendix A.

Provided the integrals (3.11) can be computed numerically at the locations of the data, they can be used to build the kernel matrix $\boldsymbol{K}$ and evaluate the objective function $\mathcal{NLML}$ given by (2.8). In the quasi-Newton L-BFGS method (discussed in section 2) that is used to train the extended hyperparameters $\boldsymbol{\theta} = [\theta_i]$ implicit in $\boldsymbol{K}$, we supply the gradient of the $\mathcal{NLML}$, the components of which are given by

$$\frac{\partial \mathcal{NLML}}{\partial \theta_i} = \frac{1}{2}\mathrm{Tr}\left(\boldsymbol{K}^{-1}\frac{\partial \boldsymbol{K}}{\partial \theta_i}\right) - \frac{1}{2}\boldsymbol{Y}^T\boldsymbol{K}^{-1}\frac{\partial \boldsymbol{K}}{\partial \theta_i}\boldsymbol{K}^{-1}\boldsymbol{Y}. \tag{3.12}$$

See [31, section 5.4]. To obtain $\partial \boldsymbol{K}/\partial\theta_i$, we note in (3.11) that the derivative $\partial/\partial\theta_i$ may be passed into the integrand and through the complex exponential factor. The resulting derivative of the product of the multiplier(s) $m$, which contains the equation parameters, and $\hat{K}$, which contains the original kernel parameters, can be obtained symbolically as a closed-form expression. The same numerical procedure is used to evaluate the resulting integrals as for (3.11). The Hessian is not supplied in closed form and is approximated from evaluations of the gradient.

When training a Gaussian process, it is advantageous to *standardize* the data [40] so that $\boldsymbol{x}, \boldsymbol{y}, \theta \sim \mathcal{O}(1)$ when possible. In the framework discussed here, this reduces the difficulty of computing the kernel functions in (3.11) by restricting the frequency and support of the integrands. When necessary, standardization for the applications considered here can be performed by rescaling the values and positions of the data point by appropriate constants. Once the differential equation is learned for the scaled solution $u_{\text{scaled}} = Au(Bx)$, the true coefficients can be obtained via inverse scaling. This is discussed in detail for an example in section 7. During training, we expect convergence to a local minimum of the $\mathcal{NLML}$, and we have not found the optimal $\boldsymbol{\theta}$ to depend significantly on the initial guess in our examples, but there is no guarantee of this. Uniqueness, stability, and convergence remain important open questions.

Although numerical calculation of the above integrals can be performed using Gauss–Hermite quadrature, this is not optimal as the fractional-order monomial $m(\boldsymbol{\xi})$ is not smooth at the origin. To obtain faster convergence with the number of quadrature points, a superior choice is generalized Gauss–Laguerre quadrature, involving a weight function of the form $x^{\alpha_{\text{gGL}}}e^{-x}$ for $\alpha_{\text{gGL}} > -1$:

$$\int_0^\infty f(x)dx = \int_0^\infty x^{\alpha_{\text{gGL}}}e^{-x}\left[e^x x^{-\alpha_{\text{gGL}}}f(x)\right]dx \approx \sum_i^n w_i e^{x_i} x_i^{-\alpha_{\text{gGL}}} f(x_i). \tag{3.13}$$

Here, $w_i$ are the Gauss–Laguerre weights, and $x_i$ the nodes. In practice, it is essential for $\alpha_{\text{gGL}}$ to match the fractional part of the power of the monomial in the integrand $f$, as the remainder yields a smooth function. We use the Golub–Welsch algorithm to find the nodes but compute the weights by evaluating the generalized Gauss–Laguerre polynomial at these nodes for higher relative accuracy.

We describe two examples and discuss the convergence of the numerical Gauss–Laguerre quadrature in each one. First, consider the left-sided Riemann–Louiville derivative $\mathcal{L}_x = {}^{RL}_{-\infty} D_x^\alpha$ in one dimension, which involves $m(\xi) = (-i\xi)^\alpha$ in the above formulas. Owing to the symmetry of $\hat{K}(\xi)$, one can write

$$ {}^{RL}_{-\infty}D_x^\alpha k = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{i(x-y,\xi)}(-i\xi)^\alpha \hat{K}(\xi) d\xi \tag{3.14}$$

$$ = \frac{1}{\sqrt{2\pi}} \left[ \int_0^\infty e^{i(x-y,\xi)}(-i\xi)^\alpha \hat{K}(\xi) d\xi + \int_{-\infty}^{0} e^{i(x-y,\xi)}(-i\xi)^\alpha \hat{K}(\xi) d\xi \right] \tag{3.15}$$

$$ = \frac{1}{\sqrt{2\pi}} \left[ \int_0^\infty e^{i(x-y,\xi)}(-i\xi)^\alpha \hat{K}(\xi) d\xi + \int_{0}^{\infty} \overline{e^{i(x-y,\xi)}(-i\xi)^\alpha} \hat{K}(\xi) d\xi \right] \tag{3.16}$$

$$ = \frac{2}{\sqrt{2\pi}} \int_0^\infty \mathrm{Re}\left[ e^{i(x-y,\xi)}(-i\xi)^\alpha \hat{K}(\xi) \right] d\xi. \tag{3.17}$$

Similarly,

$$ {}^{RL}_{-\infty}D_y^\alpha \left[ {}^{RL}_{-\infty}D_x^\alpha k \right] = \frac{2}{\sqrt{2\pi}} \int_0^\infty \mathrm{Re}\left[ e^{i(x-y,\xi)}(-i\xi)^\alpha (i\xi)^\alpha \hat{K}(\xi) \right] d\xi. \tag{3.18}$$

These integrals call for generalized Gauss–Laguerre quadrature to be performed with $\alpha_{\mathrm{gGL}} = \alpha$ for ${}^{RL}_{-\infty}D_x^\alpha k$ and $\alpha_{\mathrm{gGL}} = 2\alpha$ for ${}^{RL}_{-\infty}D_y^\alpha[{}^{RL}_{-\infty}D_x^\alpha k]$. Using the Matérn kernel with $\nu = 5/2$ as an example, the convergence of the error with the number of quadrature points is shown in Table 1. The MATLAB `integral` function is used for reference when computing the error. The setup for working with the right-sided Riemann–Louiville derivative, or for the one-dimensional fractional Laplacian, is entirely similar.

Next we consider the fractional Laplacian $\mathcal{L} = (-\Delta)^{\alpha/2}$ in two dimensions. This involves the multiplier $m(\boldsymbol{\xi}) = |\boldsymbol{\xi}|^\alpha = |\xi_1^2 + \xi_2^2|^{\alpha/2}$. We transform the integrals into polar coordinates:

$$ (-\Delta_x)^{\alpha/2} k = \frac{1}{\sqrt{2\pi}} \int_0^{2\pi} \int_0^\infty e^{i\langle \boldsymbol{x}-\boldsymbol{y},(r\cos\theta, r\sin\theta)\rangle} r^\alpha \hat{K}(r\cos\theta, r\sin\theta) r dr d\theta, \tag{3.19}$$

$$ (-\Delta_y)^{\alpha/2}(-\Delta_x)^{\alpha/2} k = \frac{1}{\sqrt{2\pi}} \int_0^{2\pi} \int_0^\infty e^{i\langle \boldsymbol{x}-\boldsymbol{y},(r\cos\theta, r\sin\theta)\rangle} r^{2\alpha} \hat{K}(r\cos\theta, r\sin\theta) r dr d\theta. \tag{3.20}$$

Table 1

*$L^\infty$ error in computing the Matérn kernel $M_{\frac{5}{2}}(x-y)$ and the kernel blocks $\mathcal{L}_x M_{\frac{5}{2}}(x-y)$ and $\mathcal{L}_x\mathcal{L}_y M_{\frac{5}{2}}(x-y)$ on $[-1,1]$ using generalized Gauss–Laguerre quadrature.*

| Function $\left(\mathcal{L}_x = {}^{RL}_{x} D_\infty^{1/2}\right)$ | $\theta^2$ | $L^\infty[-1,1]$-norm of error as function of $(x-y)$ @ number of quadrature points | | | |
|---|---|---|---|---|---|
| | | 8 | 16 | 32 | 64 |
| $M_{\frac{5}{2}}(x-y)$ | 0.1 | 1.190e-02 | 3.468e-04 | 2.917e-06 | 2.876e-07 |
| | 1 | 5.126e-03 | 8.029e-06 | 3.741e-07 | 3.204e-07 |
| | 10 | 8.163e-02 | 1.446e-02 | 2.378e-04 | 4.188e-06 |
| $\mathcal{L}_x M_{\frac{5}{2}}(x-y)$ | 0.1 | 4.752e-02 | 1.991e-03 | 2.822e-05 | 3.315e-06 |
| | 1 | 6.179e-03 | 9.528e-05 | 2.524e-07 | 1.634e-07 |
| | 10 | 6.937e-02 | 9.915e-03 | 3.456e-04 | 2.828e-06 |
| $\mathcal{L}_x\mathcal{L}_y M_{\frac{5}{2}}(x-y)$ | 0.1 | 1.624e-01 | 8.980e-03 | 2.277e-04 | 3.449e-06 |
| | 1 | 1.006e-03 | 1.515e-04 | 9.092e-07 | 9.642e-07 |
| | 10 | 6.571e-02 | 4.774e-03 | 3.370e-04 | 1.217e-06 |

TABLE 2
*$L^\infty$ error in computing the Matérn kernel $M_{\frac{5}{2}}M_{\frac{7}{2}}(\boldsymbol{x}-\boldsymbol{y})$ and the kernel blocks $\mathcal{L}_{\boldsymbol{x}}M_{\frac{5}{2}}M_{\frac{7}{2}}(\boldsymbol{x}-\boldsymbol{y})$ and $\mathcal{L}_{\boldsymbol{x}}\mathcal{L}_{\boldsymbol{y}}M_{\frac{5}{2}}M_{\frac{7}{2}}(\boldsymbol{x}-\boldsymbol{y})$ on the square $[-1,1]^2$. The same correlation parameter $\theta$ is used for both kernels. Quadrature is performed using generalized Gauss–Laguerre quadrature in the polar variable $r$ with 8, 16, 32, and 64 quadrature points, with a fixed number of 64 trapezoid rule quadrature points in $\theta$.*

| Function ($\mathcal{L}_x = (-\Delta_x)^{\alpha/2}$) | $\theta^2$ | $L^\infty[-1,1]^2$-norm of error as function of $(\boldsymbol{x}-\boldsymbol{y})$ @ number of quadrature points | | | |
|---|---|---|---|---|---|
| | | 8 | 16 | 32 | 64 |
| $M_{\frac{5}{2}}M_{\frac{7}{2}}$ | $\frac{1}{10}$ | 5.406e-02 | 6.201e-04 | 8.248e-06 | 9.939e-05 |
| | 1 | 1.368e-02 | 3.611e-04 | 2.659e-06 | 2.955e-06 |
| | 10 | 7.955e-01 | 8.385e-02 | 3.873e-03 | 2.006e-05 |
| $\mathcal{L}_x\left[M_{\frac{5}{2}}M_{\frac{7}{2}}\right]$ | $\frac{1}{10}$ | 2.025e-01 | 3.922e-03 | 6.705e-05 | 3.842e-05 |
| | 1 | 2.769e-02 | 9.611e-04 | 8.921e-06 | 8.977e-06 |
| | 10 | 3.717e-01 | 2.013e-02 | 3.340e-03 | 1.515e-05 |
| $\mathcal{L}_x\mathcal{L}_y\left[M_{\frac{5}{2}}M_{\frac{7}{2}}\right]$ | $\frac{1}{10}$ | 4.759e-01 | 2.644e-02 | 5.154e-04 | 7.347e-05 |
| | 1 | 8.117e-02 | 1.449e-03 | 8.740e-06 | 9.848e-06 |
| | 10 | 8.713e-02 | 4.553e-02 | 1.535e-03 | 8.209e-06 |

The quadrature of these integrals is performed using the trapezoid rule in $\theta$ and generalized Gauss–Laguerre quadrature in $r$, using $\alpha_{\mathrm{gGL}} = \alpha + 1$ for $(-\Delta_x)^{\alpha/2}k$ and $\alpha_{\mathrm{gGL}} = 2\alpha + 1$ for $(-\Delta_y)^{\alpha/2}(-\Delta_x)^{\alpha/2}k$. Using the product multivariate Matérn kernel $M_{\frac{5}{2}}M_{\frac{7}{2}}$ as an example, convergence with respect to the number of quadrature points in $r$ is show in Table 2, where 64 quadrature points in $\theta$ and 8, 16, 32, and 64 quadrature points in $r$ are used. Reference answers were generated by nesting the MATLAB `integral` function.

The examples in the following sections are built by defining the Matérn kernels in Mathematica and performing derivatives with respect to the hyperparameters symbolically. The expressions are ported into MATLAB code where Gauss–Laguerre quadrature with appropriate $\alpha_{\mathrm{gGL}}$ is used to compute the $\mathcal{NLML}$ (2.8) as well as the derivatives of the $\mathcal{NLML}$ with respect to the parameters. For our purposes, 64 quadrature points in one dimension and 64x64 quadrature points (as described above) in two dimensions are sufficient.

## 4. A basic example.

In this section, we will illustrate the methodology to discover the parameters $C$ and $\alpha$ in the fractional elliptic equation

$$C(-\Delta)^{\alpha/2}u = f \tag{4.1}$$

in one and two space dimensions from data on $u$ and $f$.

Following the time-independent framework, this means that we must optimize the negative log marginal likelihood (2.8), where in the covariance matrix (2.9), the kernels are given by the integral formulas (3.11). In the latter formulas, the multiplier $m$ and the Fourier transform $\hat{K}$ of the stationary prior kernel must be specified; the multiplier $m$ corresponding to the operator $C(-\Delta)^{\alpha/2}$ is $m(\xi) = C|\xi|^\alpha$ in one dimension and $m(\xi_1, \xi_2) = C|\xi_1^2 + \xi_2^2|^{\alpha/2}$ in two dimensions. We choose to use the Matérn kernel $k_{uu} = K = M_\nu$, given by (3.8) in one dimension, with a tensor product (3.9) of Matérn kernels $M^\times_{\nu_1,\nu_2}$ in two dimensions. Thus, by (3.10), $\hat{K} =$

$\hat{M}_\nu(\xi)$ in one dimension and $\hat{K} = \hat{M}_{\nu_1}(\xi_1)\hat{M}_{\nu_1}(\xi_2)$ in two dimensions. This completes the description of the covariance kernel.

In the one-dimensional case the solution/RHS pair

$$u = e^{-x^2}, \quad f = \frac{C}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{ix\xi} |\xi|^\alpha \frac{e^{-\xi^2/4}}{\sqrt{2}} d\xi \tag{4.2}$$

is used. Two data sets are generated for two experiments. In both experiments, we use the above solution/RHS pair to generate data with exact $C = 1.25, \alpha = \sqrt{2} \approx 1.4142$. The Matérn kernel with fixed $\nu = 11/2$ is used to discover the parameters, and the initial parameters for the optimization are taken to be $\theta = 1, \sigma = 1, \alpha = 1.0, C = 0.5$. In the first experiment, 7 data points at $\boldsymbol{X}_u$ for $u$ and 11 data points $\boldsymbol{X}_f$ for $f$ are generated via latin hypercube sampling. No noise is added. In the second experiment, 20 data points for each of $u$ and $f$ are generated in the same way, but normal random noise of standard deviation 0.1 for $u$ and 0.2 for $f$ is added to the data.

The Gaussian process regression for the first experiment is shown in Figure 1. The equation parameters recovered are $\alpha = 1.40158$ and $C = 1.25840$, which are within 1% of the true values. The Gaussian process regression for the second experiment is shown in Figure 2. There, we have also plotted twice the standard deviation of the Gaussian process plus twice the standard deviation of the learned noise, $2\sigma_{n_u}$ and $2\sigma_{n_f}$ (in the previous experiment, these learned parameters were miniscule). The parameters learned in the second experiment are $\alpha = 1.51151$ and $C = 1.18099$, which are within 7% and 6% of the true values, respectively.

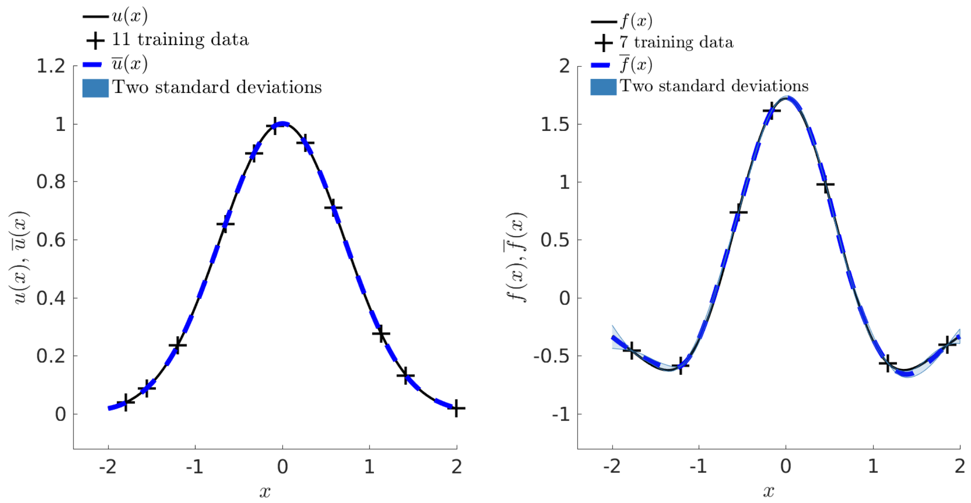

Fig. 1. *Result of training the Gaussian process in the one-dimensional example on* 7 *noise-free data points for* $u$ *(left) and* 11 *data points for* $f$ *(right) on* $[-2, 2]$*. The solid black line represents the reference* $u$ *(resp.,* $f$*), and the crosses mark the data used for training (which can be located at arbitrary positions). The dashed blue line represents the trained posterior mean* $\bar{u}$ *(resp.,* $\bar{f}$*) of the Gaussian process, and the light blue area represents* $\bar{u}$ *(resp.,* $\bar{f}$*)* $\pm$ *twice the standard deviation of the Gaussian process (on the left, this is so small as to not be visible). The trained parameters are within* 1% *of the true values. Optimization wall time:* 7 *minutes. Roughly* 1,400 *objective function evaluations. All reported wall times in this article are obtained using four threads (default MATLAB vectorization) on an Intel i*7*-*6700*K at stock frequency.*

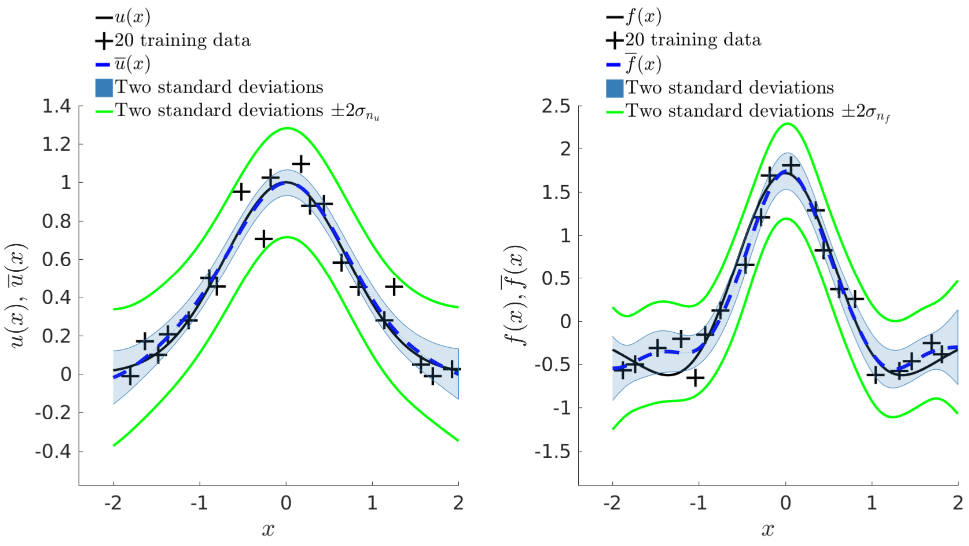

FIG. 2. *Result of training the Gaussian process on* 20 *noisy data points for each of $u$ and $f$ on* $[-2, 2]$. *The trained parameters are within* 7% *of the true values. For this example, we have also plotted in green two standard deviations of the Gaussian process,* plus *two times the learned noise parameter $\sigma_{n_u}/\sigma_{n_f}$. Optimization wall time:* 38 *seconds. Roughly* 100 *function evaluations.*

In the two-dimensional example, the exact solution pair used to generate data is now

$$u = e^{-x_1^2 - x_2^2}, \quad f = \frac{C}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{ix_1\xi_1} e^{ix_2\xi_2} |\xi_1^2 + \xi_2^2|^{\alpha/2} \frac{e^{-\xi_1^2/4 - \xi_2^2/4}}{2} d\xi_1 d\xi_2 \tag{4.3}$$

with exact $C = 1, \alpha = \sqrt{3} \approx 1.7321$. Again, the Matérn parameter $\nu = \nu_1 = \nu_2 = 11/2$ is used. The initial parameters for the optimization are $\sigma = 1, \theta_1 = 1, \theta_2 = 1, \alpha = 2, C = 1.5$. We take 40 data points for each of $u$ and $f$, generated using latin hypercube sampling on $[-2, 2] \times [-2, 2]$. The parameters $\alpha = \sqrt{3} \approx 1.7321$ and $C = 1$ were used to generate data. The result of the training is shown in Figure 3. The learned parameters are $\alpha = 1.72888$ and $C = 0.99977$, within 1% of the true values, with hyperparameters $\sigma = 0.0213$, $\theta_1 = 1.581$, and $\theta_2 = 1.586$.

These examples demonstrate the feasiblity of implementing the fractional kernels as described in the previous sections and the accuracy of discovered parameters, even with noise or small data. Moreover, there is no theoretical difficulty in increasing the dimension of the problem. However, in addition to longer runtime for the computation of formulas (3.11), the user should expect significantly more data to be required for accurate parameter estimation. For example, performing the same two-dimensional example above, with only 20 data points for each of $u$ and $f$, results in learned parameters $\alpha = 1.76516$ and $C = 0.81762$, the $C$ parameter exhibiting an error of roughly 20%. In the analogous one-dimensional example (Figure 1), a 1% error is obtained for less than half this number of data points.

In concluding this section, we point that since the estimation of the equation parameters is based on accurate Gaussian process regression through the training data, there are various well-known hazards to avoid. In addition to obvious issues such as unresolved data and data that is too noisy, too much data in featureless "flat" regions carries the risk of overfitting and should be avoided.

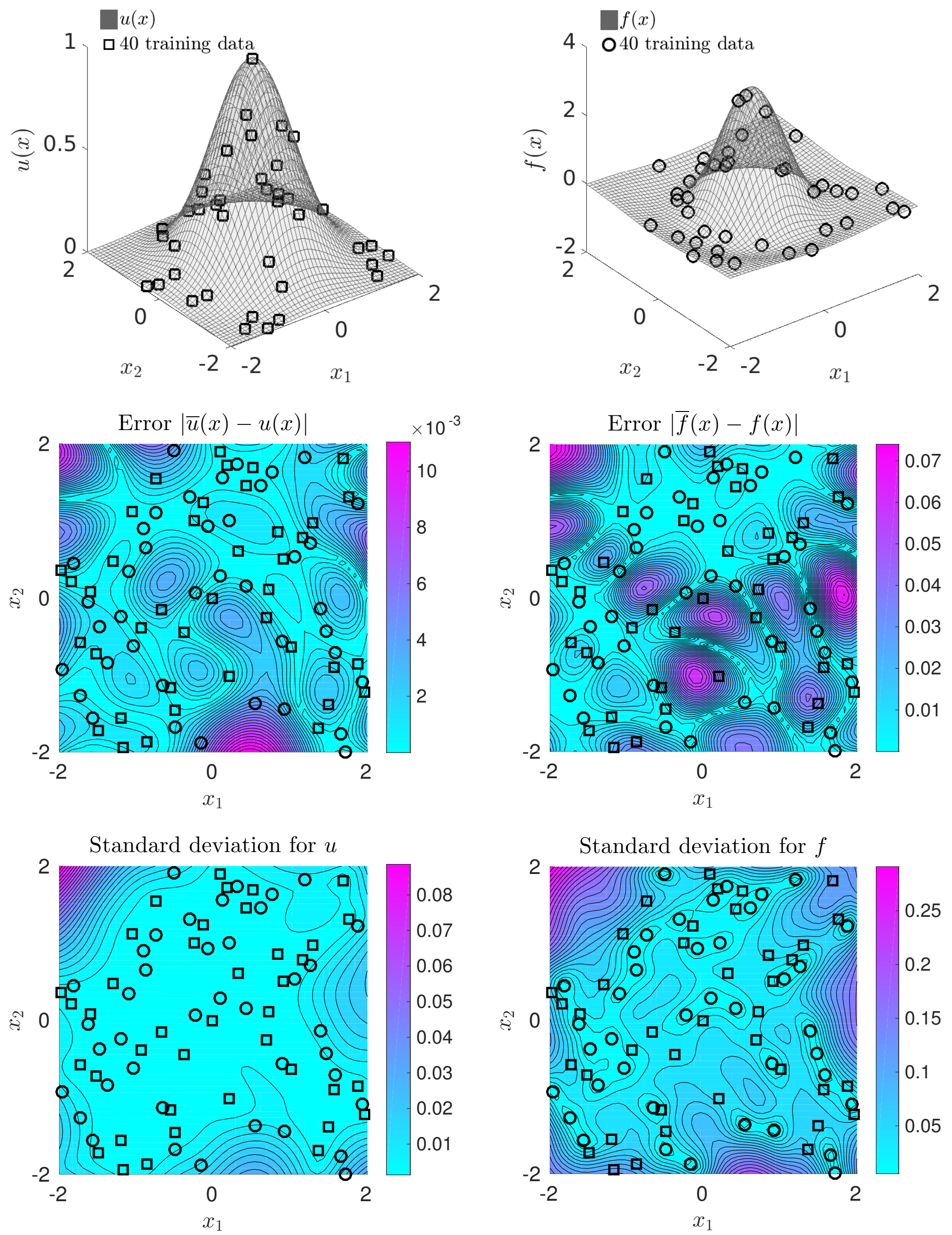

FIG. 3. *Result of training in the two-dimensional example.* Top: *Distribution of* 40 *data points on each of $u$, left, and $f$, right.* Middle: *Error between mean of the trained Gaussian process for $u$, left, and $f$, right, and the exact $u, f$ used to generate data.* Bottom*: Standard deviation of the Gaussian process for $u$, left, and $f$, right. Note that the positions of data points for $u$ are illustrated by squares, while data points for $f$ are illustrated by circles. Optimization wall time:* 74 *minutes. Roughly* 1,400 *objective function evaluations.*

## 5. Discovering and interpolating integer-order models.

As discussed in the introduction, fractional-order differential operators interpolate between classical differential operators of integer-order, reducing the task of choosing a "dictionary" of operators of various orders and the assumptions that this entails. The user controls the parsimony directly, by choosing the number of fractional terms in the candidate equation. For example, the model in section 4 was constrained to be of parsimony one, since it includes a single differential operator of fractional order. This raises several questions, such as, Can the method be used to discover integer-order operators when they drive the true dynamics? What can be expected if the user-selected parsimony is lower than the parsimony of the "true" model driving the data? Can lower-parsimony models still be used to model dynamics? To explore these questions, we consider the parametrized model

$$\frac{\partial u}{\partial t} - C\left({}_{-\infty}^{RL}D_x^{\alpha}\right)u = 0 \tag{5.1}$$

for $u(t,x), t \in \mathbb{R}^+, x \in \mathbb{R}$, where the left-sided Riemann–Louiville derivative ${}_{-\infty}^{RL}D_x^{\alpha}u$ was defined in (3.2). Note that

$$\alpha = 1 \implies {}_{-\infty}^{RL}D_x^{\alpha}u = -\partial u/\partial x, \quad \alpha = 2 \implies {}_{-\infty}^{RL}D_x^{\alpha}u = \partial^2 u/\partial x^2. \tag{5.2}$$

For $\alpha = 1, C = 1$, (5.1) reduces to the advection equation (with speed $C = 1$)

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0, \tag{5.3}$$

while for $\alpha = 2, C = 1$, it reduces to the diffusion equation (with diffusion coefficient $k = 1$)

$$\frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} = 0. \tag{5.4}$$

We perform four experiments. Importantly, we learn these equations using the time-stepping methodology, optimizing (2.8), where the kernel blocks are given by (2.16) and (2.17). We take $k = M_{\frac{19}{2}}$ (effectively a squared-exponential kernel) and again use (3.11) with generalized Gauss–Laguerre quadrature to evaluate the action of the fractional derivatives on $k$. In all of our experiments, we generate data that satisfies the initial condition $u_0 = \sin(x)$. We choose $\Delta t = 0.1$ and $n = 3$; thus, $u^n = u(t = 0.3, x), u^{n-1} = u(t = 0.2, x)$. The Gaussian process is trained on 30 data points for each of these time slices. The experiments are as follows:

1. Data generated from $u(t,x) = \sin(x - t)$, the solution to the advection equation (5.3). We learn the order $\alpha$ and coefficient $C$ in (5.1); the exact $\alpha = 1$.
2. Data generated from $u(t,x) = e^{-t}\sin(x)$, the solution to the heat equation (5.4). We learn the order $\alpha$ and coefficient $C$ in (5.1); the exact $\alpha = 2$.
3. Data generated from $u(t,x) = e^{-t}\sin(x - t)$, the solution to the integer-order advection-diffusion equation

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} - \frac{\partial^2 u}{\partial x^2} = 0. \tag{5.5}$$

   We learn the order $\alpha$ and coefficient $C$ in (5.1). Note that this archetype is limited to only one space-differential operator; we will see that the algorithm will select best order $1 < \alpha < 2$ to capture both the advection and the diffusion in the data.
4. The same advection-diffusion data as in experiment 3, but with the two-term, four-parameter candidate equation

$$\frac{\partial u}{\partial t} - C_1\left({}_{-\infty}^{RL}D_x^{\alpha_1}\right)u - C_2\left({}_{-\infty}^{RL}D_x^{\alpha_2}\right)u = 0. \tag{5.6}$$

TABLE 3
*Results of the four experiments. Wall times: roughly* 11/14/13/21 *minutes.*

| Exp. | Data | Candidate | Parameters learned |
|---|---|---|---|
| 1 | Advection: $\frac{\partial u}{\partial t}+\frac{\partial u}{\partial x}=0$ | $\frac{\partial u}{\partial t}-C\left({}^{RL}_{-\infty}D^{\alpha}_{x}u\right)=0$ | $C=1.01$ $\alpha=0.97$ |
| 2 | Diffusion: $\frac{\partial u}{\partial t}-\frac{\partial^2 u}{\partial x^2}=0$ | $\frac{\partial u}{\partial t}-C\left({}^{RL}_{-\infty}D^{\alpha}_{x}u\right)=0$ | $C=1.05$ $\alpha=2.00$ |
| 3 | Advection-diffusion: $\frac{\partial u}{\partial t}+\frac{\partial u}{\partial x}-\frac{\partial^2 u}{\partial x^2}=0$ | $\frac{\partial u}{\partial t}-C\left({}^{RL}_{-\infty}D^{\alpha}_{x}\right)u=0$ | $C=1.49$ $\alpha=1.47$ |
| 4 | Advection-diffusion: $\frac{\partial u}{\partial t}+\frac{\partial u}{\partial x}-\frac{\partial^2 u}{\partial x^2}=0$ | $\frac{\partial u}{\partial t}-C_1\left({}^{RL}_{-\infty}D^{\alpha_1}_{x}\right)u-C_2\left({}^{RL}_{-\infty}D^{\alpha_2}_{x}\right)u=0$ | $C_1=1.05$ $\alpha_1=0.98$ $C_2=1.03$ $\alpha_2=1.96$ |

We note that all of these experiments call for only a *single* fractional-order dictionary term; even Experiment 4 uses two copies of the same archetype. Experiments 1–3 use initial parameters $\alpha=0.5$ and $C=1.25$, and Experiment 4 uses $\alpha_1=0.5$, $\alpha_2=1.5, C_1=1.25, C_2=0.75$.

In Experiments 1 and 2, we note that fractional-order parameters are discovered close to (within 5% of) the true integer-order parameters. In this sense, the true dynamics can be considered recovered. The numerical difference from the true parameters despite the high number of data points (30 per slice) is likely due to approximation error from the backward Euler approximation (2.13) and may be resolved by using higher-order differentiation with more time slices [25], as simply taking $\Delta t$ to be extremely small may cause the optimization to be dominated by numerical error in computing the kernels.

Experiment 3 shows what occurs when the user-defined parsimony is less than the true dynamics used to generate data. The optimizer still converges, and to a sensible answer—the result can be interpreted as an interpolation of the two integer-order operators in the true dynamics. Moreover, as shown in Figure 4, near the time of the data used to train the model, and even much later, the fractional dynamics are a good approximation to the true dynamics, while being simpler in the sense of being driven by only one spatial derivative. This leads to potential applications of interpolating complex systems using lower-parsimony fractional models.

In experiment 4, the parsimony was increased with the inclusion of an additional indepedent copy of the fractional archetype. The advection-diffusion equation is recovered with parameters within 5% of the true values. Thus, with a single fractional archetype and user-controlled parsimony, it is possible to discover advection, diffusion, advection-diffusion, as well as a single-term fractional interpolation of advection-diffusion.

## 6. Learning fractional diffusion from $\alpha$-stable time series.

We now consider an example that will set up our application to financial time series data and explore the effect of the Matérn parameter $\nu$. The example will involve identification of $\alpha$-stable Lévy processes parameters from time series data. Equivalently, the same data is used to identify a fractional-order diffusion equation governing the transition density of the process, of the form

$$\frac{\partial u}{\partial t}=\frac{\gamma^{\alpha}}{|\cos(\pi\alpha/2)|}\left[p\left({}^{RL}_{-\infty}D^{\alpha}_{x}u\right)+(1-p)\left({}^{RL}_{x}D^{\alpha}_{\infty}u\right)\right],\quad t>0, \tag{6.1}$$

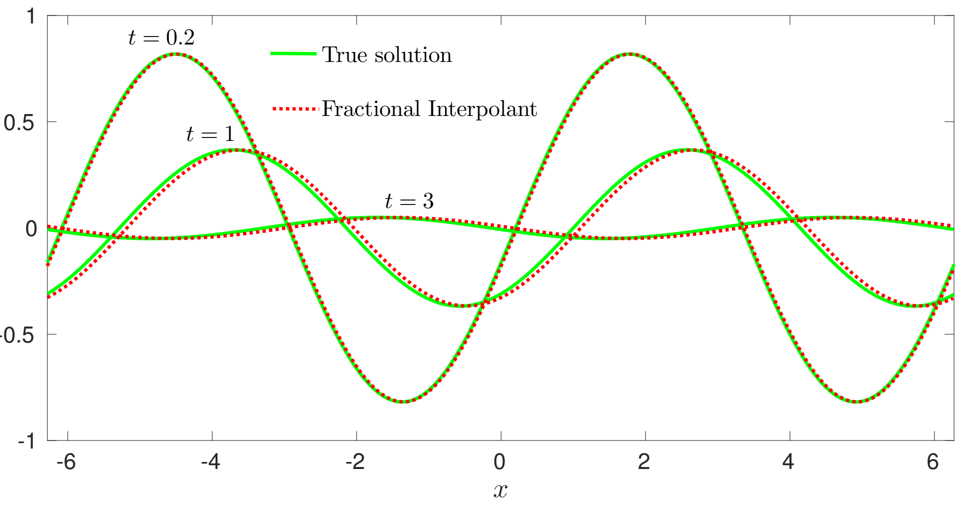

FIG. 4. *Comparison of the true advection-diffusion dynamics with the fractional-order dynamics learned in Experiment 3. Near the time ($t = 0.2$) when the equation was learned, the fractional dynamics are a good approximation, although for later time $t$ the dynamics are increasingly out of phase.*

where

$$0 < \gamma, \quad 0 < p < 1, \quad 0 < \alpha < 2. \tag{6.2}$$

The exact solution to this equation (technically, with initial condition a point distribution at zero) is in fact the $\alpha$-stable probability density

$$S_\alpha(2p - 1, \gamma t^{1/\alpha}, 0) \tag{6.3}$$

with stability parameter $\alpha$, skewness parameter $2p - 1$, scale parameter $\gamma t^{1/\alpha}$, and position parameter 0. See Proposition 5.8 in Meerschaert and Sikorskii [18].

Under the ansatz that the increments are drawn from a transition density, a time series $X_i$, $i = 1, \ldots, i_{\max}$, can be used to recover the transition densities in the following way. Suppose that the increments occur in units of time $\Delta t$. To approximate the transition density $\rho_{n\Delta t}$ at times $n\Delta t$, $n \in \mathbb{N}$, first the collection of increments

$$\left\{X_{(i+n)\Delta t} - X_{i\Delta t}\right\}_{i=1,2,\ldots,i_{\max}-k} \tag{6.4}$$

is assembled. A histogram of such increments is created and normalized; the result is an empirical probability distribution, and as the number of samples increases, the empirical distribution converges to the probability density function [38]. Therefore, if $X_t$ is a time series in which, at each time increment $\Delta t$, a space increment is drawn from the $\alpha$-stable density with parameters skewness $2p-1$, scale $\gamma(\Delta t)^{1/\alpha}$, and position 0, then these empirical histograms $\rho_{n\Delta t}$ will approximate the same density $S_\alpha(2p - 1, \gamma(n\Delta t)^{1/\alpha}, 0)$. In other words, $\rho_{n\Delta t}$ will approximate the time slices $u(t = n\Delta t, x)$ of the solution to (6.1). This setup is shown in Figure 5, which illustrates empirical distributions using 1,200 and 120,000 samples.

In preparation for the next section, we will use the much noiser data generated from an $\alpha$-stable time series of 1,200 steps and parameters $\alpha = \sqrt{2}, p = 0.8$, $\gamma = 1$, and $\Delta t = 0.01$, at times $t = 0.03$ and $t = 0.04$. This is the data shown in the bottom

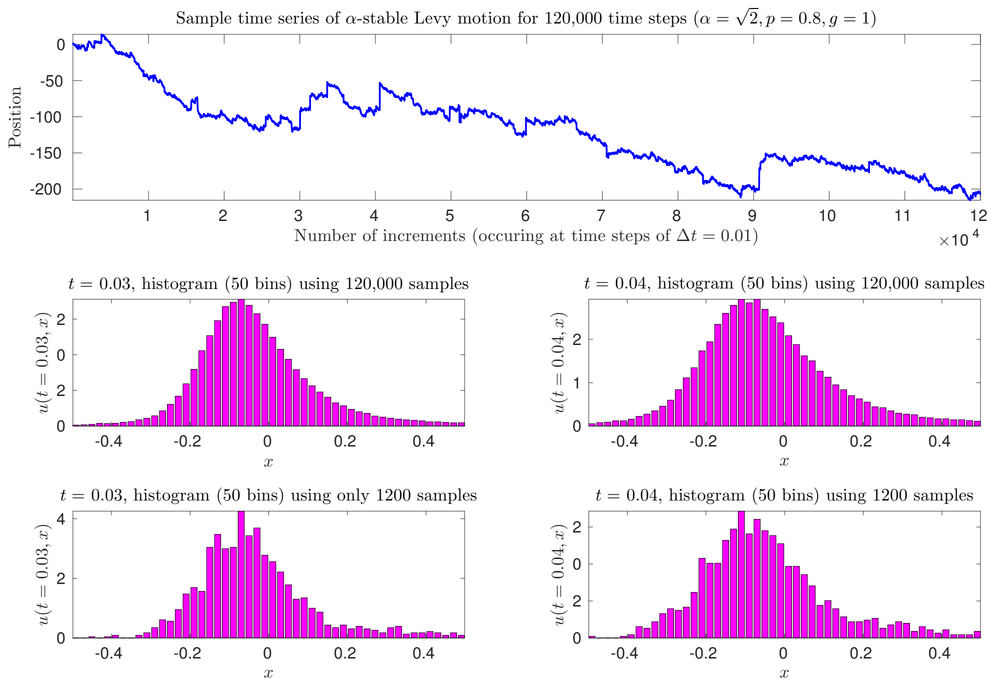

FIG. 5. *Illustration of how data is produced to discover a fractional-order diffusion equation using time series data. The top shows an example $\alpha$-stable time series with $\alpha = \sqrt{2}, p = 0.8$, $\gamma = 1$ at time increments of $\Delta t = 0.01$. To approximate the solution at $t = 0.03$, as on the left, a histogram is made of the spatial increments in three units of time along the time series and normalized. The longer the time series, the more accurate the empirical density will approximate the true density/solution. The middle row shows histograms made with a time series of* 120,000 *increments, while the bottom shows histograms made with a time series of* 1,200 *increments.*

panel of Figure 5. Because we are only seeking an equation that can be associated with an $\alpha$-stable process, we enforce $0 \leq p \leq 1$ and $0 < \alpha \leq 2$ by placing these variables in sigmoids:

$$p = \frac{1}{1 + \exp[-\tilde{p}]}, \quad \alpha = \frac{2}{1 + \exp[-\tilde{\alpha}]}. \tag{6.5}$$

The optimization is then over $\tilde{p}$ and $\tilde{\alpha}$. The parameters are initialized as $\alpha = 1.8$, $p = 0.5$, and $\gamma = 1$. Because the data appears quite rough, we have also allowed for variable order $\nu$, optimizing it as an extra hyperparameter, initializing with $\nu = 9/2$.

The result of the training is shown in Figure 6. The trained parameters are $\alpha = 1.38415, p = 0.87519, \gamma = 0.89166, \sigma = 0.05576, \theta = 0.16287$, and $\nu = 14.82891$. First, we note that the equation parameters are roughly within 10% of the values used to generate the data, despite the low number of samples and roughness of the histogram. Next, the learned value of $\nu$, which is much higher than the starting value of 4.5, shows that the Gaussian process was able to learn the noise parameter $\sigma$ and write off the roughness of the histogram as not intrinsic to the data, so that a low $\nu$ value was not required. As higher Matérn kernel $M_\nu$ is for practical purposes the same as the squared-exponential kernel and with each other [31], we conclude that for such data it is not necessary to utilize a variable-order Matérn kernel, as it prolongs the optimization needlessly. Indeed, performing the same simulation with $\nu$ fixed as 9/2 yields parameters $\alpha = 1.32148, p = 0.80294, \gamma = 0.98273$, which are arguably slightly

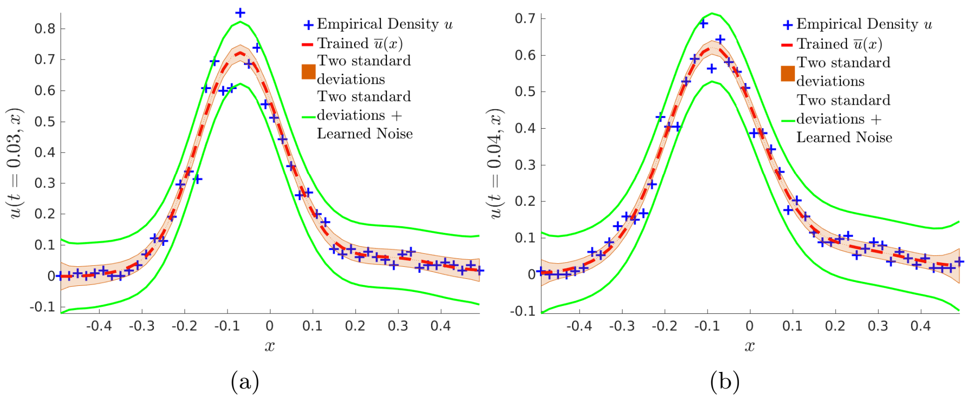

FIG. 6. *Result of training the Gaussian process on the synthetic* 1,200*-step* $\alpha$*-stable time series data. The blue crosses are the normalized histogram/empirical distribution function data. The red curve/orange bars show the mean/two standard deviations of the trained Gaussian process, while the green curve shows two standard deviations* $\pm$ *the learned noise* $2\sigma_n$*.*

closer to the true values. However, we do not claim this to be true for all applications; in particular, in cases where enough data is available to resolve local behavior clearly, variable-order $\nu$ optimization may be critical in training the Gaussian process [35].

**7. Fractional diffusion for relative stock performance.** Many types of Lévy processes—such as $\alpha$-stable processes—are well known and heavily used in financial modeling due to their heavy tails [37, 11, 4]. This began with the work of Mandelbrot in 1963 [16], who showed that returns of cotton prices are more accurately modeled by an $\alpha$-stable density with $\alpha = 1.7$ than with a normal density, followed by the work of Fama in 1965 [8], arriving at a similar conclusion for daily returns of the Dow Jones Industrial Average. More recent investigations include $\alpha$-stable behavior in Mexican financial markets [1], as well as financial modeling by more general Lévy processes [17, 9]. The implications of heavy-tailed behavior in financial processes cannot be underestimated in practice; Wilmott [39] gives the following example based on daily data of the S&P 500 from 1980 to 2004. While a 20% fall in the S&P 500 occured once in this interval of 24 years (the stock market crash of October 19, 1987) a normal distribution for S&P 500 returns (based on an average volatility of 16.9%) would imply such an event would occur only once in every $2 \times 10^{76}$ years.

A number of methods have been used to determine the parameters of an $\alpha$-distribution from empirical data (see Chapter 7 of [37] for a summary, as well as [33]). A naive approach would be to plot the empirical densities in log-log scale, where the power law tail would appear linear, and read the stability parameter $\alpha$ from the slope the curve for large argument. This can be misleading because it is not clear for what arguments an $\alpha$-stable distribution is converged to a power law, nor is the answer simple; it depends on the parameter $\alpha$, and the density enters into a transitory power law decay with power $> 2$ before settling into the "true" power law decay with $\alpha < 2$ (see [3]). Much more reliable methods include maximum likelihood estimation [20] and the generalized method of moments [7], which have had good success in practice.

We propose using machine learning of fractional diffusion equations outlined in section 6, with empirical distributions as data, to calibrate the parameters of the corresponding $\alpha$-stable Lévy process. We have demonstrated that such a method can reliably recover the parameters of synthetic data, and can handle very noisy or rough

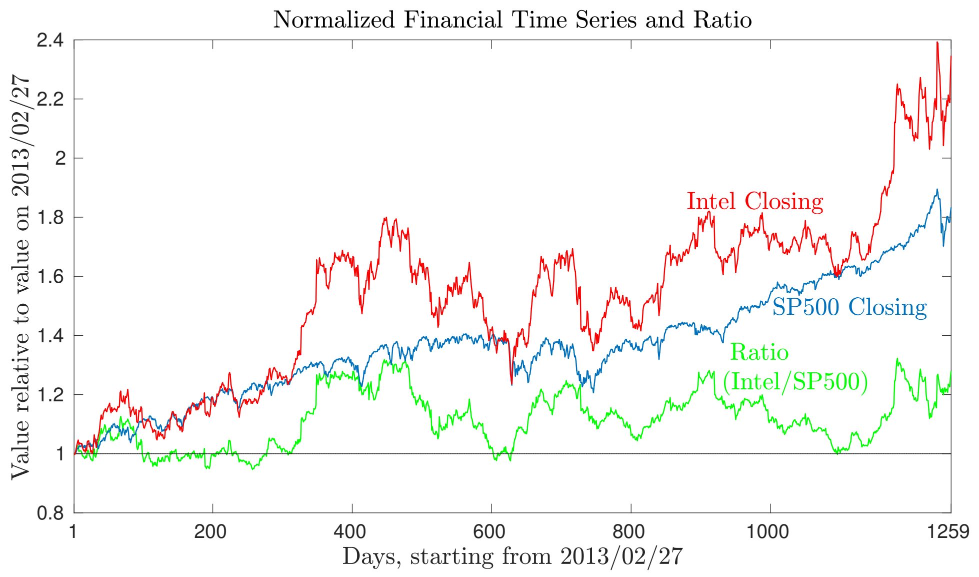

FIG. 7. *The financial time series data used to calibrate the model. The blue curve shows the daily closing values of the S&P* 500*, normalized to the value on February* 27*,* 2013*, while the red curve shows the same for Intel. The ratio (which measures the performance of Intel relative to the S&P* 500 *index in the same time period) is shown in green.*

data in section 6. The following simulations use the exact same setup, only with empirical (rather than synthetic) data, to learn the parameters of the equation:

$$\frac{\partial u}{\partial t} = \frac{\gamma^\alpha}{|\cos(\pi\alpha/2)|}\left[p\left({}^{RL}_{-\infty}D^\alpha_x u\right) + (1-p)\left({}^{RL}_{x}D^\alpha_\infty u\right)\right], \quad t>0. \tag{7.1}$$

The raw data we take for illustration is the relative stock performance of Intel versus S&P 500. We use the daily closing values of each stock in a 5-year period from February 27, 2013, to February 26, 2018. We normalize each stock to the "initial" value on February 27, 2013, then take ratio of the two stocks (Intel/SP500) to yield the time series that will be trained on. The time series used in the procedure are illustrated in Figure 7.

The natural time intervals here are $\Delta t = 1$ days. Thus, in (2.13) and in the kernels, $\Delta t = 1$, and $t = n$, where $n \in \mathbb{N}$. The parameter $\gamma$ can, in principle, capture any space-time scaling relation for the diffusion. However, very small or very large values of $\gamma$ cause optimization instability. Thus, the time series values are rescaled by a constant factor factor $\sqrt{i_{\max}}$—the scaling for Brownian motion—prior to fine tuning with $\gamma$:

$$X_i^{\text{scaled}} = \sqrt{i_{\max}}X_i = \sqrt{1259}X_i. \tag{7.2}$$

This ensures $\gamma = \mathcal{O}(1)$, because it places $X_i$ in roughly the same window as a standard 2-stable process (Brownian motion) with the same number of steps. After this preprocessing, a value of $\gamma^{\text{scaled}}$ will be learned for the scaled diffusion. The original diffusion then has true scaling parameter

$$\gamma = \gamma^{\text{scaled}}/\sqrt{i_{\max}} = \gamma^{\text{scaled}}/\sqrt{1259}. \tag{7.3}$$

This is shown in Figure 8, and the result of the training is shown in Figure 9. Following the discussion in section 6, a fixed Matérn parameter $\nu = 9/2$ was used. The initial parameters were $\alpha = 1.8, p = 0.5, \gamma^{\text{scaled}} = 1, \sigma = 0.1, \theta = 0.2$. The trained parameters were $\alpha = 1.667$, $p = 0.422$, $\gamma^{\text{scaled}} = 0.235$, and hyperparameters $\sigma = 0.0617$, $\theta = 1.173$ and learned noise $\sigma_n = 0.0344$. The true $\gamma = 0.0066$. The optimization wall time was 6 minutes and 31 seconds.

The conclusion of this training is that, in this model, the transition density $u(t,x)$ of the original Intel (normalized)/SP500 (normalized) process is governed by the fractional diffusion equation

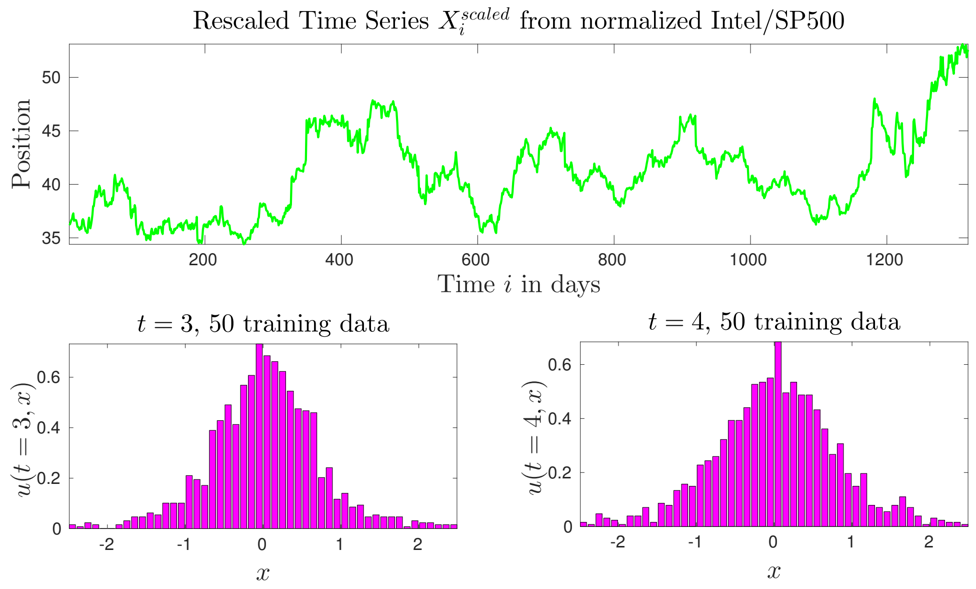

FIG. 8. *Top: The rescaled time series given by* (7.2). *The relative (normalized) Intel/SP*500 *stock is multiplied by* $\sqrt{1259}$ *to put it in the same window as standard Brownian motion. Bottom: The empirical histograms to be used as data to discover the fractional-order diffusion equation.*

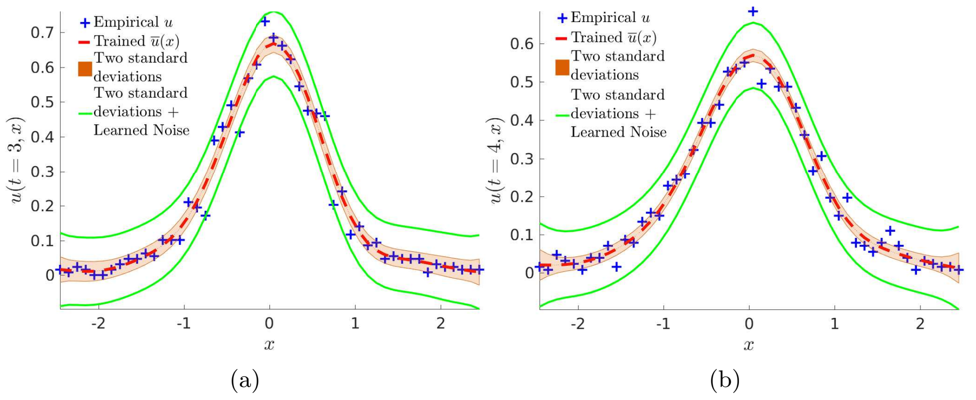

FIG. 9. *The result of training the Gaussian process using the histogram data in Figure* 8. *The blue crosses are the normalized histogram/empirical distribution function data. The red curve/orange bars show the mean/two standard deviations of the trained Gaussian process, while the green curve shows two standard deviations* $\pm$ *the learned noise* $2\sigma_n$.

$$\begin{aligned}\frac{\partial u}{\partial t} &= \frac{0.0066^{1.667}}{|\cos(1.667\pi/2)|}\left[0.422\left({}^{RL}_{-\infty}D^{1.667}_{x}u\right) + 0.578\left({}^{RL}_{x}D^{1.667}_{\infty}u\right)\right] \\ &= 0.000113\left({}^{RL}_{-\infty}D^{1.667}_{x}u\right) + 0.000155\left({}^{RL}_{x}D^{1.667}_{\infty}u\right).\end{aligned} \tag{7.4}$$

Equivalently, the time series may be described as an $\alpha$-stable process,

$$X_{i+n} - X_i \sim S_{1.667}\left(-0.156, 0.0066\left(n^{1/1.667}\right), 0\right). \tag{7.5}$$

However, for modeling purposes, it should be kept in mind that the training was performed using data in the interval $[-2.5, 2.5]/\sqrt{1259}$ with $n = 3$ and $n = 4$; increments greater than $2.5/\sqrt{1259}$ in four units of time were excluded. Thus, for consistency, the stable densities should be truncated to this interval as well and renormalized. If increments in four units of time are drawn from the 1.667-stable density (7.5) with $n = 4$ truncated to $[-2.5, 2.5]/\sqrt{1259}$, increments in one unit of time should be drawn as from the appropriate ($n = 1$) density truncated to $[-2.5, 2.5]/(\sqrt{1259} \times 4^{1/1.667}) = [-0.031, 0.031]$ and normalized:

$$X_{i+1} - X_i \sim C_{\text{norm}} \mathbb{1}_{[-0.031,0.031]} S_{1.667}(-0.156, 0.0066, 0). \tag{7.6}$$

Sampling of this distribution can be performed by sampling $S_{1.667}(-0.156, 0.0066, 0)$ and rejecting draws greater in magnitude than 0.031. Backtesting with this model, shown in Figure 10, yields good agreement with both the trend and volatility of the historical data.

Fitting $\alpha$-stable densities to financial data is important for risk management and is of relevance to trading strategies based on assumptions of underlying $\alpha$-stable random behavior. In this direction, fractional Black–Scholes equations have been introduced

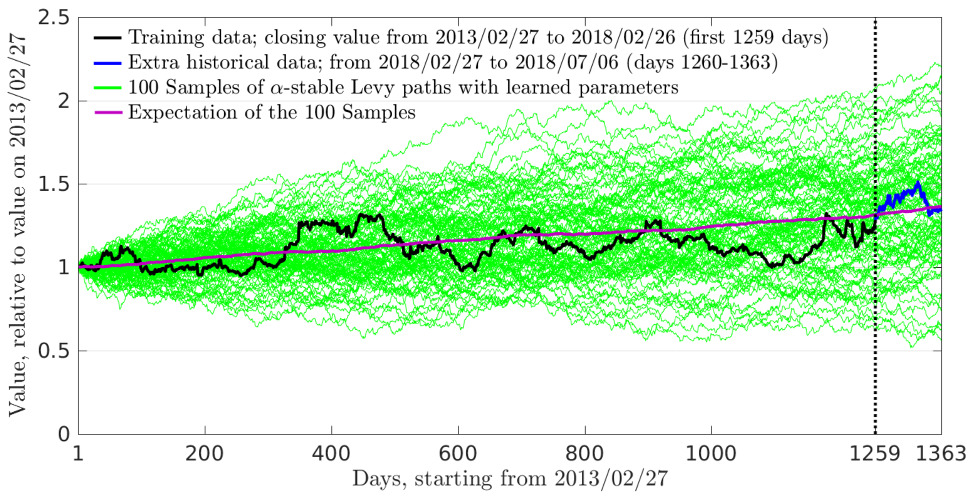

Fig. 10. *Backtesting/validation of the trained $\alpha$-stable parameters. The black curve shows the training data of normalized Intel/normalized S&P* 500 *for* 1,259 *days starting February* 27, 2013*. The blue curve shows extra historical time series data, from February* 26, 2018 *(the vertical dotted line at day* 1,259*), until July* 26, 2018 *(not used to train). Each of the* 100 *green curves is a sample path of the truncated $\alpha$-stable process* (7.6)*. The dark/filled region of the envelope of samples contains the historical data, providing a fairly sharp estimate of volatility in the first year. The purple curve is the expectation of the samples, which is accurate initially but does not take into account the mean-reverting behavior of the stock over* 5 *years.*

[6], [13] as appropriate models for hedging, since standard Black–Scholes theory is based on assumptions of normality/log-normality of the underlying processes. The example here can serve as a building block to applying fractional Black–Scholes theory to financial data.

## 8. Conclusion.

The Gaussian processes methodology based on the implementation of fractional derivatives and stationary covariance kernels in Theorem 3.1 allows for the effective and efficient discovery of linear space-fractional equations in $\mathbb{R}^d$. We have demonstrated the feasibility, robustness, and accuracy of this methodology. Due to the versatilty of fractional archetypes and user-controlled parsimony, we demonstrated the appeal of the method even for discovering integer-order equations and discussed a novel approach to interpolating multiterm linear PDEs. The methodology allows for versatile discovery of linear space-fractional differential equations in $\mathbb{R}^d$ in many applications where fractional or anomalous behavior is expected, as illustrated by the discovery of a fractional diffusion equation for relative stock performance. This can be used to calibrate $\alpha$-stable parameters from time series and has potential impact in risk management and fractional Black–Scholes theory.

This work is an extension of [27] and [24] to the case of fractional-order operators and a broader class of covariance kernels, including the Matérn class. Compared to more standard approaches for inverse and parameter estimation problems, the Gaussian process regression approach has the advantage that repeated forward solution of the differential equation is not required. Rather, the equation parameters are discovered by a single Gaussian process regression through scattered observations, which involves entirely standard maximum likelihood estimation using L-BFGS, albeit with a covariance kernel that is constrained by the differential equation. As discussed in [27] and [24], this constraint regularizes the Gaussian process and allows physical laws to be discovered with comparatively few data points. On the other hand, extension to space-dependent rather than constant coefficient fields, for example, presents a challenge, as strong prior knowledge of the coefficient field would be required in order to parametrize it and apply this methodology.

Regarding extensions of this work, one potential drawback of the methodology is that Gaussian processes may not scale well to large data sets. In this regard, remedies proposed in [12, 22] may be worth exploring. On the other hand, neural networks are intrinsically suited to larger data sets. Moreover, unlike Gaussian processes, neural networks do not require a linear relationship of the form (1.1) and can be freely used to discover parametrized nonlinear differential equations. Neural networks were used to solve and discover (integer-order) PDEs in [28, 29]. The methodology therein was further utilized in [30] and [23] to train neural networks to distill the actual dynamics of nonlinear dynamical systems and nonlinear PDEs, respectively. See also [15] for a different approach. In general, however, extension of these methods to allow for fractional-order differential operators would benefit from *fractional automatic differentiation* of neural networks, which remains a major conceptual and numerical challenge due to a lack of classical chain rule for fractional-order operators.

## Appendix A. Proof of Theorem 3.1.

Recall the shift property of the Fourier transform:

$$\mathcal{F}\{f(\boldsymbol{x}-\boldsymbol{a})\}(\boldsymbol{\xi}) = e^{-i\boldsymbol{a}\cdot\boldsymbol{\xi}}\mathcal{F}\{f(\boldsymbol{x})\}(\boldsymbol{\xi}).$$

By the stationarity of $k$,

$$\mathcal{F}_{\boldsymbol{x}}\{k(\boldsymbol{x},\boldsymbol{y})\} = \mathcal{F}_{\boldsymbol{x}}\{K(\boldsymbol{x}-\boldsymbol{y})\} = e^{-i\boldsymbol{y}\cdot\boldsymbol{\xi}}\hat{K}(\boldsymbol{\xi}).$$

By the multiplier property of $\mathcal{L}$, in Fourier space, $\mathcal{L}_{\boldsymbol{x}}k$ is given by multiplying $\mathcal{F}_{\boldsymbol{x}}\{k\}$ by the symbol $m(\boldsymbol{\xi})$:

$$\mathcal{F}_{\boldsymbol{x}}\{\mathcal{L}_{\boldsymbol{x}}k\} = e^{-i\boldsymbol{y}\cdot\boldsymbol{\xi}}m(\boldsymbol{\xi})\hat{K}(\boldsymbol{\xi}).$$

Taking the inverse Fourier transform gives

$$\begin{aligned}\mathcal{L}_{\boldsymbol{x}}k &= \mathcal{F}_{\boldsymbol{x}}^{-1}\left\{\mathcal{F}_{\boldsymbol{x}}\{\mathcal{L}_{\boldsymbol{x}}k\}\right\}\\ &= \mathcal{F}_{\boldsymbol{x}}^{-1}\left\{e^{-i\boldsymbol{y}\cdot\boldsymbol{\xi}}m(\boldsymbol{\xi})\hat{K}(\tilde{\boldsymbol{x}})\right\}\\ &= \frac{1}{\sqrt{2\pi}}\int_{\mathbb{R}^d} e^{i\boldsymbol{x}\cdot\boldsymbol{\xi}}e^{-i\boldsymbol{y}\cdot\boldsymbol{\xi}}m(\boldsymbol{\xi})\hat{K}(\boldsymbol{\xi})d\boldsymbol{\xi}. \qquad \text{(A.1)}\end{aligned}$$

This is the first of formulas (3.11). As for $\mathcal{L}_{\boldsymbol{y}}k$, define the transpose operator by $F^T(\boldsymbol{x},\boldsymbol{y}) = F(\boldsymbol{y},\boldsymbol{x})$. Then $\mathcal{L}_{\boldsymbol{y}}F = [\mathcal{L}_{\boldsymbol{x}}F^T]^T$ and symmetry of $k$ implies

$$\mathcal{L}_{\boldsymbol{y}}k = \left[\mathcal{L}_{\boldsymbol{x}}k^T\right]^T = \left[\mathcal{L}_{\boldsymbol{x}}k\right]^T.$$

In other words, $\mathcal{L}_{\boldsymbol{y}}k(\boldsymbol{x},\boldsymbol{y}) = \mathcal{L}_{\boldsymbol{x}}k(\boldsymbol{y},\boldsymbol{x})$ and does not require a separate computation (this is true for any covariance kernel). So far, only a single $d$-dimensional integation is required, which is the benefit of knowing the Fourier transform $\hat{K}$ of $K$ analytically. Next, we see how the stationary property gives the formula for $\mathcal{L}_{\boldsymbol{y}}\mathcal{L}_{\boldsymbol{x}}k$. By the multiplier property of $\mathcal{L}$, we have

$$\mathcal{L}_{\boldsymbol{y}}\mathcal{L}_{\boldsymbol{x}}k = \mathcal{F}_{\boldsymbol{y}}^{-1}\left\{\mathcal{F}_{\boldsymbol{y}}\left\{\mathcal{L}_{\boldsymbol{y}}\left[\mathcal{L}_{\boldsymbol{x}}k(\boldsymbol{x},\boldsymbol{y})\right]\right\}(\boldsymbol{\xi'})\right\} = \mathcal{F}_{\boldsymbol{y}}^{-1}\left\{m(\boldsymbol{\xi'})\mathcal{F}_{\boldsymbol{y}}\{\mathcal{L}_{\boldsymbol{x}}k(\boldsymbol{x},\boldsymbol{y})\}(\boldsymbol{\xi'})\right\}.$$

Let us examine the inner term $\mathcal{F}_{\boldsymbol{y}}\{\mathcal{L}_{\boldsymbol{x}}k(\boldsymbol{x},\boldsymbol{y})\}(\boldsymbol{\xi'})$. The Fourier transform $\mathcal{F}_{\boldsymbol{y}}$ passes through the integral over $\boldsymbol{\xi}$ in the final formula (A.1) for $\mathcal{L}_{\boldsymbol{x}}k(\boldsymbol{x},\boldsymbol{y})$. Inside that integral, $\mathcal{F}_{\boldsymbol{y}}$ only sees a constant term (independent of $\boldsymbol{y}$) times the complex exponential $e^{-i\boldsymbol{y}\cdot\boldsymbol{\xi}}$. Thus, $\mathcal{F}_{\boldsymbol{y}}\{\mathcal{L}_{\boldsymbol{x}}k(\boldsymbol{x},\boldsymbol{y})\}(\boldsymbol{\xi'})$ reduces to a $\delta$-function in $\boldsymbol{\xi'}$:

$$\begin{aligned}\mathcal{F}_{\boldsymbol{y}}\left\{\mathcal{L}_{\boldsymbol{x}}k(\boldsymbol{x},\boldsymbol{y})\right\}(\boldsymbol{\xi'}) &= \frac{1}{\sqrt{2\pi}}\int_{\mathbb{R}^d}\left[\sqrt{2\pi}\delta(\boldsymbol{\xi}+\boldsymbol{\xi}')\right]e^{i\boldsymbol{x}\cdot\boldsymbol{\xi}}m(\boldsymbol{\xi})\hat{K}(\boldsymbol{\xi})d\boldsymbol{\xi}\\ &= \int_{\mathbb{R}^d}\delta(\boldsymbol{\xi}+\boldsymbol{\xi}')e^{i\boldsymbol{x}\cdot\boldsymbol{\xi}}m(\boldsymbol{\xi})\hat{K}(\boldsymbol{\xi})d\boldsymbol{\xi}.\end{aligned}$$

Therefore, we obtain the second of formulas (3.11):

$$\begin{aligned}\mathcal{L}_{\boldsymbol{y}}\mathcal{L}_{\boldsymbol{x}}k &= \mathcal{F}_{\boldsymbol{y}}^{-1}\left\{m(\boldsymbol{\xi'})\int_{\mathbb{R}^d}\delta(\boldsymbol{\xi'}+\boldsymbol{\xi})e^{i\boldsymbol{x}\cdot\boldsymbol{\xi}}m(\boldsymbol{\xi})\hat{K}(\boldsymbol{\xi})d\boldsymbol{\xi}\right\}\\ &= \frac{1}{\sqrt{2\pi}}\int_{\mathbb{R}^d}e^{i\boldsymbol{y}\cdot\boldsymbol{\xi'}}\left\{m(\boldsymbol{\xi'})\int_{\mathbb{R}^d}\delta(\boldsymbol{\xi'}+\boldsymbol{\xi})e^{i\boldsymbol{x}\cdot\boldsymbol{\xi}}m(\boldsymbol{\xi})\hat{K}(\boldsymbol{\xi})d\boldsymbol{\xi}\right\}d\boldsymbol{\xi'}\\ &= \frac{1}{\sqrt{2\pi}}\int_{\mathbb{R}^d}\int_{\mathbb{R}^d}e^{i\boldsymbol{y}\cdot\boldsymbol{\xi'}}\delta(\boldsymbol{\xi'}+\boldsymbol{\xi})e^{i\boldsymbol{x}\cdot\boldsymbol{\xi}}m(\boldsymbol{\xi})m(\boldsymbol{\xi'})\hat{K}(\boldsymbol{\xi})d\boldsymbol{\xi'}d\boldsymbol{\xi}\\ &= \frac{1}{\sqrt{2\pi}}\int_{\mathbb{R}^d}e^{-i\boldsymbol{y}\cdot\boldsymbol{\xi}}e^{i\boldsymbol{x}\cdot\boldsymbol{\xi}}m(\boldsymbol{\xi})m(-\boldsymbol{\xi})\hat{K}(\boldsymbol{\xi})d\boldsymbol{\xi}.\end{aligned}$$

## REFERENCES

[1] L. Alfonso, R. Mansilla, and C. A. Terrero-Escalante, *On the scaling of the distribution of daily price fluctuations in the Mexican financial market index*, Phys. A, 391 (2012), pp. 2990–2996, https://doi.org/10.1016/j.physa.2012.01.023.

[2] J. Bongard and H. Lipson, *Automated reverse engineering of nonlinear dynamical systems*, Proc. Nat. Acad. Sci., 104 (2007), pp. 9943–9948, https://doi.org/10.1073/pnas.0609476104.

[3] S. Borak, W. Härdle, and R. Weron, *Stable distributions*, SFB 649 Discussion Papers SFB649DP2005-008, Humboldt University, Collaborative Research Center 649, 2005, https://EconPapers.repec.org/RePEc:hum:wpaper:sfb649dp2005-008.

[4] B. O. Bradley and M. S. Taqqu, *Financial risk and heavy tails*, in Handbook of Heavy Tailed Distributions in Finance, S. T. Rachev, ed., Handbooks in Finance 1, North-Holland, Amsterdam, 2003, pp. 35–103, https://doi.org/10.1016/B978-044450896-6.50004-2.

[5] S. L. Brunton, J. L. Proctor, and J. N. Kutz, *Discovering governing equations from data by sparse identification of nonlinear dynamical systems*, Proc. Nat. Acad. Sci., 113 (2016), pp. 3932–3937, https://doi.org/10.1073/pnas.1517384113.

[6] A. Cartea and D. del Castillo-Negrete, *Fractional diffusion models of option prices in markets with jumps*, Phys. A, 374 (2007), pp. 749–763, https://doi.org/10.1016/j.physa.2006.08.071.

[7] P. Chaussé, *Computing generalized method of moments and generalized empirical likelihood with R*, J. Statist. Softw., 34 (2010), pp. 1–35, https://doi.org/10.18637/jss.v034.i11.

[8] E. F. Fama, *The behavior of stock-market prices*, J. Business, 38 (1965), pp. 34–105, http://www.jstor.org/stable/2350752.

[9] L. Feng and X. Lin, *Pricing Bermudan options in Lévy process models*, SIAM J. Financial Math., 4 (2013), pp. 474–493, https://doi.org/10.1137/120881063.

[10] N. García Trillos and D. Sanz-Alonso, *The Bayesian formulation and well-posedness of fractional elliptic inverse problems*, Inverse Problems, 33 (2017), 065006, https://doi.org/10.1088/1361-6420/aa711.

[11] M. Haas and C. Pigorsch, *Financial economics, fat-tailed distributions*, in Complex Systems in Finance and Econometrics, R. A. Meyers, ed., Springer, New York, 2011, pp. 308–339, https://doi.org/10.1007/978-1-4419-7701-4_18.

[12] J. Hensman, N. Fusi, and N. D. Lawrence, *Gaussian Processes for Big Data*, preprint, arXiv:1309.6835, 2013.

[13] H. Kleinert and J. Korbel, *Option pricing beyond Black-Scholes based on double-fractional diffusion*, Phys. A, 449 (2016), pp. 200–214, https://doi.org/10.1016/j.physa.2015.12.125.

[14] A. Lischke, G. Pang, M. Gulian, F. Song, C. Glusa, X. Zheng, Z. Mao, W. Cai, M. M. Meerschaert, M. Ainsworth, and G. E. Karniadakis, *What Is the Fractional Laplacian?*, preprint, arXiv:1801.09767, 2018.

[15] Z. Long, Y. Lu, X. Ma, and B. Dong, *PDE-Net: Learning PDEs from Data*, preprint, arXiv:1710.09668, 2017.

[16] B. Mandelbrot, *The variation of certain speculative prices*, J. Business, 36 (1963), https://EconPapers.repec.org/RePEc:ucp:jnlbus:v:36:y:1963:p:394.

[17] A. Matache, C. Schwab, and T. P. Wihler, *Fast numerical solution of parabolic integrodifferential equations with applications in finance*, SIAM J. Sci. Comput., 27 (2005), pp. 369–393, https://doi.org/10.1137/030602617.

[18] M. M. Meerschaert and A. Sikorskii, *Stochastic Models for Fractional Calculus*, De Gruyter Stud. Math., 43, Walter de Gruyter, Berlin, 2012.

[19] R. Metzler and J. Klafter, *The random walk's guide to anomalous diffusion: A fractional dynamics approach*, Phys. Rep., 339 (2000), pp. 1–77, https://doi.org/10.1016/S0370-1573(00)00070-3.

[20] J. P. Nolan, *Maximum likelihood estimation and diagnostics for stable distributions*, in Lévy Processes: Theory and Applications, Birkhäuser-Boston, Boston, MA, 2001, pp. 379–400, https://doi.org/10.1007/978-1-4612-0197-7_17.

[21] G. Pang, P. Perdikaris, W. Cai, and G. E. Karniadakis, *Discovering variable fractional orders of advection–dispersion equations from field data using multi-fidelity Bayesian optimization*, J. Comput. Phys., 348 (2017), pp. 694–714, https://doi.org/10.1016/j.jcp.2017.07.052.

[22] M. Raissi, *Parametric Gaussian Process Regression for Big Data*, preprint, arXiv:1704.03144, 2017.

[23] M. Raissi, *Deep Hidden Physics Models: Deep Learning of Nonlinear Partial Differential Equations*, preprint, arXiv:1801.06637, 2018.
[24] M. Raissi and G. E. Karniadakis, *Hidden physics models: Machine learning of nonlinear partial differential equations*, J. Comput. Phys., 357 (2018), pp. 125–141, https://doi.org/10.1016/j.jcp.2017.11.039.
[25] M. Raissi, P. Perdikaris, and G. Karniadakis, *Numerical Gaussian processes for time-dependent and nonlinear partial differential equations*, SIAM J. Sci. Comput., 40 (2018), pp. A172–A198, https://doi.org/10.1137/17M1120762.
[26] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Inferring solutions of differential equations using noisy multi-fidelity data*, J. Comput. Phys., 335 (2017), pp. 736–746, https://doi.org/10.1016/j.jcp.2017.01.060.
[27] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Machine learning of linear differential equations using Gaussian processes*, J. Comput. Phys., 348 (2017), pp. 683–693, https://doi.org/10.1016/j.jcp.2017.07.050.
[28] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics Informed Deep Learning (Part* I*): Data-Driven Solutions of Nonlinear Partial Differential Equations*, preprint, arXiv:1711.10561, 2017.
[29] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics Informed Deep Learning (Part* II*): Data-Driven Discovery of Nonlinear Partial Differential Equations*, preprint, arXiv:1711.10566, 2017.
[30] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Multistep Neural Networks for Data-Driven Discovery of Nonlinear Dynamical Systems*, preprint, arXiv:1801.01236, 2018.
[31] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*, Adaptive Computation and Machine Learning, MIT Press, Cambridge, MA, 2006.
[32] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz, *Data-driven discovery of partial differential equations*, Science Advances, 3 (2017), https://doi.org/10.1126/sciadv.1602614.
[33] E. Scalas and K. Kim, *The art of fitting financial time series with Lévy stable distributions*, J. Korean Phys. Soc., 50 (2006).
[34] M. Schmidt and H. Lipson, *Distilling free-form natural laws from experimental data*, Science, 324 (2009), pp. 81–85, https://doi.org/10.1126/science.1165893.
[35] M. L. Stein, *Interpolation of Spatial Data: Some Theory for Kriging*, Springer Ser. Statist., Springer, New York, 1999.
[36] A. M. Stuart, *Inverse problems: A Bayesian perspective*, Acta Numer., 19 (2010), 451559, https://doi.org/10.1017/S0962492910000061.
[37] P. Tankov and R. Cont, *Financial Modelling with Jump Processes, 2nd ed.*, Chapman and Hall/CRC Financ. Math. Ser. Z, CRC Press, Boca Raton, FL, 2015, https://books.google.com/books?id=-fZtKgAACAAJ.
[38] A. W. Van der. Vaart, *Asymptotic Statistics*, Cambridge Ser. Stat. Probab. Math., Cambridge University Press, Cambridge, UK, 1998, https://doi.org/10.1017/CBO9780511802256.
[39] P. Wilmott, *Paul Wilmott Introduces Quantitative Finance*, 2nd ed., Wiley-Interscience, New York, 2007.
[40] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed., Morgan Kaufmann, San Francisco, 2011.