# Adaptive Density Tracking by Quadrature for Stochastic Differential Equations

Ryleigh A. Moore*[1] and Akil Narayan[1]

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

June 9, 2022

**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 trapezoidal quadrature rule DTQ procedures by utilizing a flexible quadrature rule that allows for unstructured, adaptive meshes. We describe the procedure for $N$-dimensions, and demonstrate that the resulting adaptive procedure can be significantly more efficient than the trapezoidal DTQ method. We show examples of our procedure for problems ranging from one to five dimensions.

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

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

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 $\boldsymbol{X}_0$.

*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 equation (1) is given by

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

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

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

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

$$\int_{\mathbb{R}^N} p(\mathbf{x},t)d\mathbf{x} = 1.$$

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

## 1.2 Current Methods

Several methods have been developed to numerically approximate the statistics of the SDE's 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 is through Monte Carlo simulation [3], which typically collects a large ensemble of realizations of $\boldsymbol{X}_t$ from equation (1). The large number of samples needed to sufficiently approximate the solution of the SDE makes using this method with sufficient accuracy computationally expensive.

Many numerical methods have been developed to 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-Plank 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 can arise if the drift is large compared to the diffusion. One method to address this challenge utilizes a moving finite element mesh where basis functions, which depend on time instead of only on space, are used to eliminate the spurious oscillations [13]. 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-Plack equation, utilizing a linear multistep method for temporal discretization [16]. Such procedures can be made to adaptively adjust the mesh and time step based on an error tolerance criterion. Deep learning approaches have also 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 [17]. 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 a concern with current methods because the required memory and computational cost increases substantially with the dimension $N$ of the problem. FEMs and finely discretized FDMs have been used to solve four-dimensional problems [10, 11], but more work is needed for higher dimensional problems to become tractable.

In this research, we extend current density tracking by quadrature methods to approximate the PDF of SDEs in high dimensions. DTQ has also been described previously as numerical path integration (NPI) [18, 19, 20] and has been applied to many different disciplines including engineering [19] and finance [21, 22, 23]. A transformed path integral approach is discussed in [24]. Results on the stability, consistency, and convergence of NPI/DTQ under certain conditions are given in [25, 26]. 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 [26], making it very appealing for further study.

### 1.3 Outline and Contributions of this Paper

We work to augment current DTQ algorithms by implementing an accurate and flexible interpolatory quadrature rule, along with adaptive mesh updates, to minimize the computational cost. Our quadrature rule allows for an $N$-dimensional, unstructured mesh that provides flexibility to allocate mesh points to areas of high density and to remove mesh points from areas of low density. The unstructured mesh allows for nontensorial discretizations and partially addresses the curse of dimensionality.

We first summarize the current DTQ method which utilizes a structured mesh and a trapezoidal quadrature rule [26]. Then, we discuss and fully detail our adaptive DTQ method, which uses an interpolatory quadrature rule on Leja points. Finally, we compare the two procedures.

## 2 Density Tracking by Quadrature

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

$$\widetilde{\mathbf{X}}_{n+1} = \widetilde{\mathbf{X}}_n + \mathbf{f}(\widetilde{\mathbf{X}}_n, t)h + \mathbf{g}(\widetilde{\mathbf{X}}_n, t)\sqrt{h}\mathbf{Z}_{n+1}. \tag{3}$$

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

Now, we interpret the time discretized equation (3) as a discrete-time Markov chain. Let $\widetilde{p}(\mathbf{x}, t_n)$ denote the PDF at location $\mathbf{x}$ at time $t_n$ of the Markov chain. From equation (3), we observe that the conditional density of $\widetilde{\mathbf{X}}_{n+1}$ given $\widetilde{\mathbf{X}}_n = \mathbf{y}$ is Gaussian with mean $\widetilde{\boldsymbol{\mu}} = \mathbf{y} + \mathbf{f}(\mathbf{y})h$ and covariance $\widetilde{\boldsymbol{\Sigma}} = h\mathbf{g}(\mathbf{y})\mathbf{g}(\mathbf{y})^T$, notated as

$$P(\widetilde{\mathbf{X}}_{n+1} = \mathbf{x} \mid \widetilde{\mathbf{X}}_n = \mathbf{y}) := G(\mathbf{x}, \mathbf{y}; \widetilde{\mu}, \widetilde{\boldsymbol{\Sigma}}),$$

where

$$G(\mathbf{x}, \mathbf{y}) := G(\mathbf{x}, \mathbf{y}; \widetilde{\boldsymbol{\mu}}, \widetilde{\boldsymbol{\Sigma}})) := \frac{1}{\sqrt{(2\pi)^N |\widetilde{\boldsymbol{\Sigma}}|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \widetilde{\boldsymbol{\mu}})^T \widetilde{\boldsymbol{\Sigma}}^{-1} (\mathbf{x} - \widetilde{\boldsymbol{\mu}})\right). \tag{4}$$

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

The evolution of the density of the Markov chain is described by the associated Chapman-Kolmogorov equation,

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

The next step is to approximate the evolution of $\widetilde{p}(\mathbf{x}, t_n)$ by discretizing equation (5) in space. Let $\{\mathbf{y}_1, \ldots, \mathbf{y}_s\}$ be a set of global mesh points where we will track the density. Then, $\widetilde{p}$ can be approximated by $\widehat{p}$ via a discretization of equation (5),

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

where $\{\omega_i\}_{i=0}^m$ are quadrature weights and $\{\boldsymbol{\eta}_i\}_{i=0}^m$ are quadrature nodes. The initial condition is given as $\widehat{p}(\mathbf{y}_j, 0) = \widetilde{p}(\mathbf{y}_j, 0)$.

One-dimensional DTQ has previously been analyzed and error estimates were established [26, 25]. Convergence of $p$ to $\widetilde{p}$, when using the Euler-Maruyama method [27], was used to help show that, for one-dimensional DTQ, the density $\widehat{p}$ converges in $L_1$ exponentially to the exact density of the Markov chain $\widetilde{p}$, and $\widehat{p}$ converges to the exact density of $p$ with a first-order convergence rate [26]. Furthermore, DTQ has been used for parameter inference problems [28], and a two-dimensional implementation was employed to analyze basketball tracking data from the National Basketball Association [29].

## 2.1 Simple One-Dimensional Interpretation of DTQ

One way to approximate equation (6) is using a trapezoidal quadrature rule [26]. We consider a one-dimensional problem with an equispaced mesh, $\{\widehat{y}_1, \ldots, \widehat{y}_q\} \subset \mathbb{R}$, which has a spatial step size $\kappa$. Then, the density at a point $\widehat{y}_j \in \{\widehat{y}_i\}_{i=1}^q$ is updated using the trapezoidal quadrature rule

$$\widehat{p}(\widehat{y}_j, t_{n+1}) = \kappa \sum_{i=1}^{q} G(\widehat{y}_j, \widehat{y}_i)\widehat{p}(\widehat{y}_i, t_n). \tag{7}$$

Mathematically, equation (7) can also be written as the matrix vector multiply

$$\mathbf{P}_{n+1} = \kappa \mathbf{G}\mathbf{P}_n$$

where $\mathbf{P}_{n+1} = [\widehat{p}(\widehat{y}_1, t_{n+1}), \ldots, \widehat{p}(\widehat{y}_q, t_{n+1})]^T$ and $\mathbf{G}_{i,v} = G(\widehat{y}_i, \widehat{y}_v)$.

## 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 the tensorization of a univariate grid,

$$\{\widehat{\mathbf{y}}_i\}_{i=1}^s = \bigotimes_{\ell=1}^{N} \{\widehat{y}_1, \ldots, \widehat{y}_q\}, \qquad \{\widehat{y}_1, \ldots, \widehat{y}_q\} \subset \mathbb{R}.$$

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

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

In vector form, the above is

$$\mathbf{P}_{n+1} = \kappa^N \mathbf{G}\mathbf{P}_n, \qquad \mathbf{G}_{i,v} = G(\widehat{\mathbf{y}}_i, \widehat{\mathbf{y}}_v),$$

where $\mathbf{P}_{n+1} \coloneqq [\widehat{p}(\widehat{\mathbf{y}}_1, t_{n+1}), \ldots, \widehat{p}(\widehat{\mathbf{y}}_s, t_{n+1})]^T$. The matrix $\kappa^N \mathbf{G}$ contains values describing the movement of density; however, it is not a Markov transition matrix in general [26].

The numerical solution $\widehat{p}$ can, in principle, be directly computed using this procedure; however, this can become expensive quickly as the dimension increases. For higher dimensional problems, we require $q^N$ mesh points for the tensorization strategy. For example, if we need 100 points per dimension, in four dimensions we will need $10^8$ points, which is computationally prohibitive. In order to help extend DTQ to higher dimensions, we provide an a different strategy to discretize the integral in equation (5), which allows for an unstructured set of mesh points.

# 3 DTQ on an Unstructured Mesh

We will now describe our procedure for implementing DTQ on an unstructured mesh in $N$ dimensions. We utilize an unstructured, adaptive mesh and an interpolatory quadrature rule to approximate the integral in equation (5) by treating a portion of the integrand as a Gaussian density. For each point in the global unstructured mesh, $\mathbf{y}_j \in \{\mathbf{y}_1, \ldots, \mathbf{y}_s\}$, we compute quadrature nodes and weights for an interpolatory quadrature rule. We denote the quadrature nodes as $\{\boldsymbol{\eta}_1, \ldots, \boldsymbol{\eta}_m\}$, where we suppress the $j$ dependence since the procedure updates one mesh point at a time (eg. $\{\boldsymbol{\eta}_1, \ldots, \boldsymbol{\eta}_m\} = \{\boldsymbol{\eta}_1, \ldots, \boldsymbol{\eta}_m\}_j$).

Now we will update the density associated to a member of the global mesh $\mathbf{y}_j$,

$$\widetilde{p}(\mathbf{y}_j, t_n) = \int_{\mathbb{R}^N} G(\mathbf{y}_j, \mathbf{y}) \widetilde{p}(\mathbf{y}, t_n) d\mathbf{y} \tag{8}$$

$$= \int_{\mathbb{R}^N} r(\mathbf{y}) \mathcal{N}(\mathbf{y}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) d\mathbf{y} \tag{9}$$

so that

$$\widehat{p}(\mathbf{y}_j, t_n) = \sum_{i=1}^{m} r(\boldsymbol{\eta}_i) \widehat{w}_i \tag{10}$$

where

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

Section 3.1 describes how we effect the integral in equation (9) by using a Laplace approximation of the integrand in equation (8) to identify $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ which subsequently allows us to define the weight function $\mathcal{N}$ (through $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$) and the new integrand $r$. These values differ for each $\mathbf{y}_j$ (i.e., $\boldsymbol{\mu} = \boldsymbol{\mu_j}, \boldsymbol{\Sigma} = \boldsymbol{\Sigma_j}$, $r = r_j$, but we suppress this $j$ dependence). Then, section 3.2 details the identification of the quadrature weights in equation (10), and section 3.3 details the selection of the quadrature nodes.

## 3.1 Laplace Approximation via Least Squares

In this section, we describe how $r$ and $\mathcal{N}$ in equation (9) are determined. We identify $\mathcal{N}$ as a Laplace approximation to the integrand of equation (8), which we implement practically by performing a local least-squares quadratic fit to the log-integrand using nearby mesh points.

The Laplace approximation is computed for each point in the mesh. We consider a specific global mesh point $\mathbf{y}_j$ for this discussion. Let $\mathfrak{N}$ be the set of points used for the Laplace approximation. We note that $\mathfrak{N}$ depends on the point $\mathbf{y}_j$ (i.e. $\mathfrak{N} = \mathfrak{N}_j$) but we suppress the $j$ dependence. When available, the quadrature nodes $\{\boldsymbol{\eta}_1, \ldots, \boldsymbol{\eta}_m\}$, which were used at the previous time step to update the density at $\mathbf{y}_j$, are used. Otherwise, we use a set of nearest neighbor points to $\mathbf{y}_j$ (including itself). When needed, the nearest neighbors are determined using the Euclidean distance. The size of the set $\mathfrak{N}$ is formalized in section 5. In the beginning stages of the procedure, quadrature nodes are not known, so we use $\mathbf{y}_j$'s nearest neighbors. After quadrature nodes are known, we typically use them in place of the nearest neighbors. In this section, we will assume the use of $\{\boldsymbol{\eta}_1, \ldots, \boldsymbol{\eta}_m\}$ for notational simplicity; however, the procedure is equivalent if nearest neighbors are used instead.

Now, we will compute the Laplace approximation. Let the $i^{th}$ component of the vector $\boldsymbol{\psi}$ be given as

$$\boldsymbol{\psi}_i := -\log(G(\mathbf{y}_j, \boldsymbol{\eta}_i)\widehat{p}(\boldsymbol{\eta}_i, t_n)), \qquad i = 1, \ldots, m$$

so that $\boldsymbol{\psi}$ is an $m$-dimensional vector. The Laplace approximation will model this log-integrand as a quadratic polynomial,

$$\boldsymbol{\psi}_i \approx \widetilde{\psi}(\boldsymbol{\eta}_i) := c + \mathbf{d}^T\boldsymbol{\eta}_i + (\boldsymbol{\eta}_i)^T\mathbf{A}\boldsymbol{\eta}_i, \tag{11}$$

for a scalar $c$, vector $\mathbf{d} \in \mathbb{R}^N$, 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. Let $\alpha \in \mathbb{N}_0^N$ be a multi-index with the standard convention,

$$\alpha = (\alpha_1, \ldots, \alpha_N), \qquad |\alpha| := \sum_{\ell=1}^{N} \alpha_\ell, \qquad \boldsymbol{\eta}^\alpha = \prod_{\ell=1}^{N} \left(\eta^{(\ell)}\right)^{\alpha_\ell},$$

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

$$\mathcal{S}_k := \operatorname{span}\left\{\boldsymbol{\eta}^\alpha \;\middle|\; \alpha \in \Upsilon_k\right\}, \qquad \dim \mathcal{S}_k = |\Upsilon_k|,$$

where we take $\Upsilon_k$ to be the set of multi-indices corresponding to a degree-$k$ approximation,

$$\Upsilon_k := \left\{\alpha \in \mathbb{N}_0^N \;\middle|\; |\alpha| \le k\right\}, \qquad |\Upsilon_k| = \binom{N+k}{N}.$$

We will perform a quadratic fit with $k = 2$. We use $\alpha^{(1)}, \ldots, \alpha^{(|\Upsilon_2|)}$, an enumeration of the elements of $\Upsilon_2$, to form a Vandermonde matrix, $\mathbf{M} \in \mathbb{R}^{m\times|\Upsilon_2|}$ defined as,

$$\mathbf{M}_{i,v} = \boldsymbol{\eta}_i^{\alpha^{(v)}}$$

Now, a least-squares fit to the data $\boldsymbol{\psi}$ is the emulator,

$$\widetilde{\psi}(\boldsymbol{\eta}) = \sum_{v=1}^{|\Upsilon_2|} \widehat{\tau}_v \boldsymbol{\eta}^{\alpha^{(v)}}, \qquad \widehat{\boldsymbol{\tau}} = (\widehat{\tau}_1, \ldots, \widehat{\tau}_{|\Upsilon_2|})^T,$$

where $\widehat{\boldsymbol{\tau}}$ is given as the least-squares solution to the linear system,

$$\mathbf{M}\widehat{\boldsymbol{\tau}} = \boldsymbol{\psi}.$$

Once the coefficients $\widehat{\boldsymbol{\tau}}$ are computed, we translate $\widetilde{\psi}$ into the symmetric quadratic form in equation (11) using the following identification of the entries of $c$, $\mathbf{d}$ and $\mathbf{A}$,

$$c = \widehat{\boldsymbol{\tau}}_{\mathcal{I}(0)}, \qquad \mathbf{d}_\ell = \widehat{\boldsymbol{\tau}}_{\mathcal{I}(\mathbf{e}_\ell)}, \qquad \mathbf{A}_{\ell,u} = \frac{1}{2-\delta_{\ell,u}}\widehat{\boldsymbol{\tau}}_{\mathcal{I}(\mathbf{e}_\ell+\mathbf{e}_u)}$$

where $\delta_{\ell,u}$ is the Kronecker delta, $\mathbf{e}_\ell \in \mathbb{N}_0^N$ is the cardinal unit vector in direction $\ell$ with entry 1 in location $\ell$ and zeros elsewhere, and $\mathcal{I}(\alpha)$ is a function that returns the linear index in $\Upsilon_2$ associated to $\alpha$,

$$v = \mathcal{I}(\alpha) \quad \Longrightarrow \quad \alpha = \alpha^{(v)}.$$

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 density $\mathcal{N}$ from this quadratic fit to the log-integrand:

**Proposition 1.** *If* $\mathbf{A}$ *in equation* (11) *is positive-definite, then*

$$\exp(-\widetilde{\psi}(\boldsymbol{\eta})) = C \exp\left(-(\boldsymbol{\eta}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{\eta}-\boldsymbol{\mu})\right), \tag{12}$$

*where*

$$\boldsymbol{\mu} = -\frac{1}{2}\mathbf{U}\boldsymbol{\Lambda}^{-1}\mathbf{d} \qquad \boldsymbol{\Sigma}^{-1} = \mathbf{A} \qquad C = \exp(-c + \frac{1}{4}\mathbf{d}^T\boldsymbol{\Lambda}^{-1}\mathbf{d})) \tag{13}$$

*Proof.* Since $\boldsymbol{A}$ is symmetric (by construction) and positive-definite, it has an orthogonal diagonalization

$$\mathbf{A} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T, \qquad \mathbf{U}\mathbf{U}^T = \mathbf{I}, \qquad \boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1, \ldots, \lambda_N)$$

with positive eigenvalues $\lambda_\ell > 0$ for all $\ell$. Define $\boldsymbol{\gamma} \coloneqq \mathbf{U}^T\boldsymbol{\eta}$, then

$$\widetilde{\psi}(\boldsymbol{\eta}) = c + \mathbf{d}^T\boldsymbol{\gamma} + (\boldsymbol{\gamma})^T\boldsymbol{\Lambda}\boldsymbol{\gamma}.$$

Let $d^{(\ell)}$ and $\gamma^{(\ell)}$ be the components of $\mathbf{d}$ and $\boldsymbol{\gamma}$, then a rearrangement yields

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

and we achieve equation (12), with $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$, and $C$ as given in equation (13). □

Using the Laplace approximation, we specify the weight function $\mathcal{N}$ using $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ from equation (13) so that

$$\widetilde{p}(\mathbf{y}_j, t_{n+1}) = \int_{\mathbb{R}^N} r(\mathbf{y})\mathcal{N}(\mathbf{y}; \boldsymbol{\mu}, \boldsymbol{\Sigma})d\mathbf{y}, \qquad r(\mathbf{y}) = \frac{G(\mathbf{y}_j, \mathbf{y})\widetilde{p}(\mathbf{y}, t_n)}{\mathcal{N}(\mathbf{y}; \boldsymbol{\mu}, \boldsymbol{\Sigma})}. \tag{14}$$

We can approximate this integral using a quadrature rule like in equation (10) with quadrature nodes $\{\boldsymbol{\eta}_1, \ldots, \boldsymbol{\eta}_m\}$.

### 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 occasionally not occur in practice. When $\mathbf{A}$ is not positive-definite, we use the alternative method shown in equation (15). Additionally, we use the alternative method in the case that the quadrature nodes selected via the procedure described in section 3.3 make the quadrature rule described in section 3.2 ill-conditioned. In practice, when the alternative method is used, it is typically to update the density of mesh points at or near the mesh boundary.

For the alternative method, we use a temporary set of quadrature nodes denoted as $\{\boldsymbol{\eta}_1^*, \ldots, \boldsymbol{\eta}_m^*\}$. Note, the quadrature nodes for the alternative procedure are such that $\{\boldsymbol{\eta}_1^*, \ldots, \boldsymbol{\eta}_m^*\} \not\subseteq \{\mathbf{y}_1, \ldots, \mathbf{y}_s\}$. This is distinctly different than the quadrature nodes $\{\boldsymbol{\eta}_1, \ldots, \boldsymbol{\eta}_m\}$ used in the standard procedure which are members of the global mesh. In the case of the alternative procedure only, the corresponding PDF values, $\widehat{p}(\boldsymbol{\eta}^*, t_{n+1})$, are determined via interpolation and/or extrapolation to

avoid ill-conditioned quadrature. In the case of extrapolation, we assign the density value to be $\min_i \widehat{p}(\mathbf{y}_i, t_n)$. Use of the alternative method varies, but it usually is only used for around 0-2% of mesh points per time step on average in two-dimensional problems.

For the alternative procedure, we take advantage of the structure of $G(\mathbf{x}, \mathbf{y})$ to procure a weight function so that

$$\begin{aligned} \widetilde{p}(\mathbf{y}_j, t_{n+1}) &= \int_{\mathbb{R}^N} r(\mathbf{y}) \mathcal{N}(\mathbf{y}; \mathbf{y}_j + h\mathbf{f}(\mathbf{y}_j), h\mathbf{g}(\mathbf{y}_j) g(\mathbf{y}_j)^T) d\mathbf{y} \\ r(\mathbf{y}) &= \frac{G(\mathbf{y}_j, \mathbf{y}) \widetilde{p}(\mathbf{y}, t_n)}{\mathcal{N}(\mathbf{y}; \mathbf{y}_j + h\mathbf{f}(\mathbf{y}_j), h\mathbf{g}(\mathbf{y}_j)\mathbf{g}(\mathbf{y}_j)^T)}. \end{aligned} \tag{15}$$

For the alternative method, the weight function $\mathcal{N}$ has a mean and variance that depends only on the current point we are updating, $\mathbf{y}_j$. We can approximate this integral using a quadrature rule like in equation (10) with quadrature nodes $\{\boldsymbol{\eta}_1^*, \ldots, \boldsymbol{\eta}_m^*\}$.

## 3.2 Quadrature Weights

We now detail how the quadrature rule in equation (10) of

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

is generated, assuming the quadrature nodes $\{\boldsymbol{\eta}_i\}_{i=1}^m$ are known. Section 3.3 later describes the selection process of $\{\boldsymbol{\eta}_i\}_{i=1}^m$ from the global mesh.

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 $\widetilde{p}$ at $\mathbf{y}_j$ as in equation (9). Now, let $\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^T$ be any decomposition of $\boldsymbol{\Sigma}$ (e.g., the Cholesky decomposition). Then, integral (9) can be rewritten as

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

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

$$\boldsymbol{\eta}_i = \mathbf{L}\boldsymbol{\zeta}_i + \boldsymbol{\mu} \tag{16}$$

where $\{\boldsymbol{\zeta}_i\}_{i=1}^m$ are nodes in $\boldsymbol{\zeta}$ space. The weights $\widehat{w}_i$ of the quadrature rule in equation (10) 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., given a degree-$k$ polynomial family $\mathcal{P}_k$, we construct a unique polynomial $Q \in \mathcal{P}_k$, so that

$$r\left(\boldsymbol{L}\boldsymbol{\zeta}_i + \boldsymbol{\mu}\right) = Q(\boldsymbol{\zeta}_i), \qquad\qquad i = 1, \ldots, m.$$

More precisely, set $m = m_k$ and let $\{\phi_i\}_{i=1}^{m_k}$ be a basis for the degree-$k$ polynomial space $\mathcal{P}_k$, so that

$$Q(\boldsymbol{\zeta}) = \sum_{i=1}^{m_k} \widehat{c}_i \phi_i(\boldsymbol{\zeta}),$$

where $\widehat{\boldsymbol{c}} = (\widehat{c}_1, \ldots, \widehat{c}_{m_k})^T$ solves the linear system,

$$\mathbf{V}\widehat{\boldsymbol{c}} = \mathbf{r}, \qquad \mathbf{V}_{i,v} = \phi_v(\zeta_i), \qquad \mathbf{V} \in \mathbb{R}^{m_k \times m_k}$$

where $\boldsymbol{r} = (r(\boldsymbol{\eta}_1), \ldots, r(\boldsymbol{\eta}_{m_k}))^T$. We generate the quadrature weights from exact integration of $Q$ in place of $r$:

$$\int r(\boldsymbol{L}\boldsymbol{\zeta} + \boldsymbol{\mu}) \mathcal{N}\left(\boldsymbol{\zeta}; \mathbf{0}, \boldsymbol{I}\right) d\boldsymbol{\zeta} \approx \int Q(\boldsymbol{\zeta}) \mathcal{N}\left(\boldsymbol{\zeta}; \mathbf{0}, \boldsymbol{I}\right) d\boldsymbol{\zeta} = \sum_{i=1}^{m_k} r\left(\boldsymbol{\eta}_i\right) \widehat{w}_i, \tag{17}$$

where $\boldsymbol{\eta}_i$ are the quadrature nodes and the quadrature weights $\widehat{w}_i$ are given by

$$\widehat{\boldsymbol{w}} = (\widehat{w}_1, \ldots, \widehat{w}_{m_k}) = \boldsymbol{\xi}^T \mathbf{V}^{-1}, \qquad \boldsymbol{\xi}_i \coloneqq \int \phi_i(\boldsymbol{\zeta}) \mathcal{N}(\boldsymbol{\zeta}; \mathbf{0}, \boldsymbol{I}) d\boldsymbol{\zeta}. \tag{18}$$

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

$$\phi_i(\boldsymbol{\zeta}) = \prod_{\ell=1}^{N} \widehat{h}_{\alpha_\ell^{(i)}}(\zeta_\ell), \tag{19}$$

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

$$\int_{\mathbb{R}} \widehat{h}_i(\zeta) \widehat{h}_v(\zeta) \mathcal{N}(\zeta; 0, 1) \, d\zeta = \delta_{i,v}, \qquad \deg \widehat{h}_i = i,$$

and $\delta_{i,\ell}$ is the Kronecker delta. The uniqueness of each $\widehat{h}_i$ 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 defined from equation (19) for $\mathcal{P}_k$ then satisfies the following multivariate orthogonality condition,

$$\int_{\mathbb{R}^N} \phi_i(\boldsymbol{\zeta}) \phi_v(\boldsymbol{\zeta}) \mathcal{N}(\boldsymbol{\zeta}; \mathbf{0}, \boldsymbol{I}) \, d\boldsymbol{\zeta} = \delta_{i,v}, \qquad \phi_1(\boldsymbol{\zeta}) \equiv 1, \tag{20}$$

so that the moments $\boldsymbol{\xi}$ in equation (18) are given by $\boldsymbol{\xi}_i = \delta_{i,1}$. Therefore, with this basis, equation (18) implies that the quadrature weights are simply given as the first row of $\mathbf{V}^{-1}$,

$$\widehat{\boldsymbol{w}}^T = \left(\mathbf{V}^{-1}\right)_{1,:}. \tag{21}$$

In terms of the quadrature nodes $\{\boldsymbol{\eta}_i\}_{i=1}^{m_k}$, we connect $\omega_i$ from equation (6) to $\widehat{w}_i$ as

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

### 3.3 (Weighted) Leja Sequences

We now describe the selection of quadrature nodes $\{\boldsymbol{\eta}_i\}_{i=1}^{m_k}$. Fix a global mesh index $j$, and let $\mathfrak{Z}$ denote the set of candidate Leja points determined from the $(\mathbf{L}, \boldsymbol{\mu})$-affine map found via the Laplace approximation in section 3.1 where $\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^T$. More specifically, the global mesh is transformed so that

$$\boldsymbol{\zeta}_i \coloneqq \mathbf{L}^{-1}(\mathbf{y}_i - \boldsymbol{\mu}), \qquad \boldsymbol{\zeta}_i \in \mathfrak{Z}.$$

Note that in practice, we take $\mathfrak{Z}$ as a a subset of the transformed global mesh for computational efficiency.

Our goal is to identify a subset of nodes $\{\boldsymbol{\zeta}_i\}_{i=1}^{m_k}$ , from $\mathfrak{Z}$, which define $\{\boldsymbol{\eta}_i\}_{i=1}^{m_k}$ through equation (16). Recall, $\{\boldsymbol{\eta}_i\}_{i=1}^{m_k}$ are used as quadrature nodes in the approximation given by equation (10). The nodes $\{\boldsymbol{\zeta}_i\}_{i=1}^{m_k}$ are computed as a discrete weighted Leja sequence from $\mathfrak{Z}$.

We will formally define one-dimensional Leja sequences, first on a compact interval $[a, b]$, and then weighted on $\mathbb{R}$. Afterwards, we will detail a linear algebra procedure to define $N$-dimensional Leja sequences.

On an interval $[a, b]$, an unweighted Leja sequence is classically defined as any sequence of points $z_\ell \in [a, b] \subset \mathbb{R}$ for $\ell = 1, 2, \ldots$ that solves the sequential optimization problem,

$$z_{J+1} = \underset{z \in [a,b]}{\operatorname{argmax}} \prod_{\ell=0}^{J} |z - z_\ell|, \tag{22}$$

where $z_0$ is arbitrarily chosen in the interval $[a, b]$ [30, 31]. Leja sequences are not unique due to the choice of the initial point $z_0$ as well as the potential for multiple maximizers at each step of equation (22).

In the one-dimensional setting, we are interested in identifying nodes whose corresponding interpolatory quadrature rule is an accurate approximation of the form in equation (17). To accomplish this, we must consider integrals over the entire real line with respect to a weight function. For now, we will use a general weight function $W$ which we will later specify as $\mathcal{N}$. A naive extension of the optimization in equation (22) that replaces $[a, b]$ by $\mathbb{R}$ is not well defined, so we adopt the strategy from [32] that uses weighted Leja sequences and shows that these sequences result in accurate interpolatory quadrature rules with respect to a weight function.

Let $z_\ell$ for $\ell = 1, 2, \ldots$ be any sequence of solutions to a modified version of equation (22),

$$z_{J+1} = \underset{z \in \mathbb{R}}{\operatorname{argmax}} \sqrt{W(z)} \prod_{\ell=0}^{J} |z - z_\ell|, \tag{23}$$

where $z_0$ 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 motivated by the fact that weighted Leja sequences defined by equation (23) satisfy the asymptotic Fekete property, and asymptotically distribute like $W$-weighted Gauss quadrature nodes [32]. Empirically, these sequences also form stable quadrature rules for approximating $W$-weighted integrals.

We cannot directly use equation (23) in the DTQ framework because we do not have the freedom to choose points arbitrarily in $\mathbb{R}$. Instead, we pose an optimization problem over the discrete candidate set $\mathfrak{Z}$. We therefore construct the following discrete, $\mathcal{N}$-weighted Leja sequence

$$\zeta_{J+1} = \underset{\zeta \in \mathfrak{Z}}{\operatorname{argmax}} \sqrt{\mathcal{N}(\zeta; 0, 1)} \prod_{\ell=0}^{J} |\zeta - \zeta_\ell|. \tag{24}$$

Ideally, the candidate set $\mathfrak{Z}$ should form a so-called weakly admissible mesh so that the points sufficiently cover the domain of interest [33, 34, 35].

The above discussion holds for one dimension, but the objective function being maximized in equation (24) does not directly generalize to higher dimensions. We will now discuss the one-dimensional problem in a form that can be extended to the multivariate case. The calculation of weighted discrete Leja sequences in equation (24) can be simplified. It is possible to show that equation (24) is equivalent to constructing a Vandermonde-like matrix via a particular kind of greedy determinant maximization [36] which reduces the process into a simple numerical linear algebra problem. The sequence in equation (24) can be computed from the pivots of a row-pivoted LU factorization of a Vandermonde-like matrix. In particular, let $\widetilde{\mathbf{V}} \in \mathbb{R}^{|\mathfrak{Z}| \times m_k}$ denote a weighted Vandermonde-like matrix on the candidate points $\mathfrak{Z}$,

$$\widetilde{\mathbf{V}}_{\ell,i} = \sqrt{\mathcal{N}(\zeta_\ell; 0, 1)}\, \widehat{h}_i(\zeta_\ell), \qquad \zeta_\ell \in \mathfrak{Z},$$

where we choose $\{\widehat{h}_i\}_{i=1}^{m_k}$ as the univariate, $\mathcal{N}(\cdot; 0, 1)$-weighted orthonormal Hermite polynomials. With $\widetilde{\mathbf{P}}\widetilde{\mathbf{V}} = \widetilde{\mathbf{L}}\widetilde{\mathbf{U}}$ the pivoted LU decomposition of $\widetilde{\mathbf{V}}$, a solution to equation (24) is given by the first $m_k$ $\widetilde{\mathbf{P}}$-permuted points

$$\{\zeta_{\ell_1}, \ldots, \zeta_{\ell_{m_k}}\} \in \mathfrak{Z}, \qquad (\ell_1, \ldots, \ell_{|\mathfrak{Z}|})^T = \widetilde{\mathbf{P}}\, (1, 2, \ldots, |\mathfrak{Z}|)^T. \tag{25}$$

Due to the equivalence between the solution to equation (24) and the linear algebra procedure, in practice we compute weighted Leja sequences via an LU factorization with partial row pivoting of a Vandermonde-like matrix [36].

This linear algebra procedure is directly generalizable to multiple dimensions. For $N > 1$, we use $N$-dimensional points $\boldsymbol{\zeta}_i$ to define

$$\widetilde{\mathbf{V}}_{\ell,i} = \sqrt{\mathcal{N}(\boldsymbol{\zeta}_\ell; \mathbf{0}, \boldsymbol{I})}\ \phi_i(\boldsymbol{\zeta}_\ell), \qquad \boldsymbol{\zeta}_\ell \in \mathfrak{Z},$$

where we use the multivariate, $\mathcal{N}(\cdot; \mathbf{0}, \boldsymbol{I})$-weighted orthonormal Hermite polynomial basis $\{\phi_i\}_{i=1}^{m_k}$. Then, we use the same greedy determinant maximization procedure (implemented via the pivoted LU approach) used in the one-dimensional case to compute the $N$-dimensional Leja sequence.

We emphasize that the points selected from the pivots of the LU factorization make up a weighted discrete Leja sequence and are used as quadrature nodes in the approximation (17). For the overall DTQ method, 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 code from the PyApprox package [37].

### 3.4 Quadrature Rule Condition Number: Leja Point Reuse and Alternative Method Use

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

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

where $\|\cdot\|_1$ is the $\ell^1$ norm on vectors, $\widehat{\boldsymbol{w}}$ are the interpolatory quadrature weights defined in equation (21), and $\epsilon$ is an adjustable threshold. In practice, the number of mesh points for 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 a large portion of the Leja points from the previous time step, which increases the speed of the algorithm substantially. We will quantify Leja point reuse in section 5 for several examples.

We also use the condition number to check if we should revert to the alternative procedure. More specifically, if our computation results in a quadrature rule with condition number such that $\Gamma > cond_{alt}$, we assume that the quadrature rule will not yield an accurate result, and revert to using the alternative procedure.

## 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.
The adaptive procedures attempts to reduce the number of mesh points required to compute $\widehat{p}(\mathbf{x}, t)$ by adaptively updating the mesh as the solution evolves in time. To do this, 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

Our adaptive procedure adds points to extend the boundary of the mesh. In one-dimensional problems, identifying the boundary points simply involves finding the maximum and minimum mesh coordinate. In order to identify the boundary of the mesh in higher dimensions, we utilize a procedure that combines a mesh triangulation and *alpha shape* procedure. First, we construct a

**Algorithm 1** Adaptive DTQ

**procedure** AdaptiveDTQ:
  **while** step forward **do**:
    add points to mesh boundary if needed (section 4.2)
    remove points from mesh boundary if needed (section 4.3)
    **for** each mesh point **do**:
      attempt local Laplace approximation (section 3.1)
      **if** the Laplace approximation is successful **then**:
        **if** attempting to reuse Leja Points **then**:
          compute quadrature weights (section 3.2)
          step forward using equation (10) with $r$ from (14) evaluated at Leja points
          **if** condition number is sufficient **then**
            save updated density value
            record if we should try to reuse Leja points at next time step
            **continue**
      attempt Leja point computation (section 3.3)
      **if** computed Leja points successfully **then**
        compute quadrature weights (section 3.2)
        step forward using equation (10) with $r$ from (14) evaluated at Leja points
        **if** condition number is sufficient **then**
          save updated density value
          record if we should try to reuse Leja points at next time step
          **continue**
      use alternative method (section 3.1.1)
      save updated density value

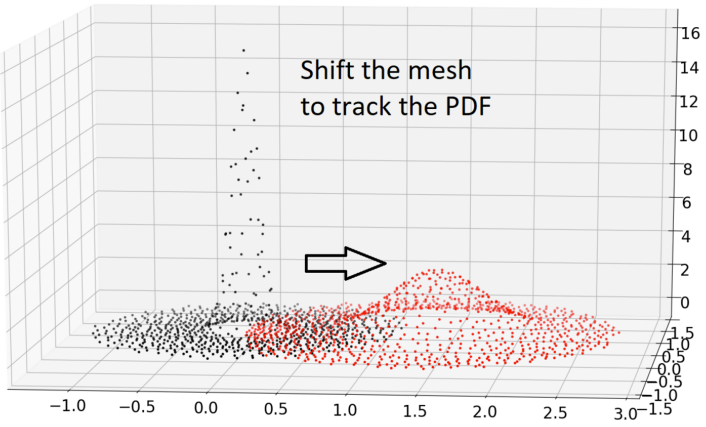


Figure 1: Shifting the mesh to keep track of the density is achieved through adding and removing mesh points.

Delaunay triangulation of the global mesh $\{\mathbf{y}_i\}_{i=1}^s$, which satisfies the condition that no mesh point lies in the interior of any circumscribing sphere of any simplex in the triangulation. The Delaunay triangulation ensures that the minimum angle is maximized for all the triangles in the triangulation to avoid thin, sliver triangles [38].

The alpha shape algorithm uses the simplices from the Delaunay triangulation to recover the boundary points of a point mesh. A pseudocode explanation of this procedure is given in Algorithm 2 and illustrated in Figure 2. This algorithm works for $N > 1$ dimensions. Figure 3 shows a completed

identification of boundary points. A survey of alpha shapes is available for more information [39]. For the procedure, we select $\widehat{\alpha} = (3/2)\Delta max$, where $\Delta max$ is the enforced maximum distance between points in the mesh.

**Algorithm 2** Alpha Shape for Finding Boundary Points

**procedure** AlphaShape($\widehat{\alpha}$, triangulation):
    compute the radius of the circumsphere for each simplex in the triangulation
    simplicesList = all simplices with circumsphere radius $< \widehat{\alpha}$
    initialize edgesList = []
    **for** simplex in simplicesList **do**:
        add simplex edges to edgesList
    boundaryEdges = edges in edgesList which appear exactly once
    boundaryVertices = all vertices associated with edges in boundaryEdges
    **return** boundaryVertices

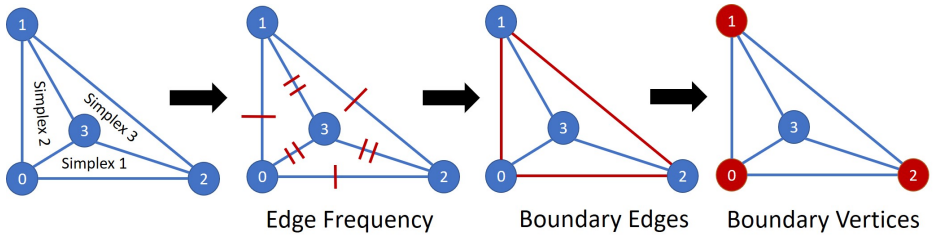


Figure 2: **Alpha shape algorithm depiction.** This outlines the alpha shape algorithm. First, identify all edges from the simplices (including duplicates). For example, the edge from point 0 to 3 has frequency two because it appears in both simplex 1 and 2. Then, all edges that appear only once are boundary edges. Finally, the vertices corresponding to the boundary edges are the boundary vertices, in this case, vertices 0, 1, and 2.

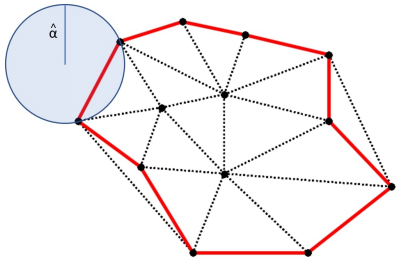


Figure 3: A depiction of a completed two-dimensional alpha shape procedure. The Delaunay triangulation is shown, and the red edges indicate the boundary that is identified.

## 4.2 Adding Boundary Points

After the mesh boundary is identified, as described in the previous section, we add mesh points based on a prescribed tolerance parameter, $\beta$, which is tuned to maintain a sufficiently small value of $\widehat{p}$ on the mesh boundary. For each boundary point where $\widehat{p}$ is larger than $10^{-\beta}$, candidate points are generated as equispaced points on an $N$-dimensional grid with spacing $\Delta max$. 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 $\Delta min$ and it must be less than a set value $\Delta max$. Essentially, $\Delta max$ and $\Delta min$ are the enforced minimum and maximum distances, respectively, between mesh points. New mesh coordinates added at time $t_{n+1}$ are assigned a density value of $\min_i \widehat{p}(\mathbf{y}_i, t_n)$. During the simulation, we begin adding points at time step number $step_{AC} \in \mathbb{Z}^+$, and from time step $step_{AC}$ we run the procedure for adding points every $step_A \in \mathbb{Z}^+$ time steps. For example, if $step_{AC} = 1$ and $step_A = 3$ we would start adding points every three time steps starting at the first time step. We typically take $step_A = 1$ so that we check to add points every time step after $step_{AC}$.

### 4.3 Removing Points

We remove the mesh points that are deemed unnecessary in terms of the value of the density $\widehat{p}$. Points are removed based on the prescribed tolerance, $\beta$ from section 4.2, which is used to maintain a sufficiently small value of $\widehat{p}$ at the boundary. More specifically, all mesh points (anywhere in the mesh, not just at the boundary) 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.

We first run this removal procedure on time step $step_{RC} \in \mathbb{Z}^+$, and subsequently run the procedure for removing points every $step_R \in \mathbb{Z}^+$ time steps. We typically take $step_{RC}, step_R \approx 10$ so that we do not start removing points right away nor at every time step.

## 5 Results

The code for the adaptive DTQ procedure is located on GitHub [40]. In this section, we demonstrate Algorithm 1 through several examples. Some examples use spatially-dependent drift and diffusion functions. Problems from one to five dimensions are included. Table 1 outlines parameters used for the Adaptive DTQ along with a brief description.

Table 1: **DTQ parameter symbols and descriptions**.

| Parameter | Description |
|---|---|
| $\epsilon$ | $1 + \epsilon$ is the condition number threshold for Leja point reuse |
| $cond_{alt}$ | The condition number threshold for using the Alternative method |
| $LP_Q$ | Number of Leja points used for the quadrature rule |
| size of $\mathfrak{N}$ | The size of the set used for the Laplace approximation |
| size of $\mathfrak{Z}$ | The number of candidate Leja points |
| $step_{AC}$ | The first step number where adding points is considered |
| $step_{RC}$ | The first step number where removing points is considered |
| $step_A$ | Determines how often points are added |
| $step_R$ | Determines how often points are removed |
| $\Delta min$ | Minimum distance allowed between points |
| $\Delta max$ | Maximum distance allowed between points |
| $\mathcal{R}$ | Used to define the radius of points in the initial mesh |
| $h$ | The temporal time step size |
| $\beta$ | Used to define values for adding and removing points |

### 5.1 Commonalities Among Examples

We demonstrate our adaptive DTQ method in this section through a variety of examples. Some examples use spatially-dependent drift and diffusion functions. Problems from one to five dimensions are included. We first describe some experimental setup characteristics that are common among all examples.

Table 2: **DTQ parameter values by dimension**. The values of the parameters used in the examples in this manuscript. The two numbers given for the size of $\mathfrak{N}$ depend on if Leja points or nearest neighbors are used (see section 3.1).

| Parameter | $N = 1$ | $N = 2$ | $N = 3$ | $N = 4$ | $N = 5$ |
|---|---|---|---|---|---|
| $\epsilon$ | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 |
| $cond_{alt}$ | 5 | 5 | 5 | 5 | 5 |
| $LP_Q$ | 6 | 10 | 15 | 15 | 40 |
| size of $\mathfrak{N}$ | $LP_Q$ or 20 | $LP_Q$ or 20 | $LP_Q$ or 150 | $LP_Q$ or 200 | $LP_Q$ or 300 |
| size of $\mathfrak{Z}$ | 50 | 150 | 150 | 250 | 450 |
| $step_{AC}$ | 1 | 1 | 1 | 1 | 1 |
| $step_{RC}$ | 10 | 10 | 10 | 10 | 10 |
| $step_A$ | 1 | 1 | 1 | 1 | 1 |
| $step_R$ | 10 | 10 | 10 | 10 | 10 |

### 5.1.1 Parameter Values

Table 2 outlines some parameters which are consistent among all examples in this section. We note that these values are adjustable and all parameter values given are likely not optimal, but they work well for the examples we show. The size of $\mathfrak{N}$ depends on if Leja quadrature nodes or nearest neighbor points are used (see section 3.1). Additional parameter values for $\Delta min, \Delta max, \mathcal{R}, h$, and $\beta$ are given in each individual example.

### 5.1.2 Initial Condition and Initial Mesh

The initial condition in these examples is taken to be a Dirac mass centered at the origin, $\mathbf{0}$. We compute the first time step as $\widehat{p}(\mathbf{y}_j, t_h) = G(\mathbf{y}_j, \mathbf{0})$ where $G$ is defined in equation (4).

The initial mesh is an equispaced grid of points which is centered at the origin with mesh spacing $\Delta min$ and with a radius within the range $[\mathcal{R} - \Delta min, \mathcal{R}]$, where $\mathcal{R}$ is a parameter defining the initial mesh radius. This range occurs due to the fact that the mesh spacing may not evenly divide into the radius. Figure 4 shows initial mesh examples with $\mathcal{R} = 2$ in 1, 2, and 3 dimensions.

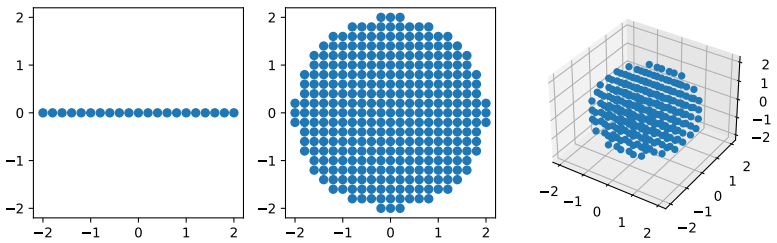

Figure 4: **Initial mesh**. Initial mesh examples with $\mathcal{R} = 2$ in $N = 1$, $N = 2$, and $N = 3$ dimensions respectively.

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

$$
\begin{aligned}
L_{2p} &= \sqrt{\frac{1}{\sum_{i=1}^{s} p(\mathbf{y}_i)} \sum_{i=1}^{s} \left( (p(\mathbf{y}_i) - \widehat{p}(\mathbf{y}_i))^2 p(\mathbf{y}_i) \right)} \\
L_2 &= \sqrt{\frac{1}{s} \sum_{i=1}^{s} (p(\mathbf{y}_i) - \widehat{p}(\mathbf{y}_i))^2} \\
L_1 &= \frac{1}{s} \sum_{i=1}^{s} |p(\mathbf{y}_i) - \widehat{p}(\mathbf{y}_i)| \\
L_\infty &= \max_{i \in [s]} |p(\mathbf{y}_i) - \widehat{p}(\mathbf{y}_i)|,
\end{aligned}
$$

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.

#### 5.1.3 Error, Leja Point Reuse, and Alternative Procedure Use Metrics

Furthermore, we report statistics on the average Leja point reuse and the alternative method use. Let $N_{steps}$ be the number of time steps taken in the simulation, then

$$
\text{Average Leja Reuse \%} = \frac{100}{N_{steps} - 2} \sum_{i=3}^{N_{steps}} \frac{\text{\# of points reusing Leja points at time } t_i}{\text{\# of points in the mesh at time } t_i}.
$$

We start at the third time step because Leja points are first computed in the second time step since the first time step is computed directly as discussed in section 5.1.2). Similarly,

$$
\text{Average Alt. Method Use \%} = \frac{100}{N_{steps} - 1} \sum_{i=2}^{N_{steps}} \frac{\text{\# of points using alt. method at time } t_i}{\text{\# of points in the mesh at time } t_i}.
$$

Again, we do not include the first time step because it is computed directly.

### 5.2 One-Dimensional Example

For this example, we consider the one-dimensional SDE with

$$
\mathbf{f}(x) = 2, \qquad \mathbf{g}(x) = 1.
$$

We set simulation parameters as $(\Delta min, \Delta max, \mathcal{R}, h, \beta) = (0.4, 0.4, 2, 0.05, 4)$. The average Leja reuse was 94% and the average alternative method use was 1.8%. The $L_{2p}$ error at $t = 10$ is 3.5e-05. The PDF at several times is shown in Figure 5.

### 5.3 Two-Dimensional Examples

Now, we will cover several examples using our adaptive DTQ procedure.

#### 5.3.1 Constant Drift and Diffusion

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

$$
\mathbf{f}(\mathbf{x}) = \begin{pmatrix} C_1 \\ 0 \end{pmatrix}, \qquad \mathbf{g}(\mathbf{x}) = \begin{pmatrix} C_2 & 0 \\ 0 & C_2 \end{pmatrix}. \tag{26}
$$

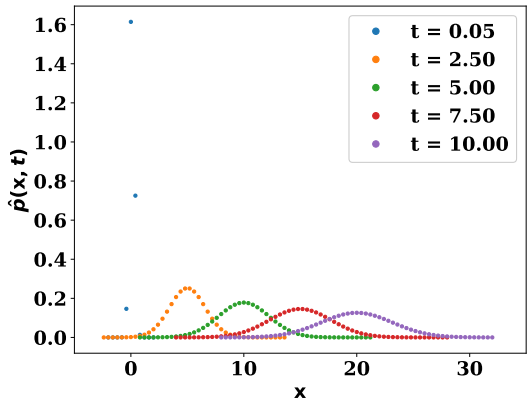


Figure 5: **One-dimensional constant drift and diffusion example**. This figure shows the computed density at several different times indicated in the legend.

This SDE corresponds to the Fokker-Planck PDE

$$\frac{\partial p(\mathbf{x},t)}{\partial t} = -C_1 \frac{\partial}{\partial x} p(\mathbf{x},t) + \frac{C_2^2}{2} \frac{\partial^2}{\partial (x^{(1)})^2} p(\mathbf{x},t) + \frac{C_2^2}{2} \frac{\partial^2}{\partial (x^{(2)})^2} p(\mathbf{x},t)$$

which has the exact solution

$$p(\mathbf{x},t) = \frac{1}{4\pi (C_2^2/2)t} \exp\left( \frac{-((x^{(1)} - C_1 t)^2 + (x^{(2)})^2)}{4(C_2^2/2)t} \right). \tag{27}$$

Here, we select $C_1 = C_2 = 1$, and we define $(\Delta min, \Delta max, \mathcal{R}, h) = (0.2, 0.2, 2, 0.01)$. We explore the effect of $\beta$, the tolerance parameter for allowable density values on boundary nodes, from section 4.2. The error at time $t = 1.15$ for varying $\beta$ are shown in Table 3.

Table 3: **Adaptive DTQ errors for different values of the boundary tolerance parameter $\beta$ at time $t = 1.15$ for the moving hill example of section 5.3.1**. Also shown are the number of points $s$ in the adaptive mesh at $t = 1.15$. Generally, as $\beta$ increases, the error decreases because we use more points to cover a larger area of the domain.

| $\beta$ | $L_{2p}$ Error | $L_2$ Error | $L_1$ Error | $L_\infty$ Error | # Points $s$ |
|---|---|---|---|---|---|
| 1 | 1.7e-02 | 1.6e-02 | 1.2e-03 | 3.5e-02 | 181 |
| 2 | 1.8e-03 | 1.6e-03 | 7.5e-05 | 2.7e-03 | 556 |
| 3 | 2.1e-04 | 2.2e-04 | 5.1e-06 | 3.3e-04 | 994 |
| 4 | 1.8e-05 | 1.9e-05 | 3.0e-07 | 3.0e-05 | 1436 |
| 5 | 1.4e-06 | 2.1e-06 | 1.7e-08 | 3.5e-06 | 1866 |
| 6 | 8.5e-08 | 2.1e-07 | 7.6e-10 | 4.3e-07 | 2319 |
| 7 | 8.5e-09 | 2.5e-08 | 6.2e-11 | 4.8e-08 | 2780 |
| 8 | 5.6e-10 | 2.6e-09 | 3.2e-12 | 8.0e-09 | 3295 |
| 9 | 2.3e-11 | 1.8e-10 | 8.0e-14 | 5.3e-10 | 3588 |
| 10 | 1.6e-12 | 1.5e-11 | 6.7e-15 | 4.7e-11 | 3996 |

We observe that $\beta$ has a somewhat direct control on error of the approach for this simple example. For larger $\beta$, more points are used to form a mesh on a larger region, which improves the overall

accuracy of the method. Table 3 shows that as $\beta$ increases, the error tends to decrease. Thus, the adaptive DTQ method can be quite accurate if $\beta$ is chosen appropriately. With that said, the selection of $\beta$ must be balanced with the computational cost associated with a larger mesh. The number of mesh points $s$ at the last time step is also shown.

### 5.3.2 Adaptive Leja Quadrature vs. Equispaced Trapezoidal DTQ

We will now compare and contrast our adaptive DTQ method with the trapezoidal rule DTQ method. For notational purposes, we will refer to the two differing DTQ methods as follows:

- $\text{DTQ}_{TR}$: non-adaptive density tracking by quadrature method using an equispaced mesh and a trapezoidal quadrature rule (section 2.2).
- $\text{DTQ}_{LQ}$ : adaptive density tracking by quadrature method using a interpolatory Leja quadrature rule with the ability to add and remove mesh points.

In many lower-dimensional problems (e.g. $N = 1, 2$) over a small domain, $\text{DTQ}_{TR}$ will be computationally faster than $\text{DTQ}_{LQ}$. With that said, $\text{DTQ}_{TR}$ will typically require more memory due to the need to form a dense grid. As the domain grows, even in smaller dimensions, $\text{DTQ}_{LQ}$ becomes more efficient than $\text{DTQ}_{TR}$, eventually becoming the preferred method. In this section, we formulate a two-dimensional example where $\text{DTQ}_{LQ}$ is both computationally faster to run and uses fewer mesh points than $\text{DTQ}_{TR}$. As the dimension of the problem increases, we expect that $\text{DTQ}_{LQ}$ will typically be the superior option of the two for many problems because $\text{DTQ}_{TR}$ will use significantly more points or simply be too memory intensive.

We compared the computational timing for $\text{DTQ}_{TR}$ against $\text{DTQ}_{LQ}$ for the constant drift and diffusion example in equation (26) with $C_1 = 1$ and $C_2 = 0.6$. The simulation parameters for $\text{DTQ}_{LQ}$ are $(\Delta min, \Delta max, \mathcal{R}, h) = (0.38, 0.38, 2, 0.05)$ with $\beta \in [2.5, 3, 4, 5, 6]$. For $\text{DTQ}_{TR}$, we use $h = 0.05$ and equispaced spatial step sizes of $\kappa \in [0.25, 0.2, 0.18, 0.15]$. Notice that we varied the accuracy and cost for each of these two methods by changing a parameter. For $\text{DTQ}_{LQ}$, we adjusted the boundary cutoff parameter, $\beta$, and for $\text{DTQ}_{TR}$ we adjusted the equispaced spatial step size $\kappa$.

Since we are considering a tensorized trapezoidal quadrature rule procedure that does not adaptively update the mesh, we need to determine the proper mesh for $\text{DTQ}_{TR}$ *a priori* using the mesh from $\text{DTQ}_{LQ}$ as a starting point. In this example, we used information about the meshes from the $\text{DTQ}_{LQ}$ result with $\beta = 4$ and added a 0% and 50% buffer. The buffer percentage is used to create a tensorized mesh with padding. Psuedocode for how this works is given in Algorithm 3 and a depiction of this is shown in Figure 6.

**Algorithm 3** Form tensorized mesh based on $\text{DTQ}_{LQ}$ mesh

**procedure** TensorizedMeshBasedOn$\text{DTQ}_{LQ}$Mesh($DTQ_{LQ}$ mesh, buffer):
    $[min_1, \ldots, min_N] = \min(DTQ_{LQ}$ mesh) in each dimension
    $[max_1, \ldots, max_N] = \max(DTQ_{LQ}$ mesh) in each dimension
    $[pad_1, \ldots, pad_N] =$ buffer/2*$[max_1 - min_1, \ldots, max_N - min_N]$
    grid $= [min_1 - pad_1, \ldots, min_N - pad_N] \times [max_1 + pad_1, \ldots, max_N + pad_N]$
    **return** grid

Essentially, we form a tensorized mesh for $\text{DTQ}_{TR}$ based on the $\text{DTQ}_{LQ}$ mesh with an additional buffer which depends on the coverage of the domain during the $\text{DTQ}_{LQ}$ simulation. We refer to the 0% buffer as an oracle solution because the knowledge needed to set a mesh which tightly supports the solution in this way requires complete knowledge of the solution support. The 50% buffer still requires knowledge of the solution; however it represents a much looser estimate than the oracle of the required mesh and is meant to be more indicative of an educated guess for the mesh.

Figure 7 shows the timings for each simulation recorded relative to the timing of the $\text{DTQ}_{LQ}$ simulation with $\beta = 2.5$, indicated by the star. The exact $L_{2p}$ errors are compared since the

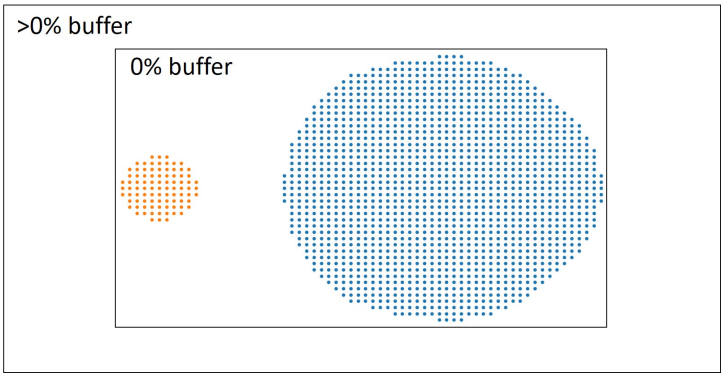


Figure 6: **Two-dimensional example of forming a tensorized mesh based on an adaptive mesh with a buffer. This figure illustrates how we form a tensorized mesh for DTQ$_{TR}$ based on the DTQ$_{LQ}$ mesh with an additional buffer. The orange and blue meshes depict the coverage during a DTQ$_{LQ}$ simulation at the starting and ending time respectively. The rectangular lines indicate the region the equispaced tensorized mesh will fill in given a buffer percentage of $0$ or $> 0$. Notice, the padding in each dimension can be different.**

exact solution is known (see equation (27)). Figure 7 shows that, for similar effort (relative time), DTQ$_{TR}$ takes about two (0% buffer) to ten (50% buffer) times longer than DTQ$_{LQ}$. Additionally, we emphasize again that DTQ$_{TR}$ requires information gathered from the DTQ$_{LQ}$ mesh in order to run the simulation.

Recall, the enforced mesh spacing for DTQ$_{LQ}$ was kept at 0.38 for all simulations since the $\beta$ parameter was adjusted for accuracy. This mesh spacing was much larger than the mesh spacing of 0.2 and 0.15 used by the accurate DTQ$_{TR}$ simulations. The number of points for each simulation are shown in Figure 8. We see that DTQ$_{TR}$ required one to two orders of magnitude more points to achieve similar error to DTQ$_{LQ}$. Also, we note that the recorded points shown in Figure 8 are at or near a maximum for the DTQ$_{LQ}$ procedure since they are reported at the final time step.

An AMD EPYC 7702P 64-Core Processor with 251 GiB of memory was used for this example.

### 5.3.3 Erf Drift Example

We consider the solution to the two-dimensional SDE in equation (1) with drift and constant diffusion

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

The error function used for the drift is defined as

$$\text{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt.$$

In this example involving the erf drift, we use DTQ$_{LQ}$ simulation parameters defined as $(\Delta min, \Delta max, \mathcal{R}, h, \beta) = (0.25, 0.3, 3, 0.04, 4)$. We also use a finely discretized DTQ$_{TR}$ simulation with parameters of $(\kappa, h) = (0.08, 0.01)$ for comparison. We would have preferred to use $\kappa = 0.05$ or smaller for accuracy purposes; however, running such an experiment was not feasible on available hardware due to memory limitations.

The simulation features a single mass breaking into four distinct hills and expanding outwards. The computed solutions of $\widehat{p}$ at various times are shown in Figure 9. The comparison shows that DTQ$_{LQ}$ with a lenient discretization holds its own against a finely discretized DTQ$_{TR}$ simulation. Here, DTQ$_{LQ}$ utilized a spatial distance between points that was more than three times larger than

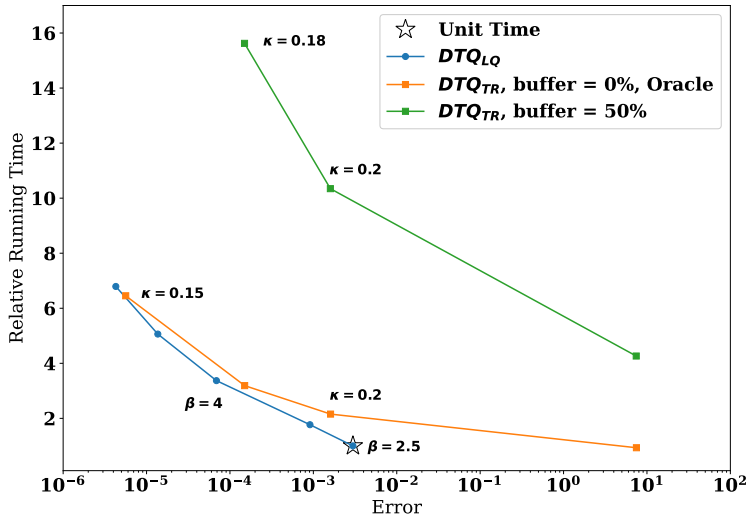


Figure 7: **DTQ$_{TR}$ and DTQ$_{LQ}$ timing comparison**. In this figure, we compare the error ($L_{2p}$) and timing between DTQ$_{TR}$ and DTQ$_{LQ}$ for a constant drift, constant diffusion example where the end time is $t = 40$.

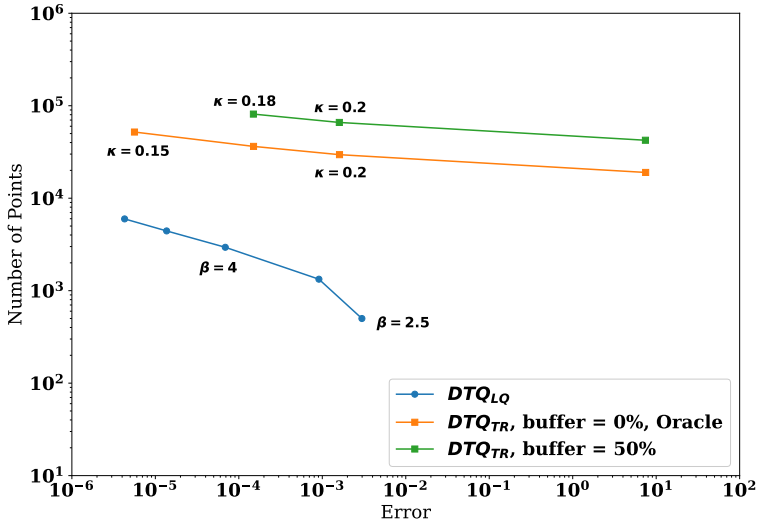


Figure 8: **DTQ$_{TR}$ and DTQ$_{LQ}$ end time mesh size comparison**. In this figure, we compare the number of points at the end time $t = 40$ between DTQ$_{TR}$ and DTQ$_{LQ}$ for a constant drift, constant diffusion example. The error is given in terms of $L_{2p}$. The trapezoidal rule uses the same number of points per time step whereas the adaptive procedure updates the mesh points to track the density.

the DTQ$_{TR}$ procedure and a temporal step size that was four times as large while still achieving very similar results. The average number of points used by DTQ$_{LQ}$ was about 2,000 per time step, and

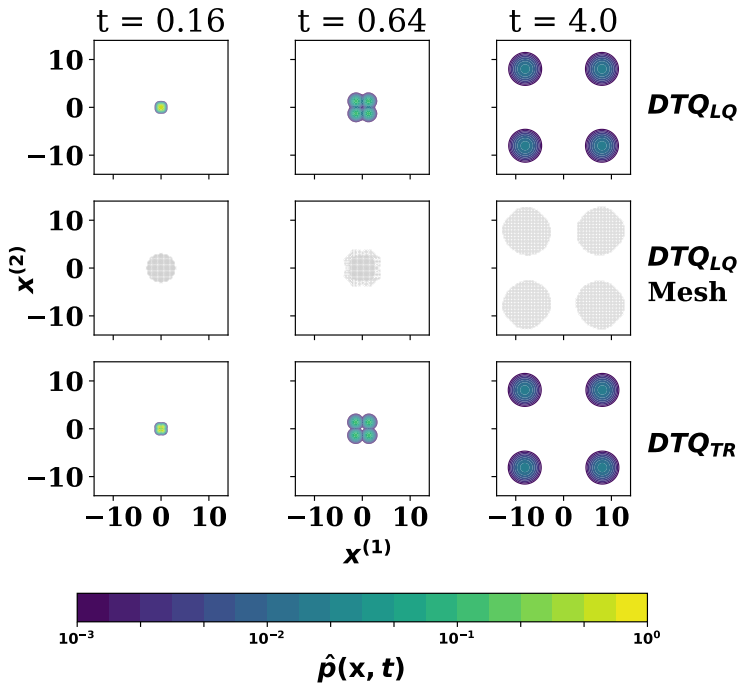


Figure 9: **Erf drift example.** The solution at the indicated times for the erf drift example of section 5.3.3. Top row: Density values of $\text{DTQ}_{LQ}$. Middle row: $\text{DTQ}_{LQ}$ mesh points. Bottom row: Density values of a finely discretized $\text{DTQ}_{TR}$. Density values below $10^{-3}$ are shown as white for visualization purposes.

the finely discretized $\text{DTQ}_{TR}$ simulation used over 120,000 per time step. If a less finely discretized grid was used for $\text{DTQ}_{TR}$, it would need to span about $[-13, 13] \times [-13, 13]$. Assuming the same spacing used by the $\text{DTQ}_{LQ}$ and using $\kappa = 0.25$, the coarser $\text{DTQ}_{TR}$ simulation would still require about 11,000 points for such a grid, which is still significantly larger than the $\text{DTQ}_{LQ}$ mesh size. Additionally, *a priori* knowledge of the necessary grid domain size is needed for $\text{DTQ}_{TR}$, which depends on the end time.

In this simulation, the percent of Leja points reused from the previous time step averaged about 84% per time step. The alternative procedure from section 3.1.1 was used for approximately 1.2% of the mesh on average per time step. The computation of these metrics is discussed in section 5.1.3.

Figure 10 shows the breakdown of time spent in the $\text{DTQ}_{LQ}$ algorithm for the erf function drift example. Figure 10(left) shows the high level breakdown between the quadrature rule and the mesh updates. Figure 10(right) gives the breakdown of the time spent computing the different pieces of the quadrature rule.

### 5.3.4 Spiral Example

We now consider the solution to the SDE in equation (1) in two dimensions where

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

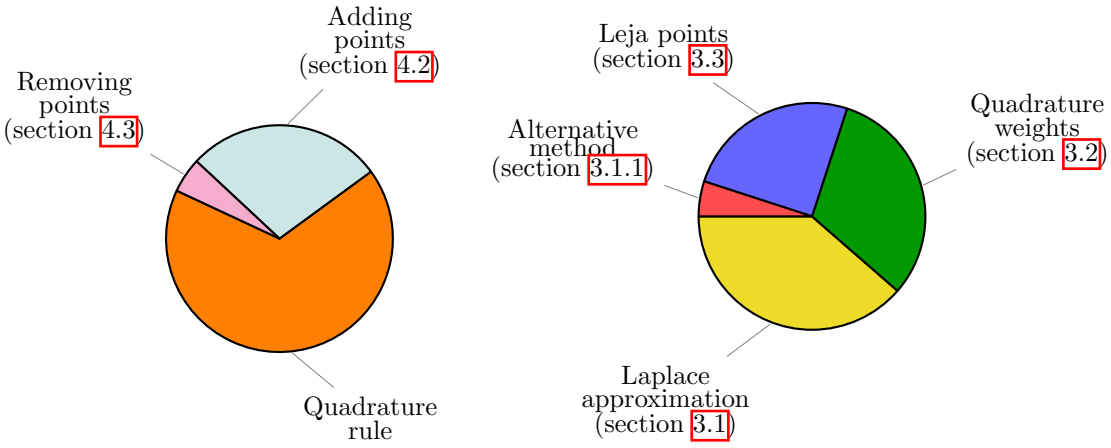


Figure 10: Breakdown of DTQ$_{LQ}$ algorithm for the erf function drift example in section 5.3.3. (Left) Breakdown of time spent between approximating the Chapman-Kolmogorov equation and mesh updates. (Right) Breakdown of the algorithm components which make up the "Quadrature rule" slice of the left chart.

with DTQ$_{LQ}$ simulation parameters $(\Delta min, \Delta max, \mathcal{R}, h, \beta) = (0.2, 0.2, 2, 0.02, 4)$ and the finely discretized DTQ$_{TR}$ simulation parameters were $(\kappa, h) = (0.05, 0.01)$. The solution to this problem features a single mass splitting into two and rotating in a clockwise spiral.

On average, DTQ$_{LQ}$ reused Leja points approximately 83% of the time per time step, and the alternative method was used about 0.01% of the time. The computation of these metrics is discussed in section 5.1.3. The computed solutions at various times are shown in Figure 11. We again see very similar results between the two procedures.

#### 5.3.5 Nonconstant Diffusion

Now we consider the SDE in equation (1) in two dimensions with a nonconstant diffusion and drift defined as

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

This is the first example with a spatially dependent diffusion term. The DTQ$_{LQ}$ simulation parameter values were $(\Delta min, \Delta max, \mathcal{R}, h, \beta) = (0.2, 0.2, 2, 0.02, 4)$ and the finely discretized DTQ$_{TR}$ simulation parameters were $(\kappa, h) = (0.05, 0.01)$. On average, Leja points were reused approximately 87% per time step, and the alternative method was used for 0% of the mesh points for each time step. The computation of these metrics is discussed in section 5.1.3. The solution at various times is shown in Figure 12.

This example was included to illustrate that DTQ$_{LQ}$ can simulate a spatially dependent drift. We see very similar results between DTQ$_{LQ}$ and a finely discretized DTQ$_{TR}$.

### 5.4 Higher Dimensional Examples

#### 5.4.1 Constant Drift and Diffusion

We now consider the solution to the SDE in equation (1) with constant drift and diffusion in higher dimensions. For $N$ dimensions, we take

$$\mathbf{f}(\mathbf{x}) = C_1\mathbf{e}_1, \qquad \mathbf{g}(\mathbf{x}) = C_2\mathbf{I}_{N\times N}$$

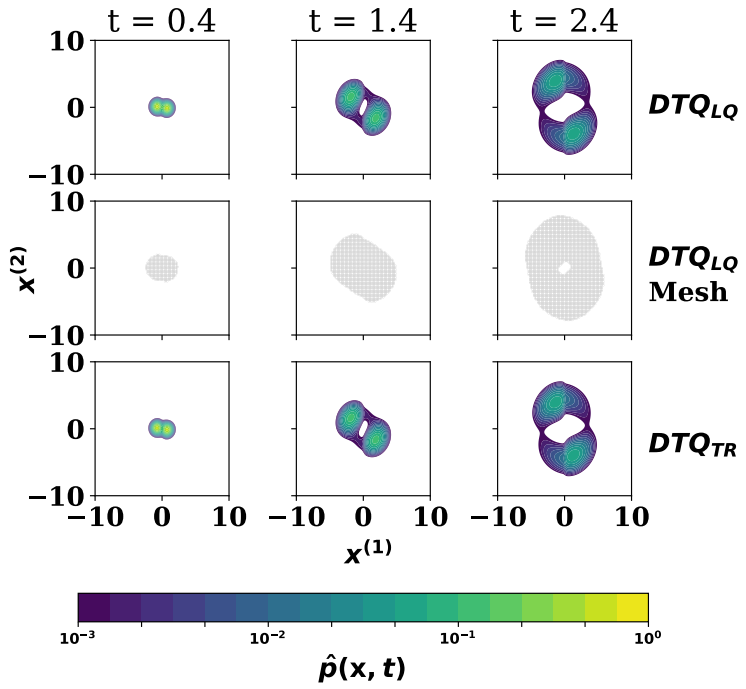


Figure 11: **Spiral example.** The solution at the indicated times for the spiral drift example of section 5.3.4. Top row: Density values of $\text{DTQ}_{LQ}$. Middle row: $\text{DTQ}_{LQ}$ mesh points. Bottom row: Density values of a finely discretized $\text{DTQ}_{TR}$. Density values below $10^{-3}$ are shown as white for visualization purposes.

where $\mathbf{I}_{N\times N}$ is defined as the the $N \times N$-dimensional identity matrix and $\mathbf{e}_1$ is the cardinal $N$-dimensional unit vector which has a 1 as the first element and zeros elsewhere. This SDE corresponds to the Fokker-Planck PDE

$$\frac{\partial p(\mathbf{x},t)}{\partial t} = -C_1 \frac{\partial}{\partial x} p(\mathbf{x},t) + \sum_{i=1}^{N} \frac{C_2^2}{2} \frac{\partial^2}{\partial (x^{(i)})^2} p(\mathbf{x},t)$$

which has the exact solution

$$p(\mathbf{x},t) = \left(\frac{1}{4\pi(C_2^2/2)t}\right)^{N/2} \exp\left(\frac{-((x^{(1)} - C_1 t)^2 + \sum_{i=2}^{N}(x^{(i)})^2)}{4(C_2^2/2)t}\right).$$

For these examples, we take $C_1 = 1$ and $C_2 = 0.6$. In the three-dimensional example, we set simulation parameters as $(\Delta min, \Delta max, \mathcal{R}, h, \beta) = (0.22, 0.22, 1, 0.02, 3)$, in the four-dimensional example we set simulation parameters as $(\Delta min, \Delta max, \mathcal{R}, h, \beta) = (0.18, 0.18, 0.8, 0.02, 3)$, and in the the five-dimensional example we set simulation parameters as $(\Delta min, \Delta max, \mathcal{R}, h, \beta) = (0.1, 0.1, 0.5, 0.01, 3)$. The results for these simulations are in Table 4.

The purpose of this example is to illustrate that we can use $\text{DTQ}_{LQ}$ successfully in $N \geq 3$ dimensions. While we are able to demonstrate feasibility up to $N = 5$ dimensions, computational constraints on memory play a role for such large dimensions. We estimate that the mesh size for

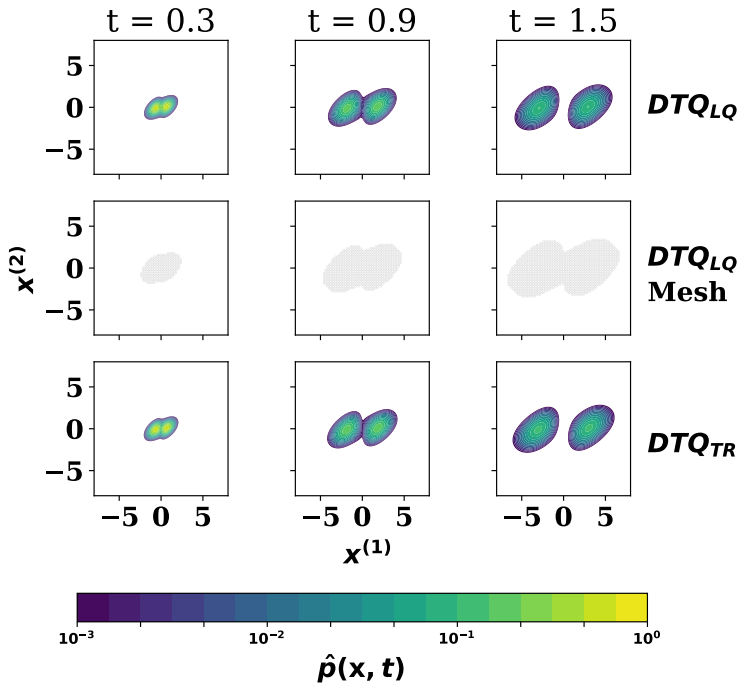


Figure 12: **Nonconstant diffusion example.** The solution at the indicated times for the nonconstant drift example of section 5.3.5. Top row: Density values of $\text{DTQ}_{LQ}$. Middle row: $\text{DTQ}_{LQ}$ mesh points. Bottom row: Density values of a finely discretized $\text{DTQ}_{TR}$. Density values below $10^{-3}$ are shown as white for visualization purposes.

Table 4: **Constant drift and diffusion error for $N = 3, 4, 5$ dimensions**

| $N$ | End Time | $h$ | $L_{2p}$ Error | Starting Mesh Size | Ending Mesh Size |
|---|---|---|---|---|---|
| 3 | 1 | 0.02 | 0.00014 | 437 | 4,144 |
| 4 | 0.5 | 0.02 | 0.00017 | 2,041 | 38,678 |
| 5 | 0.04 | 0.01 | 0.0086 | 16,875 | 38,089 |

$\text{DTQ}_{TR}$ in five dimensions that accurately covers the necessary support would be around 370,000 points, assuming a spatial grid spacing of 0.1 on a domain spanning from about $-0.6$ to 0.6 in each dimension.

### 5.4.2 Three-dimensional Error Function Drift

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

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

In this three-dimensional erf drift example, we use $\text{DTQ}_{LQ}$ simulation parameters defined as $(\Delta min, \Delta max, \mathcal{R}, h, \beta)$ $= (0.25, 0.25, 1, 0.02, 3)$. We visualize the density using a color scale from white to black. The darker the color, the lager the density. The solution at time $t = 1.22$ is shown in Figure 13. We note that the adaptive mesh has broken into eight different clusters in order to capture areas of high density without needing mesh points in areas of low density.

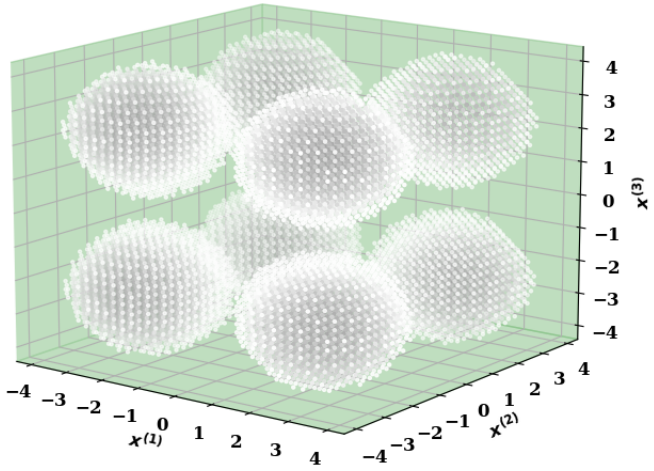


Figure 13: **Three-dimensional erf drift example. The density is shown at $t = 1.22$. The three spatial dimensions are plotted and the density is represented by the color of each point with the darker colors representing larger density.**

# 6 Conclusion

We have implemented an accurate and adaptive $N$-dimensional DTQ solver that uses fewer mesh points than current trapezoidal DTQ procedures and does not require *a priori* knowledge of the necessary mesh coverage and domain. We included examples of the method when applied to problems from one to five dimensions. The results in Figure 7 and Figure 8 show that our adaptive DTQ procedure can be significantly more efficient while using substantially fewer mesh points when compared to DTQ using the trapezoidal quadrature rule on a tensorized grid.

[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 Probablistic 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. L. Cotter, T. Vejchodsky, and R. Erban. “Adaptive finite element method assisted by stochastic simulation of chemical systems”. In: *SIAM Journal on Scientific Computing* 35.1 (2013), B107–B131.

[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.

[17] 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.

[18] 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.

[19] A. Naess and J. M. Johnsen. "Response statistics of nonlinear, compliant offshore structures by the path integral solution method". In: *Probabilistic Engineering Mechanics* 8.2 (1993), pp. 91–106. ISSN: 0266-8920. DOI: https://doi.org/10.1016/0266-8920(93)90003-E.

[20] J. S. Yu, G. Q. Cai, and Y. K. Lin. "A new path integration procedure based on Gauss-Legendre scheme". In: *International Journal of Non-linear Mechanics* 32 (1997), pp. 759–768.

[21] M. Rosa-Clot and S. Taddei. "A Path Integral Approach to Derivative Security Pricing: II. Numerical Methods". In: *International Journal of Theoretical and Applied Finance* 05 (1999), pp. 123–146.

[22] C. Skaug and A. Naess. "Fast and accurate pricing of discretely monitored barrier options by numerical path integration". In: *Computational Economics* 30 (2007), pp. 143–151.

[23] Vadim Linetsky. "The Path Integral Approach to Financial Modeling and Options Pricing". In: *Computational Economics* 11 (1997), pp. 129–163.

[24] 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.

[25] L. Chen, E. R. Jakobsen, and A. Naess. "On numerical density approximations of solutions of SDEs with unbounded coefficients". In: *Advances in Computational Mathematics* 44.3 (2018), pp. 693–721.

[26] H. S. Bhat and R. W. M. A. Madushani. "Density Tracking by Quadrature for Stochastic Differential Equations". In: *ArXiv* (2018).

[27] Vlad Bally and Denis Talay. "The Law of the Euler Scheme for Stochastic Differential Equations: II. Convergence Rate of the Density". In: *Monte Carlo Methods and Applications* 2 (Dec. 1995). DOI: 10.1515/mcma.1996.2.2.93.

[28] H. S. Bhat, R. W. M. A. Madushani, and S. 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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).

[34] 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.

[35] 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.

[36] 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.

[37] J. D. Jakeman. *Adaptive Leja Sequences*. 2019.

[38] B. Delaunay. "Sur la sphere vide". In: *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk* 7.793-800 (1934), pp. 1–2.

[39] H. Edelsbrunner. "Alpha shapes—a survey". In: *Tessellations in the Sciences* 27 (2010), pp. 1–25.

[40] R. A. Moore. *AdaptiveDTQ*. `https://github.com/RyleighAMoore/AdaptiveDTQND`. 2021.