

Adaptive Density Tracking by Quadrature for Stochastic Differential Equations

Ryleigh A. Moore*¹ and Akil Narayan¹

¹Scientific Computing and Imaging Institute, and Department of Mathematics,
University of Utah

May 19, 2021

Abstract

Density tracking by quadrature (DTQ) is a numerical procedure for computing solutions to Fokker-Planck equations that describe probability densities for stochastic differential equations (SDEs). In this paper, we extend upon existing tensorized DTQ procedures by utilizing a flexible quadrature rule that allows for unstructured, adaptive meshes. We propose and describe the procedure for N -dimensions, and demonstrate that the resulting adaptive procedure is significantly more efficient than a tensorized approach. Although we consider two-dimensional examples, all our computational procedures are extendable to higher dimensional problems.

Keywords: stochastic differential equations, Leja points, numerical methods

1 Problem History and Background

Stochastic differential equations (SDEs) are prevalent in many areas of research. Kloeden and Platen [1] outline a variety of SDE uses, including population dynamics, protein kinetics, psychology problems involving neuronal activity, investment finance and option pricing, turbulent diffusion of a particle, radio-astronomy and the analysis of stars, helicopter rotor and satellite orbit stability, biological waste treatment with analysis of air and water quality, seismology and structural mechanics, the stability of materials prone to fatigue cracking, and blood clotting dynamics and cellular energetics. In this paper, we are interested in solving SDEs and their associated Fokker-Planck equations.

1.1 Stochastic Differential Equations

Let \mathbf{W}_t be an N -dimensional Wiener Process and \mathbf{X}_t be an N -dimensional vector stochastic Itô diffusion process governed by the SDE

$$d\mathbf{X}_t = \mathbf{f}(\mathbf{X}_t, t)dt + \mathbf{g}(\mathbf{X}_t, t)d\mathbf{W}_t \quad (1)$$

with the drift $\mathbf{f}(\mathbf{X}_t, t)$ as an N -dimensional vector and the diffusion defined by an $N \times N$ -dimensional matrix $\mathbf{g}(\mathbf{X}_t, t)$. This equation is endowed with a $t = 0$ initial condition \mathbf{X}_0 .

*Corresponding author email: rmoore@math.utah.edu

R. Moore and A. Narayan were partially supported by AFOSR under award FA9550-20-1-0338. A. Narayan is partially supported by NSF DMS-1848508.

The evolution of the probability density function for \mathbf{X}_t is governed by the corresponding Fokker-Planck partial differential equation (PDE). The Fokker-Planck equation for the evolution of the probability density $p(\mathbf{x}, t)$ of the random variable \mathbf{X}_t from (1) is given by

$$\frac{\partial}{\partial t} p(\mathbf{x}, t) = - \sum_{i=1}^N \frac{\partial}{\partial x^{(i)}} [\mathbf{f}_i(\mathbf{x}, t) p(\mathbf{x}, t)] + \sum_{i,j=1}^N \frac{\partial^2}{\partial x^{(i)} \partial x^{(j)}} [\mathbf{D}_{ij}(\mathbf{x}, t) p(\mathbf{x}, t)] \quad (2)$$

where $\mathbf{x} = (x^{(1)}, \dots, x^{(N)})^T$. The diffusion tensor \mathbf{D} is related to the SDE diffusion \mathbf{g} by

$$\mathbf{D}_{ij}(\mathbf{x}, t) = \frac{1}{2} \sum_{\ell=1}^N \mathbf{g}_{i\ell}(\mathbf{x}, t) \mathbf{g}_{j\ell}(\mathbf{x}, t),$$

see, e.g., [2, p. 5]. Since $p(\mathbf{x}, t)$ is a probability density function, it satisfies a normalization condition

$$\int p(\mathbf{x}, t) d\mathbf{x} = 1.$$

This paper focuses on numerical approximation of the time-dependent probability density function $p(\mathbf{x}, t)$ governed by (2).

1.2 Current Methods

Several methods have been developed to numerically approximate the statistics of SDE solution \mathbf{X}_t , or the probability density function $p(\mathbf{x}, t)$ from the associated Fokker-Planck PDE in equation (2). Perhaps among the more straightforward approaches to approximate statistics is through Monte Carlo simulation [3], which typically collects a potentially large ensemble of realizations of \mathbf{X}_t by computing solutions to (1). The large number of samples needed to sufficiently approximate the solution of the SDE makes using this method with sufficient accuracy computationally expensive.

Numerous numerical methods compute solutions to the Fokker-Planck equation, such as finite element methods (FEMs) [4, 5, 6, 7, 8, 9, 10, 11] and finite difference methods (FDMs) [4, 11, 12]. FEMs are often preferable over FDMs to solve the Fokker-Planck equation because of their accuracy and stability; however, they can be more complicated to implement compared to FDMs. Current FDMs are empirically less numerically stable than FEMs, but they also usually require less memory and computational power to implement [4, 11]. Both FEMs and FDMs suffer from the curse of dimensionality stemming from the computational difficulty of forming a sufficiently dense mesh in N dimensions.

When using FDMs or FEMs, erroneous oscillations and negative values often arise if the drift is large compared to the diffusion. One method to address this challenge utilizes a moving finite element mesh [13]; basis functions, which depend on time instead of only on space, such as standard FEMs, and that satisfy the drift part of the equation, are used to eliminate the spurious oscillations. Some adaptive FEM procedures monitor regions of non-negligible probability and adjust the mesh coarseness appropriately [14]. Adaptive FEM procedures also adjust the mesh based on the local value and gradient of p near boundary regions [15]. Additionally, finite volume methods (FVMs) have been applied to the conservation form of the Fokker-Planck equation, utilizing a linear multistep method for temporal discretization [16]. Such procedures are also typically adaptive, adjusting the mesh and time step based on an error tolerance criterion. Furthermore, adaptive meshes are utilized in numerical path integration methods, also called transformed path integral methods [17]. These methods propagate the grid in a Lagrangian way relative to a given fixed grid in a transformed space.

Deep learning approaches have been leveraged to numerically approximate solutions to the Fokker-Planck equation. If a large amount of training data is available, neural networks can be used to learn solution behavior [18]. Of course, this requires availability of such training data, and guaranteeing generalizability and accuracy with such approaches is often difficult.

The curse of dimensionality is still a concern with the above approaches. The difficulty of solving the Fokker-Planck equation increases substantially with the dimension N of the problem. Some nontraditional FEM methods have been used to solve four-dimensional problems [10], but more work is needed for higher dimensional problems to become tractable.

In this paper, we wish to extend the results of Bhat and Madushani to approximate the probability density function of SDEs in high dimensions using density tracking by quadrature (DTQ) [19, 20]. DTQ has also been described previously as numerical path integration [21]. In one dimension, DTQ has been shown to be a convergent method that computes an approximation to the probability density function $p(\mathbf{x}, t)$ of \mathbf{X}_t on a discrete grid. In some examples, DTQ is 100 times faster compared to other methods with similar accuracy [20]. Existing DTQ methods utilize a trapezoidal rule for integration with a tensorized mesh over N -dimensional space. $N = 1$ dimension and $N = 2$ dimensional formulations have been investigated [20, 19], but the challenge of applying such methods in higher dimensions is still significant.

1.3 Outline and Contributions of this Paper

In this paper, we work to augment current DTQ methods by implementing a more accurate and flexible quadrature rule along with adaptive mesh updates to minimize the computational cost when solving the Chapman-Kolmogorov update in equation (6) using quadrature. More specifically, allowing an unstructured mesh provides flexibility in high dimensions to allocate degrees of freedom to areas of high probability and away from areas of low probability. The unstructured mesh allows for nontensorial discretizations and partially addresses the curse of dimensionality.

We summarize the DTQ method and its previous use with a tensorized mesh and trapezoidal quadrature rule by Bhat and Madushani [19]. Then, we discuss the advances made in this paper by utilizing an unordered, adaptive mesh with an interpolatory quadrature rule. A Laplace approximation is used to rewrite the Chapman-Kolmogorov integral in equation (5) so that Hermite polynomials can be utilized to interpolate the integrand on Leja points and step the solution forward in time. We also detail the implementation of an adaptive mesh where the boundary values are adjusted to track the density values while reducing the number of points necessary for the procedure. Finally, we apply the adaptive DTQ method to some example problems.

2 Density Tracking by Quadrature

2.1 DTQ Procedure

We present DTQ in the framework of N -dimensional autonomous SDEs in (1). For a fixed temporal stepsize $h > 0$, we first discretize the SDE (1) in time using the Euler-Maruyama method,

$$\tilde{\mathbf{X}}_{n+1} = \tilde{\mathbf{X}}_n + \mathbf{f}(\tilde{\mathbf{X}}_n, t)h + \mathbf{g}(\tilde{\mathbf{x}}_n, t)\sqrt{h}\mathbf{Z}_{n+1} \quad (3)$$

where $\tilde{\mathbf{X}}_n$ represents an approximation of the state \mathbf{X}_t at time $t_n = nh$. \mathbf{Z}_{n+1} is a standard N -dimensional normal random variable (i.e., mean 0, identity covariance).

The work in [19, 20] interprets the time discretized equation (3) as a discrete-time Markov chain where the initial state, $\tilde{\mathbf{X}}_0$, corresponds to a discretization of the density $p(\mathbf{x}, 0)$ of the initial condition of the SDE (1). Let $\tilde{p}(\mathbf{x}, t_n)$ denote the density at location \mathbf{x} at time t_n of the Markov chain. From equation (3), we observe that the conditional density of $\tilde{\mathbf{X}}_{n+1}$ given $\tilde{\mathbf{X}}_n = \mathbf{y}$ is Gaussian with mean $\tilde{\boldsymbol{\mu}} = \mathbf{y} + \mathbf{f}(\mathbf{y})h$ and covariance $\tilde{\boldsymbol{\Sigma}} = h\mathbf{g}(\mathbf{y})\mathbf{g}(\mathbf{y})^T$,

$$\tilde{p}(\tilde{\mathbf{X}}_{n+1} = \mathbf{x} | \tilde{\mathbf{X}}_n = \mathbf{y}) := G(\mathbf{x}, \mathbf{y}) \quad (4)$$

$$G(\mathbf{x}, \mathbf{y}) := \frac{1}{\sqrt{(2\pi)^N |\tilde{\boldsymbol{\Sigma}}|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \tilde{\boldsymbol{\mu}})^T \tilde{\boldsymbol{\Sigma}}^{-1}(\mathbf{x} - \tilde{\boldsymbol{\mu}})\right).$$

Notice that $G(\mathbf{x}, \mathbf{y})$ depends on the drift \mathbf{f} and diffusion \mathbf{g} through $\tilde{\boldsymbol{\mu}}$ and $\tilde{\boldsymbol{\Sigma}}$, but we omit this explicit notational dependence.

The evolution of the density of $\tilde{\mathbf{X}}$ is described by the associated Chapman-Kolmogorov equation,

$$\begin{aligned}\tilde{p}(\mathbf{x}, t_{n+1}) &= \int_{\mathbb{R}^N} \tilde{p}(\mathbf{x}_{n+1} = \mathbf{x} | \mathbf{x}_n = \mathbf{y}) \tilde{p}(\mathbf{y}, t_n) d\mathbf{y} \\ &= \int_{\mathbb{R}^N} G(\mathbf{x}, \mathbf{y}) \tilde{p}(\mathbf{y}, t_n) d\mathbf{y}.\end{aligned}\tag{5}$$

The evolution of $\tilde{p}(\mathbf{x}, t_n)$ proceeds by discretizing (5) in space. Let $\{\mathbf{y}_1, \dots, \mathbf{y}_s\}$ be a set of mesh points. Then, the density \tilde{p} in (5) can be approximated by \hat{p} , which is defined as a discretization of (5):

$$\hat{p}(\mathbf{y}_j, t_{n+1}) = \sum_{i=1}^m G(\mathbf{y}_j, \mathbf{y}_i) \hat{p}(\mathbf{y}_i, t_n) \omega_i\tag{6}$$

with the starting condition $\hat{p}(\mathbf{y}_j, 0) = \tilde{p}(\mathbf{y}_j, 0)$, where ω_i are quadrature weights. In $N = 1$ dimension with an equispaced mesh, the trapezoidal rule (on an infinite domain) has previously been employed, and is accompanied by error estimates [20]. DTQ using a trapezoidal rule for quadrature has been used for parameter inference problems [22], and a two-dimensional implementation was employed to analyze basketball tracking data from the National Basketball Association [20].

2.2 Tensorized DTQ

In more than one dimension, $N > 1$, a straightforward choice for the mesh is a tensorial grid, e.g., an isotropic grid is formed from tensorization of a univariate grid,

$$\{\mathbf{y}_i\}_{i=1}^s = \bigotimes_{j=1}^N \{x_1, \dots, x_q\}, \quad \{x_1, \dots, x_q\} \subset \mathbb{R}.$$

In this case, the discretization of (6) can proceed dimension by dimension. If the univariate grid $\{x_i\}_{i=1}^q$ is equispaced with mesh stepsize $\kappa > 0$, then (6) can be written as

$$\hat{p}(\mathbf{y}_j, t_{n+1}) = \kappa^N \sum_{i=1}^s G(\mathbf{y}_j, \mathbf{y}_i) \hat{p}(\mathbf{y}_i, t_n).$$

In vector form, the above is

$$\mathbf{P}_{n+1} = \kappa^N \mathbf{G} \mathbf{P}_n, \quad (\mathbf{G})_{i,j} = G(\mathbf{y}_i, \mathbf{y}_j),$$

where $\mathbf{P}_{n+1} := [\hat{p}(\mathbf{y}_1, t_{n+1}), \dots, \hat{p}(\mathbf{y}_s, t_{n+1})]^T \in \mathbb{R}^s$. $\mathbf{G}_{i,j}$ contains values describing the movement of mass density; however, the matrix $\kappa^N \mathbf{G}$ is not a stochastic matrix in general [20].

The numerical solution \hat{p} can, in principle, be directly computed using this procedure. However, for higher dimensional problems we require q^N mesh points for the tensorization strategy. For example, in four dimensions with 100 points per dimension, we will need $m = 10^8$ points, which is computationally prohibitive. In order to extend DTQ to higher dimensions, we provide an alternative strategy to discretize the integral in equation (5), which uses an unstructured set of mesh points.

3 DTQ on an Unstructured Mesh

We describe our procedure for implementing DTQ on an unstructured mesh in N dimensions. We utilize a nontensorized, adaptive mesh and an interpolatory quadrature rule to approximate the integral in (5) by treating a portion of the integrand as a Gaussian density. For each point in the

global unstructured mesh, $\mathbf{y}_j \in \{\mathbf{y}_1, \dots, \mathbf{y}_s\}$, we use a set of nearest neighbors of \mathbf{y}_j to compute Leja points to use in the quadrature rule (3.3). The set of nearest neighbors, in a specifically transformed space discussed in section 3.2, to \mathbf{y}_j , is used to compute Leja points and will be notated as $\widehat{\mathfrak{N}}$. We will call the Leja points $\{\boldsymbol{\eta}_1, \dots, \boldsymbol{\eta}_m\}$, but we will suppress the j dependence since the procedure updates one mesh point at a time (eg. $\widehat{\mathfrak{N}} = \widehat{\mathfrak{N}}^j$ and $\{\boldsymbol{\eta}_1, \dots, \boldsymbol{\eta}_m\} = \{\boldsymbol{\eta}_1, \dots, \boldsymbol{\eta}_m\}_j$).

Now we will update a member of the global mesh \mathbf{y}_j .

$$\begin{aligned} \tilde{p}(\mathbf{y}_j, t_n) &= \int_{\mathbb{R}^N} G(\mathbf{y}_j, \mathbf{y}) \tilde{p}(\mathbf{y}, t_n) d\mathbf{y} \\ &= \int_{\mathbb{R}^N} r(\mathbf{y}) \mathcal{N}(\mathbf{y}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) d\mathbf{y} \end{aligned} \quad (7)$$

$$\approx \sum_{i=1}^m r(\boldsymbol{\eta}_i) w_i \quad (8)$$

where

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{\pi^N |\boldsymbol{\Sigma}|}} \exp\left(-(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right).$$

In section 3.1, we describe how we effect the integral in (7) by using a Laplace approximation of the integrand to identify $\mathcal{N}(\cdot; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ and r . These values differ for different \mathbf{y}_j (e.g., $\boldsymbol{\mu} = \boldsymbol{\mu}_j$, $\boldsymbol{\Sigma} = \boldsymbol{\Sigma}_j$, $r = r_j$, but we again suppress this j dependence). Section 3.2 subsequently details how we identify the quadrature rule and weights in (8), and section 3.3 covers the selection of Leja points, $\{\boldsymbol{\eta}_1, \dots, \boldsymbol{\eta}_m\}$.

3.1 Laplace Approximation via Least Squares

In this section, we describe how r and \mathcal{N} in (7) are identified. In short, we identify \mathcal{N} as a Laplace approximation to the integrand of equation (5); we implement this practically by performing a local least-squares quadratic fit to the log-integrand using nearby data.

The Laplace approximation around a point \mathbf{y}_j is computed using the Leja points $\{\boldsymbol{\eta}_1, \dots, \boldsymbol{\eta}_m\}$ if they are known from a previous time step, or a set of nearest neighbor points \mathfrak{N} when Leja points are not yet known. Initially, we do not know Leja points, so for the first time step, all points in the mesh use their corresponding \mathfrak{N} set; however, Leja points are more common to use once they are known in future time steps. In this section, we will assume the use of $\{\boldsymbol{\eta}_1, \dots, \boldsymbol{\eta}_m\}$; however, the procedure is equivalent if the nearest neighbor points are used instead.

Let the i^{th} component of the vector $\boldsymbol{\psi}$ be given as

$$\boldsymbol{\psi}^{(i)} := -\log(G(\mathbf{y}_j, \boldsymbol{\eta}_i) \widehat{p}(\boldsymbol{\eta}_i, t_n))$$

for $i = 1, \dots, m$ so that $\boldsymbol{\psi}$ is an $m \times 1$ vector. The Laplace approximation would model this log-integrand as a quadratic polynomial,

$$\boldsymbol{\psi}^{(i)} \approx \tilde{\boldsymbol{\psi}}(\boldsymbol{\eta}_i) := c + \mathbf{b}^T \boldsymbol{\eta}_i + (\boldsymbol{\eta}_i)^T \mathbf{A} \boldsymbol{\eta}_i, \quad (9)$$

for a scalar c , vector $\mathbf{b} \in \mathbb{R}^N$, and a symmetric matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ that we identify via least-squares polynomial approximation. To describe this procedure, we require more notation. With $\alpha \in \mathbb{N}_0^N$ a multi-index, we use the standard convention,

$$\alpha = (\alpha_1, \dots, \alpha_N), \quad |\alpha| := \sum_{j=1}^N \alpha_j, \quad \boldsymbol{\eta}^\alpha = \prod_{j=1}^N (\eta^{(j)})^{\alpha_j},$$

with $\boldsymbol{\eta} = (\eta^{(1)}, \dots, \eta^{(N)})^T$. Then, define

$$P_k := \text{span} \{ \boldsymbol{\eta}^\alpha \mid \alpha \in \Lambda_k \}, \quad r_k = |\Lambda_k| = \dim P,$$

where we take $\Lambda_k \in \mathbb{N}_0^N$ to be the set of multi-indices corresponding to degree- k approximation,

$$\Lambda_k := \{\alpha \in \mathbb{N}_0^N \mid |\alpha| \leq k\}, \quad r_k = \binom{N+k}{N}$$

We will perform a quadratic fit with $k = 2$. With $\alpha^{(1)}, \dots, \alpha^{(r_2)}$ an enumeration of the elements of Λ_2 , then with the Vandermonde matrix, $\mathbf{M} \in \mathbb{R}^{m \times r_2}$ is defined as

$$(\mathbf{M})_{i,j} = \eta_i^{\alpha^{(j)}},$$

a least-squares fit to the data ψ is the emulator,

$$q(\boldsymbol{\eta}) = \sum_{j=1}^{r_2} h_j \boldsymbol{\eta}^{\alpha^{(j)}}, \quad \mathbf{h} = (h_1, \dots, h_{r_2})^T,$$

where \mathbf{h} is given as the least-squares solution to the linear system,

$$\mathbf{M}\mathbf{h} = \psi.$$

Once the coefficients \mathbf{h} are computed, we translate q into the symmetric quadratic form (9) using the following identification of the entries of c , \mathbf{b} and \mathbf{A} :

$$c = h_{i(0)}, \quad b_j = h_{i(e_j)}, \quad A_{j,\ell} = \frac{1}{2 - \delta_{j,\ell}} h_{i(e_j + e_\ell)}$$

where $\delta_{j,\ell}$ is the Kronecker delta, $e_j \in \mathbb{N}_0^N$ is the cardinal unit vector in direction j with entry 1 in location j and zeros elsewhere, and $i(\alpha)$ is a function that returns the linear index in Λ_2 associated to α ,

$$i = i(\alpha) \implies \alpha = \alpha^{(i)}.$$

In order to associate this quadratic fit with a normal distribution, the matrix \mathbf{A} must be positive-definite. We will explain in section 3.1.1 how we address situations when \mathbf{A} is not positive-definite. However, when \mathbf{A} is positive-definite, we have the following immediate identification of a normal distribution density \mathcal{N} from this quadratic fit to the log-integrand:

Proposition 1. *If \mathbf{A} in (9) is positive-definite, then*

$$\exp(-\tilde{\psi}(\mathbf{x})) = \mathcal{CN}(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) \sqrt{\pi^N |\boldsymbol{\Sigma}|},$$

where

$$\boldsymbol{\mu} = -\frac{1}{2} \mathbf{U} \boldsymbol{\Lambda}^{-1} \mathbf{d} \quad \boldsymbol{\Sigma}^{-1} = \mathbf{A} \quad C = \exp(-c + \frac{1}{4} \mathbf{d}^T \boldsymbol{\Lambda}^{-1} \mathbf{d}) \quad (10)$$

Proof. Since \mathbf{A} is symmetric and positive-definite, it has an orthogonal diagonalization

$$\mathbf{A} = \mathbf{U} \boldsymbol{\Lambda} \mathbf{U}^T, \quad \mathbf{U} \mathbf{U}^T = \mathbf{I}, \quad \boldsymbol{\Lambda} = \text{diag}(\lambda_1, \dots, \lambda_d).$$

with positive eigenvalues $\lambda_j > 0$ for all j . Defining $\boldsymbol{\gamma} := \mathbf{U}^T \mathbf{x}$, then

$$\tilde{\psi}(\mathbf{x}) = -(c + \mathbf{d}^T \boldsymbol{\gamma} + (\boldsymbol{\gamma})^T \mathbf{A} \boldsymbol{\gamma})$$

With $d^{(i)}, \gamma^{(i)}$ the components of $\mathbf{d}, \boldsymbol{\gamma}$, a rearrangement yields

$$\begin{aligned}
-\tilde{\boldsymbol{\psi}}^{(i)} &= c + \sum_{j=1}^N \left(d^{(j)} \gamma^{(j)} + \lambda_j (\gamma^{(j)})^2 \right) \\
&= c - \sum_{j=1}^N \frac{(d^{(j)})^2}{4\lambda_j} + \sum_{j=1}^N \left(\sqrt{\lambda_j} \gamma^{(j)} + \frac{d^{(j)}}{2\sqrt{\lambda_j}} \right)^2 \\
&= c - \frac{1}{4} \mathbf{d}^T \boldsymbol{\Lambda}^{-1} \mathbf{d} + \left(\boldsymbol{\gamma} + \frac{1}{2} \boldsymbol{\Lambda}^{-1} \mathbf{d} \right)^T \boldsymbol{\Lambda} \left(\boldsymbol{\gamma} + \frac{1}{2} \boldsymbol{\Lambda}^{-1} \mathbf{d} \right) \\
&= c - \frac{1}{4} \mathbf{d}^T \boldsymbol{\Lambda}^{-1} \mathbf{d} + \left(\boldsymbol{\eta} + \frac{1}{2} \mathbf{U} \boldsymbol{\Lambda}^{-1} \mathbf{d} \right)^T \mathbf{A} \left(\boldsymbol{\eta} + \frac{1}{2} \mathbf{U} \boldsymbol{\Lambda}^{-1} \mathbf{d} \right),
\end{aligned}$$

so that

$$\exp(-\tilde{\boldsymbol{\psi}}^{(i)}) = C \exp\left(-(\boldsymbol{\eta}_i - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{\eta}_i - \boldsymbol{\mu})\right),$$

with $\boldsymbol{\mu}, \boldsymbol{\Sigma}$, and C as given in (10). □

Using this identification of \mathcal{N} using the least-squares fit, we accomplish the equality in (7) as

$$\tilde{p}(\mathbf{y}_j, t_{n+1}) = \int_{\mathbb{R}^N} r(\mathbf{y}) \mathcal{N}(\mathbf{y}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) d\mathbf{y}, \quad r(\mathbf{y}) = \frac{G(\mathbf{y}_j, \mathbf{y}) \tilde{p}(\mathbf{y}, t_n)}{\mathcal{N}(\mathbf{y}; \boldsymbol{\mu}, \boldsymbol{\Sigma})}. \quad (11)$$

3.1.1 Alternative Method

In some situations we cannot use the above Laplace approximation procedure. For example, Proposition 1 requires that \mathbf{A} be positive-definite, which may not occur in practice, particularly when the integrand $G(\mathbf{y}_j, \boldsymbol{\eta}_i) \hat{p}(\boldsymbol{\eta}_i, t_n)$ is not locally Gaussian. When \mathbf{A} is not positive-definite, we must use the alternative method shown in equation (12). Additionally, we use the alternative method when the quadrature rule is ill-conditioned. In practice, the alternative method is typically used at or near the mesh boundary when an insufficient number of points are around the point being updated, \mathbf{y}_j . Use of the alternative method varies, but it usually is only used for around 1-2% of mesh points per time step on average in two-dimensional problems. We expect this usage would remain low in higher dimensions.

We wish to take advantage of the structure of $G(\mathbf{x}, \mathbf{y})$ to procure a weight function so that

$$\begin{aligned}
\tilde{p}(\mathbf{y}_j, t_{n+1}) &= \int_{\mathbb{R}^N} r(\mathbf{y}) \mathcal{N}(\mathbf{y}; \mathbf{y}_j + h\mathbf{f}(\mathbf{y}_j), h\mathbf{g}(\mathbf{y}_j)\mathbf{g}(\mathbf{y}_j)^T) d\mathbf{y} \\
r(\mathbf{y}) &= \frac{G(\mathbf{y}_j, \mathbf{y}) \tilde{p}(\mathbf{y}, t_n)}{\mathcal{N}(\mathbf{y}; \mathbf{y}_j + h\mathbf{f}(\mathbf{y}_j), h\mathbf{g}(\mathbf{y}_j)\mathbf{g}(\mathbf{y}_j)^T)}. \quad (12)
\end{aligned}$$

The Gaussian used for the weight function has a mean and variance that depends only on the current point we are updating, \mathbf{y}_j . Since, in practice, this alternative method is used primarily for points on or near the boundary, the density \hat{p} should be relatively flat, making this simplified procedure sufficient. If the calculated value using this procedure is negative, we replace it with the minimum value of \hat{p} at the previous time step to maintain positivity of \hat{p} .

3.2 Quadrature Weights

This section describes how the quadrature rule in (8) is generated, assuming the nodes $\{\boldsymbol{\eta}_i\}_{i=1}^m$ are provided. Section 3.3 later describes the more complex procedure of how the nodes are chosen.

For each point \mathbf{y}_j in the global mesh, the Laplace approximation of section 3.1 allows us to write the integral for the update of \hat{p} at \mathbf{y}_j as in (7). We now discretize this integral with a quadrature rule,

$$\int_{\mathbb{R}^N} r(\mathbf{y}) \mathcal{N}(\mathbf{y}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) d\mathbf{y} \approx \sum_{i=1}^m r(\boldsymbol{\eta}_i) w_i.$$

First, let $\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^T$ be any decomposition of $\boldsymbol{\Sigma}$ (e.g., through the Cholesky decomposition). Then, the integral (7) can be rewritten as

$$\int_{\mathbb{R}^N} r(\mathbf{y}) \mathcal{N}(\mathbf{y}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) d\mathbf{y} \stackrel{\mathbf{y}=\mathbf{L}\boldsymbol{\zeta}+\boldsymbol{\mu}}{=} \int_{\mathbb{R}^N} r(\mathbf{L}\boldsymbol{\zeta} + \boldsymbol{\mu}) \mathcal{N}(\boldsymbol{\zeta}, \mathbf{0}, \mathbf{I}) d\boldsymbol{\zeta},$$

and under this same map, we define quadrature nodes $\{\boldsymbol{\eta}_i\}_{i=1}^m$ which are in \mathbf{y} space as

$$\boldsymbol{\eta}_i = \mathbf{L}\boldsymbol{\zeta}_i + \boldsymbol{\mu} \quad (13)$$

where $\{\boldsymbol{\zeta}_i\}_{i=1}^m$ are nodes in $\boldsymbol{\zeta}$ space. The weights w_i of the quadrature rule are chosen as the interpolatory weights associated to a particular polynomial space. The nodes $\{\boldsymbol{\zeta}_i\}_{i=1}^m$ are chosen in a way that guarantees unisolvence of a polynomial interpolation problem, i.e., we can construct a unique polynomial $Q \in P$ such that

$$r(\mathbf{L}\boldsymbol{\zeta}_i + \boldsymbol{\mu}) = Q(\boldsymbol{\zeta}_i), \quad i \in [m].$$

More precisely, let $\{\phi_i\}_{i=1}^{m_k}$ be a basis for the degree- k polynomial space P_k , so that

$$Q(\boldsymbol{\zeta}) = \sum_{j=1}^{m_k} \hat{c}_j \phi_j(\boldsymbol{\zeta}),$$

where $\hat{\mathbf{c}} = (\hat{c}_1, \dots, \hat{c}_{m_k})^T$ solves the linear system, where we select $m = m_k$,

$$\mathbf{V}\hat{\mathbf{c}} = \mathbf{r}, \quad \mathbf{V}_{i,j} = \phi_j(\boldsymbol{\zeta}_i), \quad \mathbf{V} \in \mathbb{R}^{m \times m},$$

and where $\mathbf{r} = (r(\boldsymbol{\eta}_1), \dots, r(\boldsymbol{\eta}_m))^T$. We generate the quadrature weights as exact integration of q in place of r :

$$\int r(\mathbf{L}\boldsymbol{\zeta} + \boldsymbol{\mu}) \mathcal{N}(\boldsymbol{\zeta}, \mathbf{0}, \mathbf{I}) d\boldsymbol{\zeta} \approx \int q(\boldsymbol{\zeta}) \mathcal{N}(\boldsymbol{\zeta}, \mathbf{0}, \mathbf{I}) d\boldsymbol{\zeta} = \sum_{i=1}^m r(\boldsymbol{\eta}_i) w_i, \quad (14)$$

where w_i are given by

$$\mathbf{w} = (w_1, \dots, w_N) = \boldsymbol{\xi}^T \mathbf{V}^{-1}, \quad \boldsymbol{\xi}_j := \int \phi_j(\boldsymbol{\zeta}) \mathcal{N}(\boldsymbol{\zeta}, \mathbf{0}, \mathbf{I}) d\boldsymbol{\zeta}. \quad (15)$$

The expression for \mathbf{w} can be somewhat simplified computationally if we choose the basis ϕ_i as a family of polynomials that are L^2 -orthogonal under the weight function \mathcal{N} . Since this weight is a Gaussian, the appropriate orthonormal polynomial family are (normalized and tensorized) Hermite polynomials. In particular, with $\alpha^{(1)}, \dots, \alpha^{(m_k)}$ an(y) enumeration of the elements of Λ that satisfies $|\alpha^{(j)}| \leq |\alpha^{(j+1)}|$, then we consider the basis,

$$\phi_j(\boldsymbol{\zeta}) = \prod_{i=1}^N \hat{h}_{\alpha_i^{(j)}}(\boldsymbol{\zeta}_i), \quad (16)$$

where $\widehat{h}_\ell(\cdot)$ is the degree- ℓ normalized univariate Hermite polynomial, satisfying the orthogonality condition,

$$\int_{\mathbb{R}} \widehat{h}_\ell(\zeta) \widehat{h}_k(\zeta) \mathcal{N}(\zeta, 0, 1) d\zeta = \delta_{\ell,k}, \quad \deg \widehat{h}_\ell = \ell,$$

where $\delta_{i,j}$ is the Kronecker delta. The uniqueness of each \widehat{h}_ℓ is assured if we insist that the leading coefficient is positive. Since $\mathcal{N}(\cdot, 0, 1)$ is a probability density, $\widehat{h}_0(\zeta) \equiv 1$. The chosen basis (16) for P_k then satisfies the following multivariate orthogonality condition,

$$\int_{\mathbb{R}^N} \phi_j(\boldsymbol{\zeta}) \phi_i(\boldsymbol{\zeta}) \mathcal{N}(\boldsymbol{\zeta}, \mathbf{0}, \mathbf{I}) d\boldsymbol{\zeta} = \delta_{i,j}, \quad \phi_1(\boldsymbol{\zeta}) \equiv 1, \quad (17)$$

so that the moments $\boldsymbol{\xi}$ in (15) are given by $\boldsymbol{\xi}_j = \delta_{j,1}$. Therefore, with this basis, the formula (15) implies that the quadrature weights are simply given as the first row of \mathbf{V}^{-1} ,

$$\mathbf{w}^T = (\mathbf{V}^{-1})_{1,\cdot}. \quad (18)$$

In terms of the quadrature nodes $\{\boldsymbol{\eta}_i\}_{i=1}^m$, we connect ω_i from (6) to w_i as

$$\omega_i = \frac{r(\boldsymbol{\eta}_i)}{G(\boldsymbol{\eta}_j, \boldsymbol{\eta}_i) \widehat{p}(\boldsymbol{\eta}_i, t_n)} w_i.$$

3.3 (Weighted) Leja Sequences

We describe the selection of quadrature nodes $\{\boldsymbol{\eta}_i\}_{i=1}^m$. In the following discussion, fix a global mesh index j and recall that for a point \mathbf{y}_j in the global mesh, $\widehat{\mathfrak{N}}$ denotes the set of nearest neighbors to \mathbf{y}_j in transformed space. In general, the neighbors $\widehat{\mathfrak{N}}$ are not directly related to the points in \mathfrak{N} , and the number of points used for each set usually differs. We can now define how this modified set is identified. With $(\mathbf{L}, \boldsymbol{\mu})$ the affine map found via Laplace approximation, the global mesh is transformed, i.e., defining $\mathbf{v}_q := \mathbf{L}^{-1}(\mathbf{y}_q - \boldsymbol{\mu})$, then \mathfrak{Z} is defined as the M nearest neighbors to \mathbf{v}_j (counting itself). $\widehat{\mathfrak{N}}$ is defined through the affine map in (13),

$$\widehat{\mathfrak{N}} = \mathbf{L}\mathfrak{Z} + \boldsymbol{\mu} := \{\mathbf{L}\boldsymbol{\zeta} + \boldsymbol{\mu} \mid \boldsymbol{\zeta} \in \mathfrak{Z}\}.$$

Given this nearest neighbor set \mathfrak{Z} , our goal will be to identify a subset of nodes $\boldsymbol{\zeta}_i$, $i \in [m]$, from this set, which defines $\boldsymbol{\eta}_i$ through (13), and which are used in the approximation (14). Recall at time zero and when Leja points are not known, we use the set of nearest neighbor points \mathfrak{N} introduced in section 3.1 for the Laplace approximation to recover $\boldsymbol{\mu}$ and \mathbf{L} for the transformation.

We suppress notational dependence on j (the global mesh index) in the remaining discussion of this section. The m quadrature points $\boldsymbol{\zeta}_i$, $i \in [m]$, are a subset of \mathfrak{Z} and are computed as discrete weighted Leja sequence from \mathfrak{Z} . To formalize this connection, we first describe Leja sequences, along with weighted and discrete versions. Leja sequences are unstructured, flexible, and nested, and discrete versions are easily calculated.

In one dimension on a compact interval $[a, b]$, a Leja sequence is classically defined as any sequence of points $\zeta_n \in [a, b] \subset \mathbb{R}$ for $n = 1, 2, \dots$ that solve the sequential optimization problem,

$$\zeta_{s+1} = \operatorname{argmax}_{\zeta \in [a,b]} \prod_{i=1}^s |\zeta - \zeta_i|, \quad (19)$$

where ζ_1 is arbitrarily chosen in the interval $[a, b]$ [23, 24]. Leja sequences are not unique due to the choice of the initial point ζ_0 as well as the potential for multiple maximizers at each step of (19). Sequences chosen in this way empirically have good approximation properties. In particular, they

form polynomial interpolants of good quality and empirically yield accurate interpolatory quadrature rules.

In the one-dimensional setting, we are interested in identifying nodes whose corresponding interpolatory quadrature rule is an accurate approximation of the form (14). To accomplish this, we must consider integrals over the entire real line with respect to the Gaussian weight $\mathcal{N}(\zeta, 0, 1)$. A naive extension of the optimization (19) that replaces $[a, b]$ by \mathbb{R} is not well defined, so we adopt the strategy from [25] that uses a particular type of weighted Leja sequence, and shows that these sequences have accurate interpolatory quadrature rules with respect to the weight function.

For notational convenience, we define $w(\zeta) = \mathcal{N}(\zeta, 0, 1)$, and let ζ_i for $i = 1, 2, \dots$ be any sequence of solutions to a modified version of (19),

$$\zeta_{s+1} = \operatorname{argmax}_{\zeta \in \mathbb{R}} \sqrt{w(\zeta)} \prod_{i=1}^s |\zeta - \zeta_i|, \quad (20)$$

where ζ_1 is chosen arbitrarily as an initial point. The use of the weight function penalizes the selection of points at infinity. The use of \sqrt{w} is an appropriate choice is motivated by the fact that weighted Leja sequences defined by (20) satisfy the *asymptotic Fekete* property, and asymptotically distribute like w -Gaussian quadrature rules [25]. Empirically, these sequences also form stable quadrature rules for approximating w -weighted integrals.

We cannot directly use (20) in our framework because we do not have the freedom to choose points arbitrarily. Instead, we pose the optimization problem as one not over the continuum \mathbb{R} but instead over the discrete set \mathfrak{Z} , which are (mapped) nearest neighbors around a global mesh point. We therefore construct the following discrete, weighted Leja sequence:

$$\zeta_{s+1} = \operatorname{argmax}_{\zeta \in \mathfrak{Z}} \sqrt{w(\zeta)} \prod_{i=1}^s |\zeta - \zeta_i|. \quad (21)$$

Ideally, the candidate set \mathfrak{Z} should form a so-called weakly admissible mesh so that the points sufficiently cover the domain of interest [26, 27, 28]. The above discussion holds for one dimension, but the objective function being maximized in (21) does not directly generalize to higher dimensions. However, we can rewrite the one-dimensional problem in a form that can be extended to the multivariate case.

The calculation of weighted discrete Leja sequences in (21) can be simplified. It is possible to show that (21) is equivalent to constructing a Vandermonde-like matrix via a particular kind of greedy determinant maximization [29] which reduces the process into a simple numerical linear algebra problem. The sequence in (21) can be computed from the pivots of a row-pivoted LU factorization on a Vandermonde-like matrix. In particular, let $\tilde{\mathbf{V}} \in \mathbb{R}^{M \times m}$ denote a weighted Vandermonde-like matrix on the candidate points \mathfrak{Z} ,

$$\tilde{\mathbf{V}}_{\ell, q} = \sqrt{w(z_\ell)} \phi_q(z_\ell), \quad \mathfrak{Z} = \{z_1, \dots, z_M\}, \quad (22)$$

where $\{\phi_q\}_{q=1}^m$ is any ordered basis satisfying $P_k = \operatorname{span}\{\phi_q, q \in [k]\}$. We choose ϕ_q as the $w(\cdot)$ -orthonormal Hermite polynomials from (17). With $\mathbf{P}\tilde{\mathbf{V}} = \tilde{\mathbf{L}}\tilde{\mathbf{U}}$ the pivoted LU decomposition of $\tilde{\mathbf{V}}$, a solution to (21) for $1 \leq i \leq m$ is given by the first m points of the \mathbf{P} -permuted points in \mathfrak{Z} :

$$\zeta_i = z_{p_i}, \quad (p_1, \dots, p_M)^T = \mathbf{P}(1, 2, \dots, M)^T. \quad (23)$$

Because of this equivalence between the solution to (21) and pivoted linear algebra, in practice we compute equation (20) via an LU factorization with partial row pivoting of a Vandermonde-like matrix [29].

Note that this linear algebraic procedure is directly generalizable to multiple dimensions. In dimension $N > 1$, defining $\tilde{\mathbf{V}}$ as in (22) (with z_j replaced with the N -dimensional points \mathbf{z}_j from \mathfrak{Z} , and ϕ_q the P_k -orthonormal basis from (17)), we can again accomplish a greedy determinant

maximization procedure implemented via the pivoted LU approach described above. We define the points ζ_i from (23) as the weighted discrete Leja sequence that we use for quadrature nodes in the approximation (14), which we emphasize are computable by a simple LU decomposition on a weighted Vandermonde-like matrix on \mathfrak{Z} .

For the overall DTQ procedure, the procedure above must be repeated for every point in the global mesh (i.e., for every global index j). Our computation of these discrete weighted Leja sequences makes use of the PyApprox package [30].

3.3.1 Leja Point Reuse

Although the Laplace-approximated affine map parameters $(\mathbf{L}, \boldsymbol{\mu})$ are recomputed at every time step, to save computational effort, we recompute Leja sequences only if a stability condition is violated. Leja points are reused from time step to time step as long as the condition number

$$\Gamma := \|\mathbf{w}\|_1 < 1 + \epsilon,$$

where $\|\cdot\|_1$ is the ℓ^1 norm on vectors, \mathbf{w} are the interpolatory quadrature weights from (18), and ϵ is a tunable parameter. In practice, the number of mesh points on which we can reuse Leja sequences from the previous time step, depends on the drift and diffusion. However, we find that we are often able to reuse Leja points from the previous time step which increases the speed of the algorithm substantially. We will quantify Leja sequence reuse in the results section.

4 DTQ with an Adaptive Mesh

In this section, we describe the adaptive part of the procedure, which updates the mesh based on the density as time evolves. We outline the overall Adaptive DTQ method in Algorithm 1.

Algorithm 1 Adaptive DTQ

```

1: procedure ADAPTIVEDTQ:
2:   while step forward do:
3:     add points to mesh boundary if needed (section 4.2)
4:     remove points from mesh boundary if needed (section 4.3)
5:     for each mesh point do:
6:       attempt local quadratic fit (section 3.1)
7:       if quadratic fit is successful then:
8:         locate Leja points (reuse or compute) (section 3.3)
9:         compute quadrature weights (section 3.2)
10:        step forward in time using (8) with  $r$  from (11)
11:      else:
12:        use alternative procedure (section 3.1.1)
13:        step forward in time using (8) with  $r$  from (12)

```

The adaptive procedure attempts to reduce the number of mesh points required to compute $\hat{p}(\mathbf{x}, t)$ by adaptively updating the mesh as the solution evolves in time. The basic idea is that we form a mesh that covers most of the support/mass of the solution. Figure 1 gives a visual example of how the mesh is updated to track the density.

4.1 Identifying the mesh “boundary”

The adaptive procedure operates only on the boundary of the mesh, removing or adding points as appropriate. In order to identify the “boundary” of the mesh, we utilize a procedure that combines a mesh triangulation and *alpha shape* procedure. First, we construct a Delaunay triangulation of the

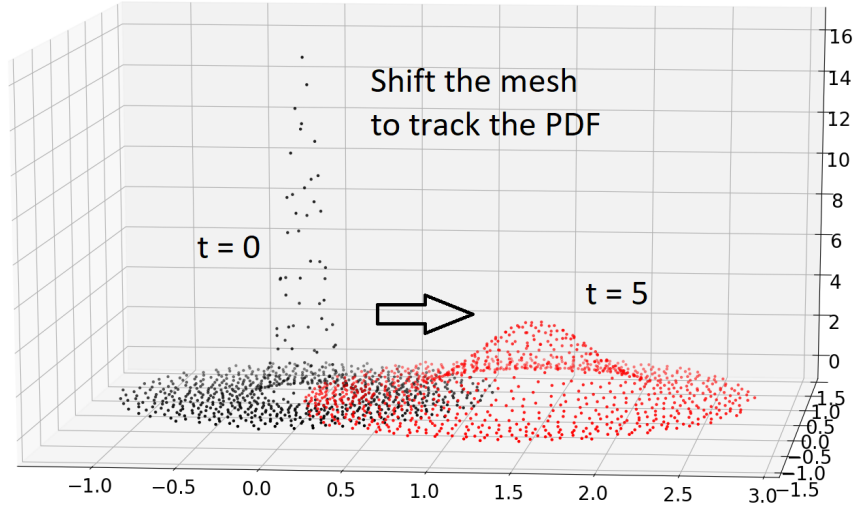


Figure 1: Shifting the mesh to keep track of the density is achieved through adding and removing mesh points.

global mesh $\{\mathbf{y}_j\}_{j=1}^s$, which is a triangulation satisfying the condition that no mesh point lies in the interior of any circumscribing circle of any triangle in the triangulation. The Delaunay triangulation ensures that the minimum angle is maximized for all the triangles in the triangulation to avoid sliver triangles [31]. The alpha shape algorithm recovers the boundary points of a point mesh in Algorithm 2. A survey of alpha shapes from Edelsbrunner is available for more information [32]. For the procedure, we select $\hat{\alpha} = \Lambda$, which is the enforced maximum distance between points in the mesh. Then, for each simplex found using the Delaunay triangulation, the radius of the circumcircle is computed. If the radius is less than $\hat{\alpha}$, then we know the two points associated with that edge are boundary points.

Algorithm 2 AlphaShape

```

1: procedure ALPHASHAPE( $\hat{\alpha}$ ):
2:   procedure ADDEGE(boundaryEdgesSet, edgeToAdd):
3:     if edgeToAdd not in boundaryEdgesSet then:
4:       append edgeToAdd to boundaryEdgesSet
5:     return boundaryEdgesSet
6:   initialize boundaryEdgesSet
7:   for each simplex do:
8:     compute the radius of the simplex circumcircle
9:     if radius <  $\hat{\alpha}$  then:
10:      for each simplex edge do:
11:        boundaryEdges = AddEdge(boundaryEdges, simplexEdge)
12:   return boundaryEdgesSet

```

4.2 Adding Boundary Points

We assume the mesh boundary is identified as described in the previous section. We add mesh points based on a prescribed tolerance parameter, β , which is tuned to maintain a sufficiently small value

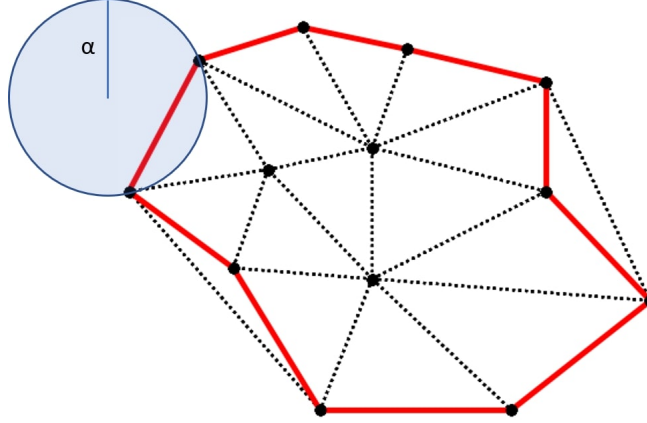


Figure 2: A depiction of a completed two-dimensional alpha shape procedure. The Delaunay triangulation is shown, and the red edges indicate the boundary that is identified.

of \hat{p} on the mesh boundary. For each boundary point where \hat{p} is larger than $10^{-\beta}$, candidate points are generated as θ equispaced points on a circle (in two dimensions) or higher order N -dimensional sphere of radius $(\lambda + \Lambda)/2$. Here, θ is the number of equispaced points that are candidates for adding to the mesh. For a candidate point to be added to the mesh, the distance to the closest point in the mesh must be greater than a set value λ and it must be less than a set value Λ . Essentially, λ and Λ are the enforced minimum and maximum distances, respectively, between mesh points. After all new points to add to the mesh are determined, the density values \hat{p} are computed via local, linear interpolation. If the new point is outside the convex hull of the current mesh or interpolates to a negative value, we assign it the value of the minimum of \hat{p} from the previous time step. This procedure for adding points is typically run at every time step.

4.3 Removing Boundary and Interior Points

We remove the mesh points that are deemed unnecessary in terms of the value of the density \hat{p} . Points are removed based on the prescribed tolerance, β from section 4.2, which is used to maintain a sufficiently small value of \hat{p} at the boundary. More specifically, all mesh points that are smaller than $10^{-\beta-0.5}$ are removed from the mesh. This procedure for adding points is typically run periodically; however, not at every time step.

5 Results

The code for the adaptive DTQ procedure and examples is located on GitHub [33]. We demonstrate Algorithm 1 in this section on several two-dimensional ($N = 2$) examples. We first describe some experimental setup characteristics that are common among all examples. The initial condition is taken to be a Dirac mass centered at the origin and a radial initial mesh centered at the origin comprised of concentric circles with radii that are multiples of the enforced minimum distance between points, λ , and ranges up to a radius $\mathcal{R} - (\mathcal{R} \bmod \lambda)$, where \mathcal{R} is a parameter defining the initial mesh radius.

When adding new points in these examples, we use $\theta = 8$ so that 8 candidate samples are considered for adding to the mesh. The threshold for Leja point reuse is $\epsilon = 0.1$, and the threshold of the condition number for using the alternative method is set as 8. We use $m = 10$ Leja points for the quadrature rule (choosing the index set Λ to correspond to polynomials of degree $k = 3$ in $N = 2$ dimensions). The number of nearest neighbors in the set \mathfrak{N} used for the Laplace approximation is 20, and the number of nearest neighbors in the set $\hat{\mathfrak{N}}$ used for selecting Leja points is 40.

We report errors in the experiments below, which are measured in the following ways. Let p be the exact solution and \hat{p} be the computed (adaptive DTQ) solution. With s the number of points in the mesh, define the following spatial errors at a fixed time:

$$\begin{aligned} L_{2w} &= \sqrt{\frac{1}{\sum_{j=1}^s p(\mathbf{x}_j)} \sum_{j=1}^s \left((p(\mathbf{x}_j) - \hat{p}(\mathbf{x}_j))^2 p(\mathbf{x}_j) \right)} \\ L_2 &= \sqrt{\frac{1}{m} \sum_{j=1}^s (p(\mathbf{x}_j) - \hat{p}(\mathbf{x}_j))^2} \\ L_1 &= \frac{1}{m} \sum_{j=1}^s |p(\mathbf{x}_j) - \hat{p}(\mathbf{x}_j)| \\ L_\infty &= \max_{j \in [s]} |p(\mathbf{x}_j) - \hat{p}(\mathbf{x}_j)|, \end{aligned}$$

which are, respectively, approximations of the $L_p^2(\mathbb{R}^N)$, $L^2(\mathbb{R}^N)$, $L^1(\mathbb{R}^N)$, and $L^\infty(\mathbb{R}^N)$ norms.

5.1 Moving Hill Example

Consider the solution to the SDE in equation (1) with constant drift and diffusion,

$$\mathbf{f} = \begin{pmatrix} A \\ 0 \end{pmatrix}, \quad \mathbf{g} = \begin{pmatrix} B & 0 \\ 0 & B \end{pmatrix},$$

This SDE corresponds to the Fokker-Planck PDE

$$\frac{\partial p(\mathbf{x}, t)}{\partial t} = -A \frac{\partial}{\partial x} p(\mathbf{x}, t) + \frac{B^2}{2} \frac{\partial^2}{\partial (x^{(1)})^2} p(\mathbf{x}, t) + \frac{B^2}{2} \frac{\partial^2}{\partial (x^{(2)})^2} p(\mathbf{x}, t)$$

which has the exact solution

$$p(\mathbf{x}, t) = \frac{1}{4\pi Bt} \exp\left(\frac{-(x^{(1)} - A)^2 + (x^{(2)})^2}{4Bt}\right)$$

We set simulation parameters as $(\lambda, \Lambda, \mathcal{R}, h) = (0.15, 0.17, 1.5, 0.01)$. Recall that λ, Λ are mesh spacing parameters, \mathcal{R} is the initial mesh radius, and h is the temporal step size.

5.1.1 β Parameter

We explore the effect of the β parameter, which from section 4.2 is the tolerance parameter for allowable density values on boundary nodes. The adaptive DTQ approach error values for time $t = 1.15$ for varying β are shown in Table 1.

We observe that β has a somewhat direct control on error of the approach for this simple example: For larger β , more points are used to form a mesh on a larger region, which improves the overall accuracy of the method. Table 1 shows that as β increases, the error tends to decrease. Thus, the adaptive DTQ method can be quite accurate if β is chosen appropriately. However, the selection of β must be balanced with the computational cost associated with a larger mesh. The number of mesh points s at the last time step are also shown in Table 1.

5.1.2 Adaptive vs Tensorized DTQ

We compared the computational timing for a two-dimensional tensorized DTQ procedure (section 2.2) against the adaptive DTQ procedure for this example. All timings were recorded relative to the time the adaptive DTQ procedure takes to run, indicated by the star in Figure 3. The hardware

β	L_{2w} Error	L_2 Error	L_1 Error	L_∞ Error	# Points s
1	2.450786e-02	2.206468e-02	1.970326e-03	8.008655e-02	179
2	1.581447e-03	2.460677e-03	4.913179e-05	1.170814e-02	717
3	8.981020e-05	1.458679e-04	2.454267e-06	5.253691e-04	1335
4	1.160785e-05	1.480656e-05	2.243876e-07	5.567885e-05	1913
5	1.322241e-06	1.535408e-06	1.897660e-08	4.314279e-06	2486
6	1.259077e-07	1.485520e-07	1.413359e-09	2.896678e-07	3134
7	1.275439e-08	1.629120e-08	1.252540e-10	2.654943e-08	3674
8	7.997140e-10	1.747622e-09	6.116165e-12	2.734659e-09	4238
9	9.476715e-11	1.808187e-10	5.943385e-13	2.850035e-10	4796
10	2.295830e-11	2.162911e-11	1.158005e-13	7.404828e-11	5359

Table 1: Adaptive DTQ errors for different values of the boundary tolerance parameter β at time $t = 1.15$ for the moving hill example of section 5.1. Also shown are the number of points s in the adaptively formed mesh at $t = 1.15$. Generally, as β increases, the error decreases because we use more points to cover a larger area of the domain.

for these timings used an AMD EPYC 7702P 64-Core Processor with 255 GiB of system memory. We ran each adaptive and tensorized method scenario five times and reported the average time. We varied the accuracy and cost for these two methods by changing a parameter. For the adaptive procedure, we adjusted the β parameter, and for the tensorized method we adjusted the equispaced spatial step size κ from section 2.2. The minimum and maximum values in each dimension used by the mesh points in the most accurate adaptive procedure shown in Figure 3 were used to generate the rectangular domain with essential support for the tensorized procedure.

Figure 3 shows that, for similar effort (relative time), the tensorized procedure achieves 10^{-2} error versus 10^{-6} error for the adaptive procedure. To achieve approximately 10^{-2} error, the tensorized procedure requires about 10 times more computational effort than the adaptive DTQ for this example. We emphasize that this example is very simple, and we expect adaptive DTQ to be quicker and even more accurate than the tensorized approach for more complicated examples.

5.2 Four hills example

We consider the solution to the SDE (1) in two dimensions with drift and constant diffusion,

$$\mathbf{f} = \begin{pmatrix} 3\text{erf}(10x^{(1)}) \\ 3\text{erf}(10x^{(2)}) \end{pmatrix}, \quad \mathbf{g} = \begin{pmatrix} 0.75 & 0 \\ 0 & 0.75 \end{pmatrix}$$

We set simulation parameters as $(\lambda, \Lambda, \mathcal{R}, h, \beta) = (0.12, 0.14, 1, 0.01, 3)$. The solutions at various times are shown in Figure 4. The adaptive DTQ procedure begins with 287 mesh points and increases the mesh size to 3,720 mesh points at $t = 1.15$. If a tensorized grid spanned the domain sufficiently, it would need to span about $[-5.7, 5.7] \times [-5.7, 5.7]$. Spacing of $\lambda = 0.12$ would require about 9,000 points. However, *a priori* knowledge of the necessary grid domain size is frequently not available and depends on our choice of terminal time $t = 1.15$.

In this example, the percent of Leja points reused from the previous time step averaged about 80% per time step. This reuse is one source of computational savings for the adaptive procedure. The alternative procedure for local Gaussian fitting from section 3.1.1 was used for less than 0.75% of the mesh on average per time step.

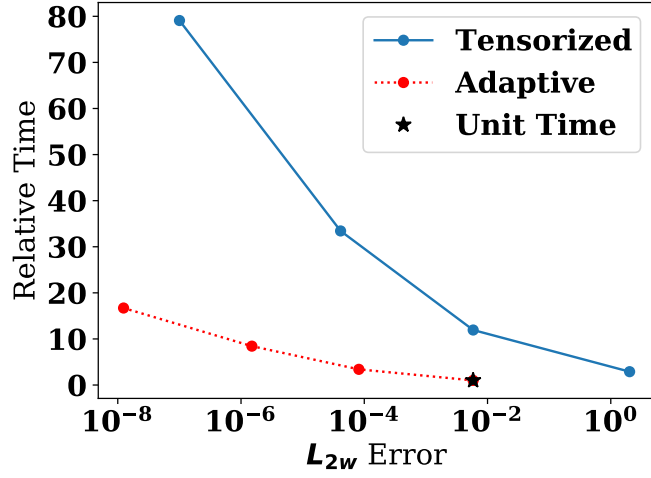


Figure 3: Relative timings for the tensorized vs. adaptive DTQ procedures for the moving hill example of section 5.1. The timings are reported relative to the time taken to run the scenario marked by the star shown in the figure. The cost increases when more accuracy is desired for both methods.

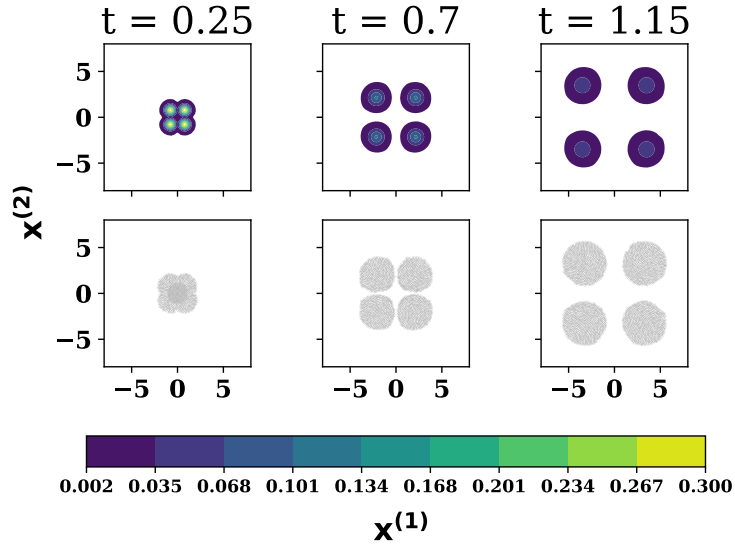


Figure 4: The solution at various times for the four hills example of section 5.2. Top row: Density values are colored only for values greater than 2×10^{-3} for visualization purposes. Bottom row: Adaptive DTQ mesh points at the indicated times.

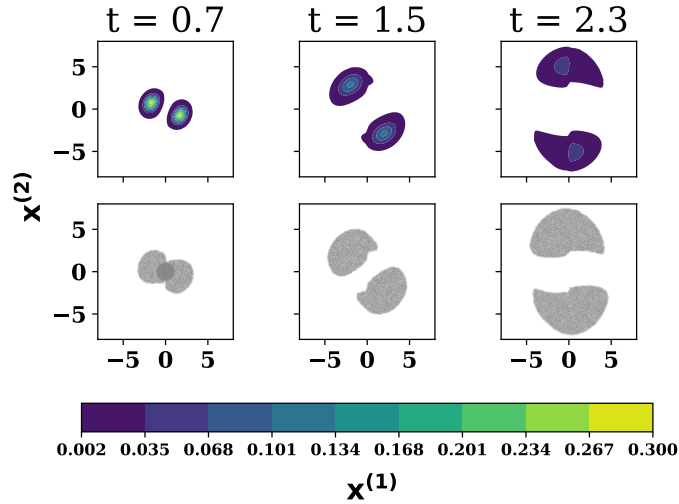


Figure 5: The solution at various times for the two-dimensional spiral example of section 5.3. Top row: Colored density plot, visualizing only values greater than 2×10^{-3} . Bottom row: The adaptive mesh used at the listed times.

5.3 Spiral Example

Consider the solution to the SDE (1) in two dimensions with drift and diffusion,

$$\mathbf{f} = \frac{3}{\|\mathbf{x}\|_2 + 10} \begin{bmatrix} 10\text{erf}(10x^{(1)}) + 5x^{(2)} \\ (-4x^{(1)} + 2x^{(2)}) \end{bmatrix}, \quad \mathbf{g} = \begin{bmatrix} 0.6 & 0 \\ 0 & 0.6 \end{bmatrix},$$

with simulation parameters $(\lambda, \Lambda, \mathcal{R}, h, \beta) = (0.08, 0.1, 1, 0.02, 3)$. The solution to this problem features a single mass splitting into two and rotating in a clockwise spiral. The computational solution at various times is shown in Figure 5. On average, Leja points were reused approximately 85% of the time per time step, and the alternative method for Gaussian fitting was used about 0.9% of the time.

5.4 More Complex Diffusion

Finally, consider the SDE in equation (1) in two dimensions with drift and diffusion

$$\mathbf{f} = \begin{bmatrix} 2\text{erf}(10x^{(1)}) \\ 0 \end{bmatrix}, \quad \mathbf{g} = \begin{bmatrix} 0.01(x^{(1)})^2 + 0.7 & 0.2 \\ 0.2 & 0.01(x^{(2)})^2 + 0.7 \end{bmatrix},$$

with simulation parameter values $(\lambda, \Lambda, \mathcal{R}, h, \beta) = (0.1, 0.12, 1, 0.01, 3)$. The solution at various times is shown in Figure 6. On average, Leja points were reused approximately 85% per time step, and the alternative Gaussian fit method was used for about 0.5% of the mesh points for each time step.

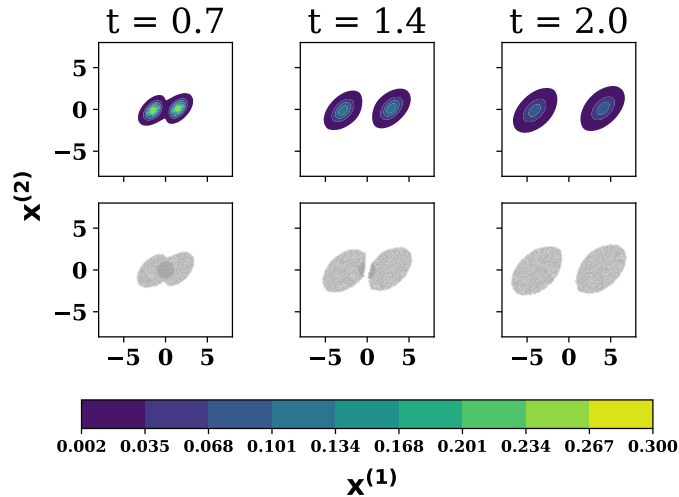


Figure 6: The solution at various times for the example in section 5.4. Top row: Colored density plot, visualizing only values greater than 2×10^{-3} . Bottom row: The adaptive mesh used at the listed times.

6 Conclusion

We have outlined the building blocks for an accurate and adaptive N -dimensional DTQ solver that uses fewer mesh points than a tensorized DTQ procedure and does not require *a priori* knowledge of the domain or mesh size. The adaptive procedure can be about 10 times more efficient than a tensorized approach, even on examples where a tensorized approach is expected to perform well. Two-dimensional examples were shown as a proof of concept for the adaptive procedure. Future work will involve applying the adaptive DTQ procedure to higher dimensional problems.

- [1] P. E. Kloeden and E. Platen. “Applications of Stochastic Differential Equations”. In: *Numerical Solution of Stochastic Differential Equations*. Springer, 1992, pp. 253–275.
- [2] H. Risken. *The Fokker-Planck equation : methods of solution and applications*. eng. 2nd ed. Springer series in synergetics ; v. 18. 1989. ISBN: 354061530X.
- [3] E. Platen and N. Bruti-Liberati. “Monte Carlo Simulation of SDEs”. In: *Numerical Solution of Stochastic Differential Equations with Jumps in Finance*. Springer, 2010, pp. 477–505.
- [4] L. Pichler, A. Masud, and L. Bergman. “Numerical Solution of the Fokker-Planck Equation by Finite Difference and Finite Element Methods-A Comparative Study”. In: vol. 2. Jan. 2013, pp. 69–85. DOI: 10.1007/978-94-007-5134-7_5.
- [5] L. A. Bergman and J. C. Heinrich. “On the reliability of the linear oscillator and systems of coupled oscillators”. In: *International Journal for Numerical Methods in Engineering* 18.9 (1982), pp. 1271–1295. ISSN: 10970207. DOI: 10.1002/nme.1620180902.
- [6] R. S. Langley. “A finite element method for the statistics of non-linear random vibration”. In: *Journal of Sound and Vibration* 101.1 (1985), pp. 41–54. ISSN: 10958568. DOI: 10.1016/S0022-460X(85)80037-7.
- [7] B. F. Spencer and L. A. Bergman. “On the numerical solution of the Fokker-Planck equation for nonlinear stochastic systems”. In: *Nonlinear Dynamics* 4.4 (1993), pp. 357–372.

- [8] A. Masud and R. A. Khurram. “A multiscale/stabilized finite element method for the advection–diffusion equation”. In: *Computer Methods in Applied Mechanics and Engineering* 193.21-22 (2004), pp. 1997–2018.
- [9] M. Kumar, P. Singla, S. Chakravorty, and J. Junkins. “The partition of unity finite element approach to the stationary fokker-planck equation”. In: *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*. 2006, p. 6285.
- [10] S. F. Wojtkiewicz and L. A. Bergman. “Numerical solution of high dimensional Fokker-Planck equations”. In: *8th ASCE Specialty Conference on Probabilistic Mechanics and Structural Reliability, Notre Dame, IN, USA*. Citeseer. 2000.
- [11] S. F. Wojtkiewicz, L. A. Bergman, B. F. Spencer, and E. A. Johnson. “Numerical solution of the four-dimensional nonstationary Fokker-Planck equation”. In: *IUTAM Symposium on Nonlinearity and Stochastic Structural Dynamics*. Springer. 2001, pp. 271–287.
- [12] P. Kumar and S. Narayanan. “Solution of Fokker-Planck equation by finite element and finite difference methods for nonlinear systems”. In: *Sadhana - Academy Proceedings in Engineering Sciences* 31.4 (2006), pp. 445–461. ISSN: 02562499. DOI: 10.1007/BF02716786.
- [13] G. W. Harrison. “Numerical solution of the Fokker Planck equation using moving finite elements”. In: *Numerical Methods for Partial Differential Equations* 4.3 (1988), pp. 219–232. ISSN: 10982426. DOI: 10.1002/num.1690040305.
- [14] S. M. Robinson. “Copyright © by SIAM . Unauthorized reproduction of this article is prohibited .” In: *Society* 18.3 (2007), pp. 1046–1060.
- [15] M. Razi, P. J. Attar, and P. Vedula. “Adaptive numerical solutions of Fokker-Planck equations in computational uncertainty quantification”. In: *Collection of Technical Papers - AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference* April (2011). ISSN: 02734508. DOI: 10.2514/6.2011-1976.
- [16] L. Ferm, P. Lotstedt, and P. Sjoberg. “Adaptive, Conservative Solution Of The Fokker-Planck Equation”. In: (2004), pp. 1–25. URL: papers2://publication/uuid/C832881A-9117-47E7-8D41-72D9E25278C.
- [17] G. M. Subramaniam and P. Vedula. “A transformed path integral approach for solution of the Fokker–Planck equation”. In: *Journal of Computational Physics* 346 (2017), pp. 49–70. ISSN: 10902716. DOI: 10.1016/j.jcp.2017.06.002. URL: <http://dx.doi.org/10.1016/j.jcp.2017.06.002>.
- [18] Y. Xu, H. Zhang, Y. Li, K. Zhou, Q. Liu, and J. Kurths. “Solving Fokker-Planck equation using deep learning”. In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 30.1 (2020), p. 013133.
- [19] H. S. Bhat, R. W. M. A. Madushani, and S. Rawat. “Bayesian inference of stochastic pursuit models from basketball tracking data”. In: *International Conference on Bayesian Statistics in Action*. Springer. 2016, pp. 127–137.
- [20] H. S. Bhat and R. W. M. A. Madushani. “Density Tracking by Quadrature for Stochastic Differential Equations”. 2018.
- [21] M. F. Wehner and W. G. Wolfer. “Numerical evaluation of path-integral solutions to Fokker-Planck equations”. In: *Physical Review A* 27.5 (1983), p. 2663.
- [22] Harish S. Bhat, R. W. M. A. Madushani, and Shagun Rawat. *Parameter inference for stochastic differential equations with density tracking by quadrature*. Vol. 231. Springer International Publishing, 2018, pp. 99–113. ISBN: 9783319760346. DOI: 10.1007/978-3-319-76035-3_7. URL: http://dx.doi.org/10.1007/978-3-319-76035-3%7B%5C_%7D7.
- [23] A. Edrei. “Sur les déterminants récurrents et les singularités d’une fonction donnée par son développement de Taylor”. In: *Compositio Mathematica* 7 (1940), pp. 20–88.

- [24] F. Leja. “Sur certaines suites liées aux ensembles plans et leur application à la représentation conforme”. In: *Annales Polonici Mathematici* 4.1 (1957), pp. 8–13.
- [25] A. Narayan and J. D. Jakeman. “Adaptive Leja sparse grid constructions for stochastic collocation and high-dimensional approximation”. In: *SIAM Journal on Scientific Computing* 36.6 (Apr. 2014), A2952–A2983. DOI: 10.1137/140966368.
- [26] L. Bos, J.-P. Calvi, N. Levenberg, A. Sommariva, and M. Vianello. “Geometric weakly admissible meshes, discrete least squares approximations and approximate Fekete points”. In: *Mathematics of Computation* 80.275 (Jan. 2011), pp. 1623–1638. ISSN: 0025-5718. (Visited on 02/14/2012).
- [27] L. Bos, S. De Marchi, A. Sommariva, and M. Vianello. “Weakly Admissible Meshes and Discrete Extremal Sets”. In: *Numerical Mathematics: Theory, Methods and Applications* 4 (2011), pp. 1–12.
- [28] Y. Xu and A. Narayan. “Randomized weakly admissible meshes”. In: *arXiv:2101.04043 [cs, math, stat]* (Jan. 2021). arXiv: 2101.04043. URL: <http://arxiv.org/abs/2101.04043>.
- [29] L. Bos, S. De Marchi, A. Sommariva, and M. Vianello. “Computing multivariate Fekete and Leja points by numerical linear algebra”. In: *Journal on Numerical Analysis* 48.5 (2010), pp. 441–470.
- [30] J. D. Jakeman. *Adaptive Leja Sequences*. 2019.
- [31] B. Delaunay. “Sur la sphere vide”. In: *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk* 7.793-800 (1934), pp. 1–2.
- [32] H. Edelsbrunner. “Alpha shapes—a survey”. In: *Tessellations in the Sciences* 27 (2010), pp. 1–25.
- [33] R. A. Moore. *AdaptiveDTQ*. <https://github.com/RyleighAMoore/AdaptiveDTQ>. 2021.