SIAM J. SCI. COMPUT.
Vol. 40, No. 4, pp. A2033–A2061

NUMERICAL INTEGRATION IN MULTIPLE DIMENSIONS WITH DESIGNED QUADRATURE*

VAHID KESHAVARZZADEH†, ROBERT M. KIRBY‡, AND AKIL NARAYAN§

Abstract. We present a systematic computational framework for generating positive quadrature rules in multiple dimensions on general geometries. A direct moment-matching formulation that enforces exact integration on polynomial subspaces yields nonlinear conditions and geometric constraints on nodes and weights. We use penalty methods to address the geometric constraints and subsequently solve a quadratic minimization problem via the Gauss–Newton method. Our analysis provides guidance on requisite sizes of quadrature rules for a given polynomial subspace and furnishes useful user-end stability bounds on error in the quadrature rule in the case when the polynomial moment conditions are violated by a small amount due to, e.g., finite precision limitations or stagnation of the optimization procedure. We present several numerical examples investigating optimal low-degree quadrature rules, Lebesgue constants, and 100-dimensional quadrature. Our capstone examples compare our quadrature approach to popular alternatives, such as sparse grids and quasi-Monte Carlo methods, for problems in linear elasticity and topology optimization.

Key words. numerical integration, multiple dimensions, polynomial approximation, quadrature optimization

AMS subject classifications. 41A55, 65D32

DOI. 10.1137/17M1137875

1. Introduction.

Numerical quadrature, the process of computing approximations to integrals, is widely used in many fields of science and engineering. A convenient and popular choice is a quadrature rule that uses point evaluations of a function f:

$$\int_\Gamma f(\boldsymbol{x})\omega(\boldsymbol{x})\mathrm{d}x \approx \sum_{j=1}^{n} f(\boldsymbol{x}_j)w_j,$$

where Γ is some set in d-dimensional Euclidean space $\mathbb{R}^d$, ω is a positive weight function, and $\boldsymbol{x}_j$ and w_j are the nodes and weights, respectively, of the quadrature rule that must be determined. The main desirable properties of quadrature rules are accuracy for a broad class of functions, a small number n of nodes/weights, and positivity of the weights. (Positive weights are desired so that the absolute condition number of the quadrature rule is controlled.)

In one dimension, Gaussian quadrature rules [29, 44] satisfy many of these desirable properties, but computing an efficient quadrature rule (or "cubature" rule) for

*Submitted to the journal's Methods and Algorithms for Scientific Computing section July 10, 2017; accepted for publication (in revised form) April 17, 2018; published electronically July 3, 2018.
http://www.siam.org/journals/sisc/40-4/M113787.html

Funding: This work was supported by ARL under Cooperative Agreement W911NF-12-2-0023. The work of the first and third authors was partially supported by AFOSR FA9550-15-1-0467. The work of the third author was partially supported by DARPA EQUiPS N660011524053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of ARL or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

†Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT 84112 (vkeshava@sci.utah.edu).

‡School of Computing, University of Utah, Salt Lake City, UT 84112 (kirby@sci.utah.edu).

§Department of Mathematics, University of Utah, Salt Lake City, UT 84112 (akil@sci.utah.edu).

higher dimensions is a considerably more challenging problem. When Γ and ω are of tensor-product form, one straightforward construction results from tensorization of univariate quadrature rules. However, the computational complexity required to evaluate f at the nodes of a tensorized quadrature rule quickly succumbs to the curse of dimensionality.

Substantial progress has been made in constructing attractive multivariate quadrature rules. Sparse grids rely on a sophisticated manipulation of univariate quadrature rules [7, 16]. Quasi-Monte Carlo methods generate sequences that have low-discrepancy properties [33, 34, 37]. Mathematical characterizations of quadrature rules with specified exactness on polynomial spaces yield efficient nodes and weights [5, 9, 42, 56].

The main contribution of this paper is a systematic computational approach for designing multivariate quadrature rules with exactness on general finite-dimensional polynomial spaces. Using polynomial exactness as a desideratum for constructing quadrature rules is not the only approach one could use (e.g., quasi-Monte Carlo methods do not adopt this approach). However, when the integrand f can be accurately approximated by a polynomial expansion with a small number of significant terms, then approximating the integral with a quadrature rule that is designed to integrate the significant terms can be very efficient [10, 11]. In particular, finite-dimensional polynomial spaces can well-approximate solutions to some parametric operator equations [12], and empirical tests with many engineering problems show that polynomial approximations are very efficient [1, 2, 8].

Our computational approach revolves around optimization; many algorithms for computing nodal sets via optimization have already been proposed [29, 30, 36, 45, 46, 48, 53]. Our method, which we call *designed quadrature*, has the following advantages:

- we can successfully compute nodal sets in up to 100 dimensions;
- positivity of the weights is ensured;
- quadrature rules over nonstandard geometries can be computed; and
- a prescribed polynomial accuracy can be sought over general polynomial spaces, not restricted to, e.g., total degree spaces.

Our approach is simple: we formulate moment-matching conditions and geometric constraints that prescribe nonlinear conditions on the nodes and weights. This direct formulation allows significant flexibility with respect to geometry, weight function ω, and polynomial accuracy. Indeed, our procedures can compute quadrature rules with hyperbolic cross-polynomial spaces (section 4.5) and can constrain nodal locations to awkward geometries (see section 4.4). Our computational approach is to use constrained optimization algorithms to compute a quadrature rule from the moment-matching conditions. Our mathematical analysis provides a stability bound on error of the quadrature rule if the moment-matching conditions are violated (e.g., due to numerical finite precision). We apply our designed quadrature rules to several realistic problems in computational science, including problems in linear elasticity and topology optimization. Comparisons against competing methods, such as sparse grids and low-discrepancy sequences, illustrate that designed quadrature often attains superior accuracy with many fewer nodes.

Our procedure is not without shortcomings: Being a direct moment-matching problem, our framework relies on large-scale optimization in high dimensions. For a specified polynomial subspace on which we require integration accuracy, we cannot a priori determine the number of nodes that our procedure will produce (although we review some theory that provides upper and lower bounds for n). We likewise cannot ensure that our algorithm produces an optimal quadrature rule size, but our

numerical results suggest favorable comparison with alternative techniques; see section 4.2. Some of the optimization tools we use have tunable parameters; we have made automated choices for these parameters but leave to future work proving that the algorithm performs well for arbitrary dimensions, weight functions, or polynomial spaces.

This paper is organized as follows. In section 2 we discuss the mathematical setting and formulate the optimization problem. This section also presents theory for the requisite number of nodes and stability of quadrature rules for approximate moment-matching. Section 3 details the computational framework for generating designed quadrature rules. Numerical results are shown in section 4.

2. Multivariate quadrature.

2.1. Notation. Let ω be a given nonnegative weight function (e.g., a probability density function) whose support is $\Gamma \subset \mathbb{R}^d$, where $d \geq 1$ and Γ need not be compact. A point $\boldsymbol{x} \in \mathbb{R}^d$ has components $\boldsymbol{x} = \left(x^{(1)}, x^{(2)}, \ldots, x^{(d)}\right)$. The space $L^2_\omega(\Gamma)$ is the set of functions f defined by

$$L^2_\omega(\Gamma) = \left\{ f : \Gamma \to \mathbb{R} \;\middle|\; \|f\| < \infty \right\}, \quad \|f\|^2 = (f,f), \quad (f,g) = \int_\Gamma f(\boldsymbol{x}) g(\boldsymbol{x}) \omega(\boldsymbol{x}) \mathrm{d}\boldsymbol{x}.$$

We use standard multi-index notation: $\boldsymbol{\alpha} \in \mathbb{N}_0^d$ denotes a multi-index, and Λ a collection of multi-indices. We have

$$\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_d), \qquad \boldsymbol{x}^{\boldsymbol{\alpha}} = \prod_{j=1}^d \left(x^{(j)}\right)^{\alpha_j}, \qquad |\boldsymbol{\alpha}| = \sum_{j=1}^d \alpha_j.$$

We impose a partial ordering on multi-indices via componentwise comparisons: with $\boldsymbol{\alpha}$, $\boldsymbol{\beta} \in \mathbb{N}_0^d$, then $\boldsymbol{\alpha} \leq \boldsymbol{\beta}$ if and only if all componentwise inequalities are true. A multi-index set Λ is called *downward closed* if

$$\boldsymbol{\alpha} \in \Lambda \quad \Longrightarrow \quad \boldsymbol{\beta} \in \Lambda \quad \forall\, \boldsymbol{\beta} \leq \boldsymbol{\alpha}.$$

We assume throughout this paper that the weight function has finite polynomial moments of all orders:

$$\int_\Gamma \left(\boldsymbol{x}^{\boldsymbol{\alpha}}\right)^2 \omega(\boldsymbol{x}) < \infty, \qquad \boldsymbol{\alpha} \in \mathbb{N}_0^d.$$

This assumption ensures existence of polynomial moments. Our ultimate goal is to construct a set of n points $\{\boldsymbol{x}_q\}_{q=1}^n \subset \Gamma$ and positive weights $w_q > 0$ such that

$$I(f) = \int_\Gamma f(\boldsymbol{x}) \omega(\boldsymbol{x}) \mathrm{d}x \approx \sum_{q=1}^n w_q f(\boldsymbol{x}_q) \tag{1a}$$

for functions f within a "large" class of functions. We attempt to achieve this by enforcing equality above for f in a subspace Π of polynomials:

$$\int_\Gamma f(\boldsymbol{x}) \omega(\boldsymbol{x}) \mathrm{d}x = \sum_{q=1}^n w_q f(\boldsymbol{x}_q), \qquad f \in \Pi. \tag{1b}$$

The quadrature strategy is accurate if f can be well-approximated by a polynomial from Π. There are numerous technical conditions on Π and f that yield quantitative

statements about polynomial approximation accuracy; see, e.g., [3]. In this article, we assume that Π is given and fixed through some a priori study ensuring that there exists a polynomial in Π that accurately approximates f to within some user-specified tolerance. Typically we will define Π through some finite multi-index set Λ:

$$\Pi = \operatorname{span}\left\{ \boldsymbol{x}^{\boldsymbol{\alpha}} \;\middle|\; \boldsymbol{\alpha} \in \Lambda \right\}.$$

In many applications, the function f typically exhibits smoothness (e.g., integrable high-order derivatives), which in turn implies that polynomial approximations converge at a high order with respect to the degree of approximation. Under the assumption that f is smooth, we therefore expect that the integral of a polynomial that approximates f is a good approximation if the approximating polynomial space Π contains high-degree polynomials. Our main goal in this paper is then familiar when viewed through the lens of classical analysis: make Π as large as possible while keeping n as small as possible.

Two particularly popular choices for polynomial spaces Π can be defined by the index sets

$$\Lambda_{\mathcal{T}_r} = \left\{ \boldsymbol{\alpha} \in \mathbb{N}_0^d \;\middle|\; |\boldsymbol{\alpha}| \le r \right\}, \qquad \Lambda_{\mathcal{H}_r} = \left\{ \boldsymbol{\alpha} \in \mathbb{N}_0^d \;\middle|\; \prod_{j=1}^{d} (\alpha_j + 1) \le r + 1 \right\}$$

for some nonnegative integer r. Both of these multi-index sets are downward closed. The total order and hyperbolic cross-polynomial subspaces are defined by, respectively,

$$\Pi_{\mathcal{T}_r} = \operatorname{span}\left\{ \boldsymbol{x}^{\boldsymbol{\alpha}} \;\middle|\; \boldsymbol{\alpha} \in \Lambda_{\mathcal{T}_r} \right\}, \qquad \Pi_{\mathcal{H}_r} = \operatorname{span}\left\{ \boldsymbol{x}^{\boldsymbol{\alpha}} \;\middle|\; \boldsymbol{\alpha} \in \Lambda_{\mathcal{H}_r} \right\}. \tag{2}$$

The algorithm we present in this paper applies to general polynomial spaces, but our numerical examples will focus on the spaces above since they are common in large-scale computing problems.

2.2. Univariate rules: Gauss quadrature. When $\Gamma \subset \mathbb{R}$, the optimal quadrature rule is provided by the ω-Gauss quadrature rule. In one dimension, we use the shorthand $\Pi_k = \Pi_{\mathcal{T}_k}$. The first step in defining this rule is to prescribe an orthonormal basis for Π_k. A Gram–Schmidt argument implies that such a basis of orthonormal polynomials exists with elements $p_m(\cdot)$, where $\deg p_m = m$. All univariate orthonormal polynomial families satisfy the three-term recurrence relation,

$$x p_m(x) = \sqrt{b_m}\, p_{m-1}(x) + a_m p_m(x) + \sqrt{b_{m+1}}\, p_{m+1}(x), \tag{3}$$

for $m \ge 0$, with $p_{-1} \equiv 0$ and $p_0 \equiv 1/\sqrt{b_0}$ to seed the recurrence. The recurrence coefficients are given by

$$a_m = (x p_m, p_m), \qquad b_m = \frac{(p_m, p_m)}{(p_{m-1}, p_{m-1})}$$

for $m \ge 0$, with $b_0 = (p_0, p_0)$. Classical orthogonal polynomial families, such as the Legendre and Hermite polynomials, fit this mold with explicit formula for the a_n and b_n coefficients [44]. Gaussian quadrature rules are n-point rules that exactly integrate polynomials in Π_{2n-1} [39, 14].

THEOREM 2.1 (Gaussian quadrature). *Let $x_1, \ldots, x_n$ be the roots of the nth orthogonal polynomial $p_n(x)$, and let $w_1, \ldots, w_n$ be the solution of the system of equations*

$$\sum_{q=1}^{n} p_j(x_q) w_q = \begin{cases} \sqrt{b_0} & \text{if } j = 0, \\ 0 & \text{for } j = 1, \ldots, n-1. \end{cases} \tag{4}$$

Then $x_q \in \Gamma$ *and* $w_q > 0$ *for* $q = 1, 2, \ldots, n$ *and*

$$\int_\Gamma \omega(x)p(x)dx = \sum_{q=1}^{n} p(x_q)w_q \tag{5}$$

holds for all polynomials $p \in \Pi_{2n-1}$.

Historically significant algorithmic strategies for computing Gauss quadrature rules are given in [15, 18]. The elegant linear algebraic formulations described in these references compute the quadrature rule with knowledge of only of a finite number of recurrence coefficients a_n, b_n.

2.3. Multivariate polynomials. If Γ and $\omega(\boldsymbol{x})$ are both tensorial, then the generalization of univariate orthogonal polynomials to multivariate ones is straightforward. The tensorial structure implies

$$\Gamma = \times_{j=1}^{d} \Gamma_j, \qquad \omega(\boldsymbol{x}) = \prod_{j=1}^{d} \omega_j\left(x^{(j)}\right)$$

for univariate domains $\Gamma_j \subset \mathbb{R}$ and univariate weights $\omega_j(\cdot)$. If $p_n^{(j)}(\cdot)$ is the univariate orthonormal polynomial family associated with ω_j over Γ_j, then

$$\pi_{\boldsymbol{\alpha}}(\boldsymbol{x}) = \prod_{j=1}^{d} p_{\alpha_j}^{(j)}\left(x^{(j)}\right), \qquad \boldsymbol{\alpha} \in \mathbb{N}_0^d, \tag{6}$$

defines a family of multivariate polynomials orthonormal under ω, i.e., $(\pi_{\boldsymbol{\alpha}}, \pi_{\boldsymbol{\beta}}) = \delta_{\boldsymbol{\alpha},\boldsymbol{\beta}}$, where δ is the Kronecker delta. The polynomial spaces in (2) can be written as

$$\Pi_{\mathcal{T}_r} = \operatorname{span}\left\{\pi_{\boldsymbol{\alpha}} \;\middle|\; \boldsymbol{\alpha} \in \Lambda_{\mathcal{T}_r}\right\}, \qquad \Pi_{\mathcal{H}_r} = \operatorname{span}\left\{\pi_{\boldsymbol{\alpha}} \;\middle|\; \boldsymbol{\alpha} \in \Lambda_{\mathcal{H}_r}\right\}.$$

The following result is the cornerstone of our algorithm.

PROPOSITION 2.2. *Let* Λ *be a multi-index set with* $\mathbf{0} \in \Lambda$. *Suppose that* $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ *and* $w_1, \ldots, w_n$ *are the solution of the system of equations*

$$\sum_{q=1}^{n} \pi_{\boldsymbol{\alpha}}(\boldsymbol{x}_q)w_q = \begin{cases} 1/\pi_{\mathbf{0}} & \text{if } \boldsymbol{\alpha} = \mathbf{0}, \\ 0 & \text{if } \boldsymbol{\alpha} \in \Lambda \backslash \{\mathbf{0}\}; \end{cases} \tag{7}$$

then

$$\int_{\boldsymbol{\Gamma}} \omega(\boldsymbol{x})\pi(\boldsymbol{x})d\boldsymbol{x} = \sum_{q=1}^{n} \pi(\boldsymbol{x}_q)w_q \tag{8}$$

holds for all polynomials $\pi \in \Pi_\Lambda$.

The proof is straightforward by noting that $\int_\Gamma \pi_{\boldsymbol{\alpha}}(\boldsymbol{x})\omega(\boldsymbol{x})\mathrm{d}\boldsymbol{x} = 0$ when $\boldsymbol{\alpha} \neq \mathbf{0}$ due to orthogonality, and thus (7) is a moment-matching condition. Unlike Theorem 2.1, this multivariate result does not guarantee the positivity of weights, nor does it ensure that the nodes lie in Γ. We enforce these conditions in our computational framework in section 3. Finally, we note that Proposition 2.2 is true even when Γ and ω are not tensorial. We concentrate on the tensorial situation in this paper because a tensorial assumption is standard for large dimension d.

One of the main uses of quadrature rules is in the construction of polynomial approximation via discrete quadrature. If f is a given continuous function and Θ is a given multi-index set, then

$$f(\boldsymbol{x}) \approx f_\Theta(\boldsymbol{x}) = \sum_{\alpha\in\Theta} \widehat{f}_{\boldsymbol{\alpha}} \pi_{\boldsymbol{\alpha}}(\boldsymbol{x}), \qquad \widehat{f}_{\boldsymbol{\alpha}} = \sum_{q=1}^{n} \pi_{\boldsymbol{\alpha}}\left(\boldsymbol{x}_q\right) f\left(\boldsymbol{x}_q\right) w_q, \tag{9}$$

where $\widehat{f}_{\boldsymbol{\alpha}}$ are meant to approximate the Fourier (L^2_ω-projection) coefficients of f. Ideally, if $f \in \Pi_\Theta$, then $f_\Theta = f$; i.e., this construction reproduces polynomials in Π_Θ. As one expects, this only happens when the quadrature rule is sufficiently accurate, as defined by the size of Λ in (7).

PROPOSITION 2.3. *Let Λ be a downward-closed multi-index set, and suppose that $\boldsymbol{x}_q$ and w_q for $q = 1, \ldots, n$ define a quadrature rule satisfying* (7)*. Let Θ be any index set satisfying*

$$\Theta + \Theta = \left\{\boldsymbol{\alpha} + \boldsymbol{\beta} \;\middle|\; \boldsymbol{\alpha}, \boldsymbol{\beta} \in \Theta\right\} \subseteq \Lambda. \tag{10}$$

If $f \in \Pi_\Theta$, then f_Θ defined in (9) *satisfies $f_\Theta = f$.*

Proof. Suppose $f \in \Pi_\Theta$, so that

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{\alpha}\in\Theta} f_{\boldsymbol{\alpha}} \pi_{\boldsymbol{\alpha}}(\boldsymbol{x}), \qquad f_{\boldsymbol{\alpha}} = \left(f, \pi_{\boldsymbol{\alpha}}\right),$$

where the formula for the coefficients $f_{\boldsymbol{\alpha}}$ is due to orthogonality. We will show that the computed quadrature coefficients $\widehat{f}_{\boldsymbol{\alpha}}$ defined in (9) satisfy $\widehat{f}_{\boldsymbol{\alpha}} = f_{\boldsymbol{\alpha}}$. Fix $\boldsymbol{\beta} \in \Theta$. Then

$$f(\boldsymbol{x})\pi_{\boldsymbol{\beta}}(\boldsymbol{x}) = \sum_{\boldsymbol{\alpha}\in\Theta} f_{\boldsymbol{\alpha}} \pi_{\boldsymbol{\alpha}}(\boldsymbol{x})\pi_{\boldsymbol{\beta}}(\boldsymbol{x}).$$

There are coefficients $c_{\boldsymbol{\alpha},\boldsymbol{\gamma}}$ such that

$$\pi_{\boldsymbol{\alpha}} = \sum_{\boldsymbol{\gamma}\le\boldsymbol{\alpha}} c_{\boldsymbol{\alpha},\boldsymbol{\gamma}} \boldsymbol{x}^{\boldsymbol{\gamma}}.$$

Therefore,

$$\pi_{\boldsymbol{\alpha}}(\boldsymbol{x})\pi_{\boldsymbol{\beta}}(\boldsymbol{x}) = \left(\sum_{\boldsymbol{\gamma}\le\boldsymbol{\alpha}} c_{\boldsymbol{\alpha},\boldsymbol{\gamma}} \boldsymbol{x}^{\boldsymbol{\gamma}}\right)\left(\sum_{\boldsymbol{\gamma}\le\boldsymbol{\beta}} c_{\boldsymbol{\beta},\boldsymbol{\gamma}} \boldsymbol{x}^{\boldsymbol{\gamma}}\right) = \sum_{\boldsymbol{\gamma}\le\boldsymbol{\alpha}+\boldsymbol{\beta}} d_{\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma}} \boldsymbol{x}^{\boldsymbol{\gamma}}$$

for some coefficients $d_{\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma}}$. The index $\boldsymbol{\alpha}+\boldsymbol{\beta} \in \Lambda$ owing to the assumption (10), and since Λ is downward closed, then we have that $\pi_{\boldsymbol{\alpha}}(\boldsymbol{x})\pi_{\boldsymbol{\beta}}(\boldsymbol{x}) \in \Pi_\Lambda$. Therefore, the n-point quadrature rule integrates $\pi_{\boldsymbol{\alpha}}(\boldsymbol{x})\pi_{\boldsymbol{\beta}}(\boldsymbol{x})$, and thus

$$\widehat{f}_{\boldsymbol{\beta}} = \sum_{q=1}^{n} f(\boldsymbol{x}_q)\pi_{\boldsymbol{\beta}}(\boldsymbol{x}_q) = \sum_{\boldsymbol{\alpha}\in\Theta} f_{\boldsymbol{\alpha}} \sum_{q=1}^{n} \pi_{\boldsymbol{\alpha}}(\boldsymbol{x})\pi_{\boldsymbol{\beta}}(\boldsymbol{x}) = \sum_{\boldsymbol{\alpha}\in\Theta} f_{\boldsymbol{\alpha}} \left(\pi_{\boldsymbol{\alpha}}, \pi_{\boldsymbol{\beta}}\right) = f_{\boldsymbol{\beta}}.$$

Since $\widehat{f}_{\boldsymbol{\beta}} = f_{\boldsymbol{\beta}}$, then $f_\Theta = f$. ∎

The notion above of reproduction of multivariate polynomials is consistent with univariate Gauss quadrature: In one dimension with an n-point Gauss quadrature rule, we can reproduce polynomials up to degree $n-1$: Take $\Lambda = \{0, \ldots, 2n-1\}$, and choose $\Theta = \{0, \ldots, n-1\}$. The polynomial f_Θ constructed by the procedure (9) matches the function f if $f \in \Pi_\Theta$ since $\Theta + \Theta \subset \Lambda$. The above result codifies this condition in the multivariate case. Note that $\Theta \subset \Lambda$ is not a strict enough condition since the approximate Fourier coefficients defined in (9) will not necessarily be accurate. We also note that the integrand is a product of polynomials; therefore requiring exactness on polynomial products is the correct condition, hence the $\Theta+\Theta \subset \Gamma$ requirement.

Given a multi-index set Λ, there is a smallest possible quadrature size n such that (7) holds. This smallest n is given by the size of the largest Θ satisfying (10).

THEOREM 2.4 ([24]). *Let Λ be a downward-closed index set. The size n of any quadrature rule satisfying* (7) *has lower bound*

$$n \geq \mathcal{L}(\Lambda) \coloneqq \max\left\{|\Theta| \;\middle|\; \Theta + \Theta \subseteq \Lambda\right\}.$$

The number $\mathcal{L}(\Lambda)$ defined above is called the maximal half-set size in [24], and a corresponding $\mathcal{L}(\Lambda)$-point quadrature rule is a minimal rule. In that reference, concrete examples of (i) nonexistence, and (ii) existence but nonuniqueness of minimal multivariate quadrature rules achieving the lower bound above are shown. If $\Lambda = \Lambda_{2n-2}$ in the univariate case, Gaussian quadrature rules are nonunique. Our numerical algorithm essentially seeks to find minimal rules, but we can rarely find such quadrature rules. However, our generated quadrature rule sizes are only modestly larger than the optimal $\mathcal{L}(\Lambda)$.

2.4. Quadrature stability. Gaussian quadrature rules defined by Theorem 2.1 can be computed via linear algebra, but multivariate quadrature rules defined by (7) have no known analogous computational simplification. In order to solve this nonlinear system of equations we utilize Newton's method. We therefore expect that (7) is not exactly satisfied by the computed solution, or it is satisfied to within some tolerance.

Fixing a downward-closed index set Λ with size $M = |\Lambda|$, consider the matrix $\boldsymbol{X} \in \mathbb{R}^{d\times n}$ whose n columns are the samples $\boldsymbol{x}_j$, and let $\boldsymbol{w} \in \mathbb{R}^n$ be a vector containing the n weights. Let $\boldsymbol{V}(\boldsymbol{X}) \in \mathbb{R}^{n\times M}$ denote the Vandermonde-like matrix with entries

$$(V)_{k,j} = \pi_{\boldsymbol{\alpha}(k)}\left(\boldsymbol{x}_j\right), \qquad j = 1, \ldots, n,\ k = 1, \ldots, M, \tag{11}$$

where we have introduced an ordering $\boldsymbol{\alpha}(1), \ldots \boldsymbol{\alpha}(m)$ on the elements of Λ. We assume $\boldsymbol{\alpha}(1) = \mathbf{0}$, but the remaining ordering of elements is irrelevant. The system (7) can then be written as

$$\boldsymbol{V}\left(\boldsymbol{X}\right)\boldsymbol{w} = \boldsymbol{e}_1/\pi_{\mathbf{0}},$$

where $\boldsymbol{e}_1 = (1, 0, 0, \ldots, 0)^T \in \mathbb{R}^M$ is a cardinal unit vector. Instead of achieving the equality above, our computational solver computes an approximate solution $(\boldsymbol{X}, \boldsymbol{w})$ to the above system, satisfying

$$\left\|\boldsymbol{V}\left(\boldsymbol{X}\right)\boldsymbol{w} - \boldsymbol{e}_1/\pi_{\mathbf{0}}\right\|_2 = \epsilon \geq 0. \tag{12}$$

Our next result quantifies the effect of the residual ϵ on the accuracy of the designed quadrature rule. To prove this result, we require the additional assumption that the quadrature weights are positive, which is enforced in our computations.

PROPOSITION 2.5. *Let $\omega(\boldsymbol{x})$ be a probability density function on Γ, and let Λ be any multi-index set containing $\mathbf{0}$ (i.e., Π_Λ contains constant functions). Assume that $(\boldsymbol{X}, \boldsymbol{w})$ satisfies (12) with some $\epsilon \geq 0$, and assume the weights are all positive. Then for any $f \in L^2_\omega(\Gamma)$,*

$$\left| \int f(\boldsymbol{x})\omega(\boldsymbol{x})\mathrm{d}\boldsymbol{x} - \sum_{q=1}^{n} w_q f(\boldsymbol{x}_q) \right| \leq \epsilon \left\| f \right\| + \max_{j=1,\ldots,n} \left| f(\boldsymbol{x}_j) - p(\boldsymbol{x}_j) \right|, \tag{13}$$

where $p \in \Pi_\Lambda$ is the $L^2_\omega(\Gamma)$-orthogonal projection of f onto Π_Λ.

This result does apply to all our computed designed quadrature rules since we enforce positivity of the weights. It is not applicable to other polynomial-based rules where weights can be negative, such as sparse grids.

Proof. For an arbitrary $p \in \Pi_\Lambda$, the following holds:

$$p(\boldsymbol{x}) = \sum_{\boldsymbol{\alpha} \in \mathcal{I}} p_{\boldsymbol{\alpha}} \pi_{\boldsymbol{\alpha}}(\boldsymbol{x}), \qquad\qquad p_{\boldsymbol{\alpha}} = (p, \pi_{\boldsymbol{\alpha}}), \tag{14}$$

and thus $\|p\|^2 = (p,p) = \sum_{\boldsymbol{\alpha}\in\Lambda} p_{\boldsymbol{\alpha}}^2$. We have

$$\begin{aligned} \left| \int_\Gamma f(\boldsymbol{x})\omega(\boldsymbol{x})\mathrm{d}\boldsymbol{x} - \sum_{q=1}^{n} w_q f\left(\boldsymbol{x}_q\right) \right| \leq & \underbrace{\left| \int_\Gamma (f(\boldsymbol{x}) - p(\boldsymbol{x}))\omega(\boldsymbol{x})\mathrm{d}\boldsymbol{x} \right|}_{(a)} + \underbrace{\left| \sum_{q=1}^{n} \left(p(\boldsymbol{x}_q) - f(\boldsymbol{x}_q)\right) w_q \right|}_{(b)} \\ & + \underbrace{\left| \int_\Gamma p(\boldsymbol{x})\omega(\boldsymbol{x})\mathrm{d}\boldsymbol{x} - \sum_{q=1}^{n} w_q p(\boldsymbol{x}_q) \right|}_{(c)}. \end{aligned} \tag{15}$$

We now choose p as the $L^2_\omega(\Gamma)$-orthogonal projection of f into Π_Λ:

$$p = \operatorname*{argmin}_{q\in\Pi_\Lambda} \|f - q\| \implies \int_\Gamma \left[f(\boldsymbol{x}) - p(\boldsymbol{x})\right] \phi(\boldsymbol{x})\omega(\boldsymbol{x})\mathrm{d}\boldsymbol{x} = 0 \quad \forall\, \phi \in \Pi_\Lambda. \tag{16}$$

Since $\mathbf{0} \in \Lambda$, the above holds in particular for $\phi(\boldsymbol{x}) \equiv 1$ so that

$$\text{(a)} = \left| \int_\Gamma \left(f(\boldsymbol{x}) - p(\boldsymbol{x})\right) \omega(\boldsymbol{x})\mathrm{d}\boldsymbol{x} \right| = 0.$$

Term (b) can be bounded as

$$\text{(b)} \leq \sum_{q=1}^{n} |w_q| \left| p(\boldsymbol{x}_q) - f(\boldsymbol{x}_q) \right| \leq \max_{q=1,\ldots,n} \left| p(\boldsymbol{x}_q) - f(\boldsymbol{x}_q) \right|,$$

where the last inequality uses the fact that $\sum_{q=1}^{N} |w_q| = \sum_{q=1}^{N} w_q = \int_\Gamma \omega(\boldsymbol{x})\mathrm{d}\boldsymbol{x} = 1$ since the weights are positive and ω is a probability density. Finally, term (c) can be bounded as follows: Since $p \in \Pi_\Lambda$ then by (14),

$$\sum_{q=1}^{n} w_q p(\boldsymbol{x}_q) = \sum_{q=1}^{n} \sum_{\boldsymbol{\alpha}\in\Lambda} w_q p_{\boldsymbol{\alpha}} \pi_{\boldsymbol{\alpha}}(\boldsymbol{x}_q) = \sum_{\boldsymbol{\alpha}\in\Lambda} p_{\boldsymbol{\alpha}} \left(\sum_{q=1}^{n} w_q \pi_{\boldsymbol{\alpha}}(\boldsymbol{x}_q) \right).$$

The term in parenthesis on the right-hand side is an entry in the vector $\boldsymbol{V}(\boldsymbol{X})\boldsymbol{w}$ from the relation (12); note also that $\widehat{\pi}_{\boldsymbol{\alpha}}$ (cf. (9)) equals an entry in the vector $\boldsymbol{b}$. Therefore, combining the above equation and using the Cauchy–Schwarz inequality yields

$$\begin{aligned}(\mathrm{c}) &= \left| \int_\Gamma p(\boldsymbol{x})\omega(\boldsymbol{x})\mathrm{d}\boldsymbol{x} - \sum_{q=1}^{n} w_q p(\boldsymbol{x}_q) \right| = \left| \sum_{\boldsymbol{\alpha}\in\Lambda} p_{\boldsymbol{\alpha}} \left(\int_\Gamma \pi_{\boldsymbol{\alpha}}(\boldsymbol{x})\omega(\boldsymbol{x})\mathrm{d}\boldsymbol{x} - \sum_{q=1}^{n} w_q \pi_{\boldsymbol{\alpha}}(\boldsymbol{x}_q) \right) \right| \\ &= \left| \sum_{\boldsymbol{\alpha}\in\Lambda} p_{\boldsymbol{\alpha}} \left(\delta_{\boldsymbol{\alpha},\mathbf{0}}/\pi_{\mathbf{0}} - \sum_{q=1}^{n} w_q \pi_{\boldsymbol{\alpha}}(\boldsymbol{x}_q) \right) \right| \leq \sqrt{\sum_{\boldsymbol{\alpha}\in\Lambda} p_{\boldsymbol{\alpha}}^2}\, \|\boldsymbol{V}(\boldsymbol{X})\boldsymbol{w} - \boldsymbol{e}_1/\pi_{\mathbf{0}}\| \leq \epsilon\, \|p\| \leq \epsilon \|f\|,\end{aligned}$$

where the final inequality is Bessel's inequality, which holds since we have chosen p as in (16). Combining our estimates for terms (a), (b), and (c) in (15) completes the proof. ∎

Relative to the pointwise error committed by best $L^2_\omega(\Gamma)$ approximations, the estimate provided by Proposition 2.5 bounds the quadrature error in terms of the quantity ϵ, which is explicitly computable given a quadrature rule.

2.5. A popular alternative: Sparse grids. A (Smolyak) sparse grid is a structured point configuration in multiple dimensions, formed from unions of tensorized univariate rules. Quadrature weights often accompany points in a sparse grid. We briefly describe sparse grids for polynomial integration in this section; they will be used for comparison in our numerical results section.

Consider a tensorial Γ as in section 2.3, and for simplicity assume that the univariate domains $\Gamma_j = \Gamma_1$ are the same and that the univariate weights $\omega_j = \omega_1$ are the same. Let $\mathbb{X}_i$ denote a univariate quadrature rule (nodes and weights) of "level" $i \geq 1$, and define $\mathbb{X}_0 = \emptyset$. The number of points n_i in the quadrature rule $\mathbb{X}_i$ is increasing with i, but can be freely chosen. For multi-index $\boldsymbol{i} \in \mathbb{N}^d$, a d-variate tensorial rule and its corresponding weights are

$$\mathbb{A}_{d,\boldsymbol{i}} = \mathbb{X}_{i_1} \otimes \cdots \otimes \mathbb{X}_{i_d}, \quad w^{(\boldsymbol{q})} = \prod_{r=1}^{d} w_{i_r}^{(q_r)}. \tag{17}$$

The univariate difference operator between sequential levels is written as

$$\Delta_i = \mathbb{X}_i - \mathbb{X}_{i-1}, \qquad i \geq 1, \tag{18}$$

and for any $k \in \mathbb{N}$, this approximation difference can be used to construct a d-variate, level-k-accurate sparse grid operator [7, 38],

$$\mathbb{A}_{d,k} = \sum_{r=0}^{k-1} \sum_{\substack{\boldsymbol{i}\in\mathbb{N}^d \\ |\boldsymbol{i}|=d+r}} \Delta_{i_1} \otimes \cdots \otimes \Delta_{i_d} = \sum_{r=k-d}^{k-1} (-1)^{k-1-r} \binom{d-1}{k-1-r} \sum_{\substack{\boldsymbol{i}\in\mathbb{N}^d \\ |\boldsymbol{i}|=d+r}} \mathbb{X}_{i_1} \otimes \cdots \otimes \mathbb{X}_{i_d}, \tag{19}$$

where the latter equality is shown in [52]. If the univariate quadrature rule $\mathbb{X}_i$ exactly integrates univariate polynomials of order $2i-1$ or less, then the Smolyak rule $\mathbb{A}_{d,k}$ is exact for d-variate polynomials of total order $2k-1$ [23]. One is tempted to use Gauss quadrature rules for the $\mathbb{X}_i$ to obtain optimal efficiency, but since the differences Δ_i appear in the Smolyak construction, utilizing nested univariate rules instead can generate sparse grids with many fewer nodes than nonnested constructions. One can

use, for example, nested Clenshaw–Curtis rules [55], the nested Gauss–Patterson or Gauss–Kronrod rules [16, 26, 35], or Leja sequences [31].

Sparse grids have been used with great success in many modern applications, and thus are good candidates for comparison against our approach of designed quadrature. However, sparse grids that integrate polynomials in a certain multi-index set use far more points than the minimum number prescribed by Theorem 2.4 (see Figure 3 for an empirical comparison) and frequently produce quadrature rules with negative weights. Our results in section 4 show that designed quadrature uses many fewer points than sparse grids for a given accuracy level, and guarantees positive quadrature weights.

3. Computational framework. Our procedure aims to compute nodes $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \in \Gamma^n$ and positive weights $\boldsymbol{w} \in (0, \infty)^n$ that enforce equality in (7). A direct formulation of (7) is

$$\begin{aligned} \boldsymbol{R}(\boldsymbol{d}) = \boldsymbol{V}(\boldsymbol{X})\boldsymbol{w} - \boldsymbol{e}_1/\pi_{\boldsymbol{0}} = \boldsymbol{0}, & \\ \boldsymbol{x}_j \in \Gamma, \quad j = 1, \ldots, n, & \\ w_j > \boldsymbol{0}, \quad j = 1, \ldots n, & \end{aligned} \tag{20}$$

where $\boldsymbol{d} = (\boldsymbol{X}, \boldsymbol{w})$ are the decision variables. Instead of directly solving this constrained root finding problem, we introduce a closely related constrained optimization problem:

$$\begin{aligned} \min_{\boldsymbol{X}, \boldsymbol{w}} \quad & ||\boldsymbol{R}||_2 \\ \text{subject to} \quad & \boldsymbol{x}_j \in \Gamma, \quad j = 1, \ldots, n, \\ & w_j > \boldsymbol{0}, \quad j = 1, \ldots, n. \end{aligned} \tag{21}$$

Clearly a solution to (20) also solves (21), but the reverse is not necessarily true. We compute solutions to (21), and when these solutions exhibit large nonzero values of $\|\boldsymbol{R}\|$, we increase the quadrature rule size n and repeat. Using this strategy, we empirically find that for a specified ϵ we can satisfy $\|\boldsymbol{R}\| \leq \epsilon$ in all situations we have tried. Thus, our approach solves a relaxed version of (20) via repeated applications of (21). Our computational approach to solve (21) requires four major ingredients, each of which is described in the subsequent sections:

- Section 3.1 – Penalization: objective augmentation, transforming constrained root finding into unconstrained minimization problem.
- Section 3.2 – Iteration: unconstrained minimization via the Gauss–Newton algorithm.
- Section 3.3 – Regularization: numerical regularization to address ill-conditioned Gauss–Newton update steps.
- Section 3.4 – Initialization: specification of an initial guess.

We highlight above that regularization is required for our optimization. The objective $\boldsymbol{R}$ in (21) is highly ill-conditioned as a function of the decision variables. Without regularization, the update steps specified by the Gauss–Newton algorithm generally do not result in convergence. However, with the regularization, we have found that our optimization results in steps with a decreasing residual. These observations can be corroborated by the numerical results in section 4, and in particular Table 2, which lists CPU time and iterations required for computing 4-dimensional rules.

Since our algorithm only minimizes the norm of $\boldsymbol{R}$, the quadrature rule we compute is not guaranteed to integrate any polynomials exactly, only up to some tolerance parameter $\epsilon \geq \|\boldsymbol{R}\|$. This is the utility of Proposition 2.5: if our optimization algorithm terminates with a particular value of ϵ, we have a quantitative understanding

of how ϵ affects the quality of the quadrature rule relative to best L^2-approximating polynomials.

Since we produce a quadrature rule that is only ϵ-exact, there may be many quadrature rules that achieve this tolerance. In particular, our algorithm is not guaranteed to produce optimal quadrature rules, but in comparison with some other tabulated rules from [40, 43, 53, 54], we find that our nodal counts are no greater than in those references. There is one lone exception for integrating degree-8 polynomials in three dimensions, where we find a rule with one point greater than reported in [53]. Details are in section 4.2 and in Table 1.

Finally, our algorithm is subject to the same limitations as many other minimization algorithms: it may only find a local minimum of the objective, and not a global minimum.

3.1. Penalization. Penalty methods are techniques for solving constrained optimization problems such as (21). Penalty methods augment the objective with a high cost for violated constraints, and subsequently solve an unconstrained optimization problem on the augmented objective.

We use a popular penalty function, the nonnegative and smooth quadratic function. For example, in $d = 1$ dimensions on $\Gamma = [-1, 1]$ with an n-point quadrature rule, the constraints and corresponding penalties P_j, $j = 1, \ldots, (d+1)n = 2n$, as a function of the $2n$ decision variables $\boldsymbol{d} = (\boldsymbol{X}, \boldsymbol{w})$ can be expressed as

$$\begin{aligned} -1 \le x_j \le 1 \quad &\Longrightarrow \quad P_j(\boldsymbol{d}) = \left(\max[0, x_j - 1, -1 - x_j]\right)^2, \\ w_j \ge 0 \quad &\Longrightarrow \quad P_{n+j}(\boldsymbol{d}) = \left(\max[0, -w_j]\right)^2 \end{aligned}$$

for $j = 1, \ldots, n$. The total penalty associated with the constraints is then

$$P^2(\boldsymbol{d}) = \sum_{j=1}^{(d+1)n} P_j^2(\boldsymbol{d}).$$

A penalty function approach to solving the constrained problem (21) uses a sequence of unconstrained problems indexed by $k \in \mathbb{N}$ having objective functions

$$g(c_k, \boldsymbol{d}) \coloneqq \left\| \widetilde{\boldsymbol{R}}_k \right\|_2^2 = \|\boldsymbol{R}\|_2^2 + c_k^2 P^2(\boldsymbol{d}), \tag{22}$$

where we have defined the vector

$$\widetilde{\boldsymbol{R}}_k = \begin{bmatrix} \boldsymbol{R} \\ c_k P_1 \\ c_k P_2 \\ \vdots \\ c_k P_{(d+1)n} \end{bmatrix}.$$

The positive constants c_k are monotonically increasing with k, i.e., $c_{k+1} > c_k$. Each unconstrained optimization yields an updated solution point $\boldsymbol{d}^k$, and as $c_k \to \infty$ the solution point of the unconstrained problem will converge to the solution of the constrained problem. The following lemma, adopted from [28], is used to show convergence of the penalty method.

LEMMA 3.1. *Let $\boldsymbol{d}^k$ be the minimizer for $g(c_k, \cdot)$ and $c_{k+1} > c_k$. Then*

$$g(c_k, \boldsymbol{d}^k) \le g(c_{k+1}, \boldsymbol{d}^{k+1}), \qquad P(\boldsymbol{d}^k) \ge P(\boldsymbol{d}^{k+1}), \qquad ||\boldsymbol{R}(\boldsymbol{d}^k)|| \le ||\boldsymbol{R}(\boldsymbol{d}^{k+1})||.$$

Furthermore, let $\boldsymbol{d}^*$ *be a solution to problem* (21)*. Then, for each* k*,*

$$||\boldsymbol{R}(\boldsymbol{d}^k)|| \leq g(c_k, \boldsymbol{d}^k) \leq ||\boldsymbol{R}(\boldsymbol{d}^*)||.$$

The above lemma denotes that the sequence of $g(c_k, \boldsymbol{d}^k)$ is nondecreasing and bounded above by the optimal objective value of the constrained optimization problem. The following theorem establishes the global convergence of the penalty method. More precisely it verifies that any limit point of the sequence is a solution to (21).

THEOREM 3.2 ([28]). *Let* $\{\boldsymbol{d}^k\}$*,* $k \in \mathbb{N}$*, be a sequence of minimizers of* (22)*. Then any limit point of the sequence is a solution to problem* (21)*, i.e.,* $\lim_{k\in\mathbb{N}} P(\boldsymbol{d}^k) = 0$ *and* $\lim_{k\in\mathbb{N}} ||R(\boldsymbol{d}^k)|| \leq ||R(\boldsymbol{d}^*)||$.

The above theorem shows both that a limit point denoted by $\bar{\boldsymbol{d}}$ is a feasible solution since $P(\bar{\boldsymbol{d}}) = 0$, and that it is optimal since $||\boldsymbol{R}(\bar{\boldsymbol{d}})||_2^2 \leq ||\boldsymbol{R}(\boldsymbol{d}^*)||_2^2$.

We can now formulate an unconstrained minimization problem with a sequence of increasing c_k on the objectives g in (22) for the decision variables $\boldsymbol{d} = (\boldsymbol{X}, \boldsymbol{w})$,

$$\min_{\boldsymbol{d}} g(c_k, \boldsymbol{d}), \tag{23}$$

which replaces the constrained root finding problem (20).

It remains for us to specify how the constants c_k are chosen: if $\boldsymbol{d}$ is the current iterate for the decision variables, we use the formula

$$c_k = \max\left\{A, \frac{1}{||\boldsymbol{R}(\boldsymbol{d})||_2}\right\},$$

where A is a tunable parameter that is meant to be large. We use $A = 10^3$ in our simulations. Also note that we never have $c_k = \infty$ so that our iterations cannot exactly constrain the computed solution to lie in the feasible set. To address this in practice we reformulate constraints to have nonzero penalty within a small radius inside the feasible set. For example, instead of enforcing $w_j > 0$, we enforce $w_j > 10^{-6}$.

Note that one may also consider barrier/interior point methods to enforce constraints; however, in our algorithm we find that penalty methods are more suitable in transforming the constrained root finding problem to an unconstrained minimization problem.

3.2. The Gauss–Newton algorithm. Having transformed the constrained problem (21) into a sequence of unconstrained problems (23), we can now use standard unconstrained optimization tools.

Two popular approaches for unconstrained optimization are gradient descent and Newton's method. Both approaches in the context of our minimization require the Jacobian of the objective function with respect to the decision variables. We define

$$\widetilde{\boldsymbol{J}}_k = \frac{\partial \widetilde{\boldsymbol{R}}_k}{\partial \boldsymbol{d}} = \begin{bmatrix} \boldsymbol{J} \\ c_k \partial P_1/\partial \boldsymbol{d} \\ c_k \partial P_2/\partial \boldsymbol{d} \\ \vdots \\ c_k \partial P_{(d+1)n}/\partial \boldsymbol{d} \end{bmatrix}, \qquad \boldsymbol{J}(\boldsymbol{d}) \coloneqq \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{d}} \in \mathbb{R}^{M\times(d+1)n}, \tag{24}$$

where $\frac{\partial P_j}{\partial \boldsymbol{d}} \in \mathbb{R}^{1\times(d+1)n}$ is the Jacobian of P_j with respect to the decision variables. With use of our quadratic penalty function, these penalty Jacobians are Lipschitz

continuous in the decision variables, and easily evaluated since they are quadratic functions. The matrix $\boldsymbol{J}$ has entries

$$(J)_{m,(i-1)d+j} = \frac{\partial \pi_{\boldsymbol{\alpha}(m)}(\boldsymbol{x}_i)}{\partial x_i^{(j)}} w_i, \qquad (J)_{m,nd+i} = \pi_{\boldsymbol{\alpha}(m)}(\boldsymbol{x}_j), \tag{25}$$

for $m = 1, \ldots, M$, $i = 1, \ldots, n$, and $j = 1, \ldots, d$. Above, we define $\pi_{\boldsymbol{\alpha}(m)}$ as in (11). Computing entries of the Jacobian matrix $\boldsymbol{J}$ is straightforward: Assuming the basis $\boldsymbol{\pi}_{\boldsymbol{\alpha}}$ is of tensor-product form (see section 2.3), we then need only compute derivatives of univariate polynomials. A manipulation of the three-term recurrence relation (3) yields the recurrence

$$\sqrt{b_{m+1}} p'_{m+1}(x) = (x - a_m) p'_m(x) - \sqrt{b}_m p'_{m-1}(x) + p_m(x).$$

The partial derivatives in $\boldsymbol{J}$ may be evaluated using the relation above along with (6).

We index iterations with k, which is the same k as that defining the sequence of unconstrained problems (23). Thus, our choice of c_k changes at each iteration. Gradient descent proceeds via iteration of the form

$$\boldsymbol{d}^{k+1} = \boldsymbol{d}^k - \alpha \frac{\partial \|\widetilde{\boldsymbol{R}}_k\|_2}{\partial \boldsymbol{d}}, \qquad \frac{\partial \|\widetilde{\boldsymbol{R}}_k\|_2}{\partial \boldsymbol{d}} = \frac{\widetilde{\boldsymbol{J}}_k^T \widetilde{\boldsymbol{R}}}{\|\widetilde{\boldsymbol{R}}_k\|_2},$$

with α a customizable step length that is frequently optimized via, e.g., a line-search algorithm. In contrast, a variant of Newton's root finding method applied to rectangular systems is the Gauss–Newton method [39], having update iteration

$$\boldsymbol{d}^{k+1} = \boldsymbol{d}^k - \Delta \boldsymbol{d}, \qquad \Delta \boldsymbol{d} = \left(\widetilde{\boldsymbol{J}}_k^T \widetilde{\boldsymbol{J}}_k \right)^{-1} \widetilde{\boldsymbol{J}}_k^T \widetilde{\boldsymbol{R}}_k, \tag{26}$$

where both $\widetilde{\boldsymbol{J}}_k$ and $\widetilde{\boldsymbol{R}}_k$ are evaluated at $\boldsymbol{d}^k$. The iteration above reduces to the standard Newton's method when the system is square, i.e., $M = n(d+1)$. Newton's method converges quadratically to a local solution for a sufficiently close initial guess $\boldsymbol{d}^0$, versus the gradient descent, which has linear convergence [4]. We find that Gauss–Newton iterations are robust for our problem.

Assuming an initial guess $\boldsymbol{d}^0$ is given, we can repeatedly apply the Gauss–Newton iteration (26) until a stopping criterion is met. We terminate our iterations when the residual norm falls below a user-defined threshold ϵ, i.e., $||\widetilde{\boldsymbol{R}}||_2 < \epsilon$.

A useful quantity to monitor during the iteration process is the magnitude of the Newton decrement, which often reflects quantitative proximity to the optimal point [6]. In its original form, the Newton decrement is the norm of the Newton step in the quadratic norm defined by the Hessian. That is, for optimizing $f(\boldsymbol{x})$, the Newton decrement norm is $||\Delta \boldsymbol{d}||_{\nabla^2 f(\boldsymbol{x})} = (\Delta \boldsymbol{d}^T \nabla^2 f(\boldsymbol{x}) \Delta \boldsymbol{d})^{1/2}$, where $\nabla^2 f$ is the Hessian of f. In our minimization procedure with nonsquare systems we use

$$\eta = \left(\Delta \boldsymbol{d}^T (\widetilde{\boldsymbol{J}}_k^T \widetilde{\boldsymbol{R}}_k) \right)^{1/2} \tag{27}$$

as a surrogate for a Hessian-based Newton decrement which decreases as $\boldsymbol{d} \to \boldsymbol{d}^*$.

Finally we note that, for a given quadrature rule size n, we cannot guarantee that a solution to (20) exists. In this case our Gauss–Newton iterations will exhibit residual norms stagnating at some positive value, while the Newton decrement is almost zero. When this occurs, we reinitialize the decision variables and enrich the current set of decision variables with additional nodes and weights and continue the optimization procedure. This procedure of gradually increasing the number of nodes and weights is described more in section 3.4.

3.3. Regularization. The critical part of our minimization scheme is the evaluation of Newton step (26). For our rectangular system, this is the least-squares solution $\Delta \boldsymbol{d}$ to the linear system

$$\widetilde{\boldsymbol{J}} \Delta \boldsymbol{d} = \widetilde{\boldsymbol{R}},$$

where $\widetilde{\boldsymbol{J}} = \widetilde{\boldsymbol{J}}_k\left(\boldsymbol{d}^k\right)$ and $\widetilde{\boldsymbol{R}} = \widetilde{\boldsymbol{R}}_k\left(\boldsymbol{d}^k\right)$; in this section we omit explicit notational dependence on the iteration index k. The matrix $\widetilde{\boldsymbol{J}}$ is frequently ill-conditioned, which hinders a direct solve of the above least-squares problem. To address this we can consider a generic regularization of the above equality:

$$\underset{\Delta \boldsymbol{d}}{\text{minimize}} \quad ||\widetilde{\boldsymbol{J}} \Delta \boldsymbol{d} - \widetilde{\boldsymbol{R}}||_p \quad \text{subject to} \quad ||\Delta \boldsymbol{d}||_q < \tau, \tag{28}$$

where p, q, and τ are free parameters. The trade-off between the objective norm and the solution norm is characterized as a Pareto curve and shown to be convex in [50, 51] for generic norms $1 \leq (p, q) \leq \infty$. Exploiting this Pareto curve, the authors in [50, 51] devise an efficient algorithm and implementation [49] for computing the regularized solution when $p = 2$, $q = 1$. These values correspond to the LASSO problem [47], which promotes solution sparsity and subset selection.

Since sparsity is not our explicit goal, we opt for $p = q = 2$. This problem can be solved exactly [19], but at significant expense, and the procedure lacks clear guidance on choosing τ. We thus adopt an alternative approach. A penalized version of the $p = q = 2$ optimization (28) is Tikhonov regularization:

$$\Delta \boldsymbol{d}_\lambda = \text{argmin}\left\{ ||\widetilde{\boldsymbol{J}} \Delta \boldsymbol{d} - \widetilde{\boldsymbol{R}}||_2^2 + \lambda ||\Delta \boldsymbol{d}||_2^2 \right\}, \tag{29}$$

where λ is a regularization parameter that may be chosen by the user. This parameter has significant impact on the quality of the solution with respect to the original least-squares problem. Assuming that we have a definitive value for λ, then the solution to (29) can be obtained via the singular value decomposition (SVD) of $\widetilde{\boldsymbol{J}}$. The SVD of matrix $\widetilde{\boldsymbol{J}}_{\mathrm{N} \times \mathrm{M}}$ (for $\mathrm{N} < \mathrm{M}$) is given by

$$\widetilde{\boldsymbol{J}} = \sum_{i=1}^{\mathrm{N}} \boldsymbol{u}_i \sigma_i \boldsymbol{v}_i^T, \tag{30}$$

where σ_i are singular values (in decreasing order), and $\boldsymbol{u}_i$ and $\boldsymbol{v}_k$ are the corresponding left- and right-singular vectors, respectively. The solution $\Delta \boldsymbol{d}_\lambda$ is then obtained as

$$\Delta \boldsymbol{d}_\lambda = \sum_{i=1}^{\mathrm{N}} \rho_i \frac{\boldsymbol{u}_i^T \widetilde{\boldsymbol{R}}}{\sigma_i} \boldsymbol{v}_i, \tag{31}$$

where ρ_i are Tikhonov filter factors denoted by

$$\rho_i = \frac{\sigma_i^2}{\sigma_i^2 + \lambda^2} \simeq \begin{cases} 1, & \sigma_i \gg \lambda, \\ \sigma_i^2 / \lambda^2, & \sigma_i \ll \lambda. \end{cases} \tag{32}$$

Tikhonov regularization affects (or filters) singular values that are below the threshold λ. Therefore a suitable λ is bounded by the extremal singular values of $\widetilde{\boldsymbol{J}}$. One approach to select λ is via analysis of the "L-curve" of singular values [20, 21]. The

corner of the L-curve can be interpreted as the point with maximum curvature; evaluation or approximation of the curvature with respect to the singular value index can be used to find the index with maximum curvature, and the singular value corresponding to this index prescribes λ.

In practice, we evaluate the curvature of the singular value spectrum via finite differences on $\log(\sigma_i)$ (where the singular values are directly computed) and select the singular value that corresponds to the first spike in the spectrum. The regularization parameter can be updated after several, e.g., 30, Gauss–Newton iterations. However, for small-sized problems, i.e., small dimension d and $|\Lambda|$, a fixed appropriate λ throughout the Gauss–Newton scheme also yields solutions.

Based on our numerical observations, adding a regularization parameter to all of the singular values and computing the regularized Newton step as $\Delta \boldsymbol{d}_\lambda = \sum_{i=1}^{\mathrm{N}} [(\boldsymbol{u}_i^T \widetilde{\boldsymbol{R}})/(\sigma_i + \lambda)]\boldsymbol{v}_i$ enhances the convergence when $\boldsymbol{d}$ is close to the root, i.e., $||\widetilde{\boldsymbol{R}}||$ is small.

3.4. Initialization. The first step of the algorithm requires an initial guess $\boldsymbol{d}^0$ for nodes and weights; a particularly difficult aspect of this is the initial choice of quadrature rule size n. Our algorithm tests several values of quadrature rule sizes n between an upper and lower bound; the determination of these bounds is described below.

With the multi-index set Λ given, Theorem 2.4 provides a lower bound on the value of n, and this lower bound $\mathcal{L}(\Lambda)$ is the optimal size for a quadrature rule. We are unaware of sufficient conditions under which optimal quadrature rules exist. However, optimal-sized quadrature rules have been shown in special cases, e.g., [41], for total degree spaces $\Lambda_{\mathcal{T}_k}$ with $k = 2, 3, 5$. We have found that our algorithm is able to recover these optimal-sized rules in the previously mentioned cases.

We formulate an upper bound on quadrature rule sizes based on a popular competitor: sparse grid constructions. The number of sparse grid points $|\mathbb{A}_{d,k}|$ required to satisfy (7) with $\Lambda = \Lambda_{\mathcal{T}_k}$ can be estimated as $|\mathbb{A}_{d,k}| \approx \frac{(2d)^{k-1}}{(k-1)!}$ [13] for sparse grid constructions with nonnested univariate Gauss quadrature rules. Tabulation of the exact number of points for sparse grids constructed via univariate nested rules from the Hermite and Legendre systems is provided in [22].

Our numerical results show that the number of designed quadrature nodes needed to satisfy (7) is $n = \kappa|\mathbb{A}_{d,k}|$, where $\kappa \in [0.5, 0.9]$ using $|\mathbb{A}_{d,k}|$ from [22]. We have found that an effective approach to choosing the number of points is to perform a backtracking line-search procedure, which initializes $\kappa = 0.9$, solves the optimization problem, and gradually decreases κ until the Gauss–Newton method does not converge to a desirable tolerance. Our strategy for eliminating nodes when κ is decreased is to discard those with the smallest weights.

After the initial pass that generates n nodes and weights achieving $\|\widetilde{\boldsymbol{R}}\| \leq \epsilon$, we attempt to remove nodes with smallest weights as described previously. However, this may cause the optimization to stagnate without achieving the desired tolerance. When this happens, we enrich the nodal set by gradually adding more nodes until we can achieve the tolerance. This process is repeated until the elimination and enrichment procedures result in no change of the quadrature rule size; see Algorithm 1, lines 9–18.

Once an initial number of nodes n is determined ($\kappa = 0.9$), that number of d-variate Monte Carlo samples or Latin hypercube samples are generated as the initial nodes. This is easily done for the domain $\Gamma = [-1, 1]^d$. Weights can be generated uniformly at random $[0, 1]$ with $\sum_i w_i = |\Lambda|$ or set as a fixed value, e.g., $w_i = |\Lambda|/n$. We normalize the weights by $|\Lambda|$ in the numerical procedure to avoid very small weights. To accommodate for this, we can set $1/\pi_{\mathbf{0}} = |\Lambda|$ in (20), and after we obtain

Algorithm 1 Designed quadrature.

1: Initialize nodes and weights $\boldsymbol{d}$ with $n = 0.9|\mathbb{A}_{d,k}|$ and specify the residual tolerance, e.g., $\epsilon = 10^{-8}$.
2: Set $n_0 = 0$.
3: **while** $||\widetilde{\boldsymbol{R}}|| > \epsilon$ **do**
4: Compute $\widetilde{\boldsymbol{R}}$ and $\widetilde{\boldsymbol{J}}$ using (22), (20), (24), and (25).
5: Determine the regularization parameter λ from the SVD of $\widetilde{\boldsymbol{J}}$.
6: Compute the regularized Newton step $\Delta\boldsymbol{d}$ from (31).
7: Update the decision variables $\boldsymbol{d}^{k+1} = \boldsymbol{d}^k - \Delta\boldsymbol{d}$.
8: Compute the residual norm $||\widetilde{\boldsymbol{R}}||_2$ and Newton decrement η from (27).
9: **if** $\eta < \epsilon$ and $||\widetilde{\boldsymbol{R}}||_2 \gg \epsilon$ **then**
10: Increase n, initialize new nodes and weights, and go to line 3.
11: **end if**
12: **end while**
13: **if** $n = n_0$ **then**
14: Return
15: **else**
16: $n_0 \leftarrow n$.
17: Decrease n by eliminating nodes with smallest weights, and go to line 3. (See discussion about κ in section 3.4.)
18: **end if**

a solution, we can renormalize the weights based on the true value of $1/\pi_{\mathbf{0}}$.

On the domain $\Gamma = \mathbb{R}^d$, we are usually concerned with the weight $\omega(\boldsymbol{x}) = \exp(-\|\boldsymbol{x}\|_2^2)$. Monte Carlo samples can be generated as realizations of a standard normal random variable, and we transform Latin hypercube samples on $[0,1]^d$ to $\mathbb{R}^d$ via inverse transform sampling corresponding to a standard normal random variable. (When Λ contains polynomials of very high degree, there are more sophisticated sampling methods that can produce better initial guesses [32].) We initialize the weights by setting $w_i = \exp(-||x_i||_2^2/2)$ and normalizing w_i with respect to $|\Lambda|$ as described above.

Algorithm 1 summarizes sections 3.1–3.4, including all the steps for our designed quadrature method.

4. Numerical examples.

4.1. Illustrative numerical example in $d = 2$. In this example we consider $d = 2$ for a uniform weight on $\Gamma = [0,1]^2$ with an $r = 2$ total degree polynomial space with index set $\Lambda_{\mathcal{T}_2}$. This index set has six indices, corresponding to six constraints in (7). Using $n = 3$ nodes, there are $(d+1)n = 9$ decision variables. Note that exact formulas for the optimal quadrature rule are known in this case [43]. The augmented Jacobian $\widetilde{\boldsymbol{J}}$ in (24) is a 15×9 matrix. We initialize three nodes with a Latin hypercube design on $[0,1]^2$ and use uniform weights. The singular values of the Jacobian matrix are shown in Figure 1 for the initial and final decision variables corresponding to three different choices of the regularization parameter λ. The results suggest that any positive value in $[0.01, 5]$ can be used as a regularization parameter. We use a constant λ throughout the iterations and fix the residual tolerance $\epsilon = 10^{-8}$. The evolution of residual $\|\widetilde{\boldsymbol{R}}\|$, Newton decrement η, and penalty parameter c_k is shown in Figure 1. Smaller λ values appear to yield faster convergence.

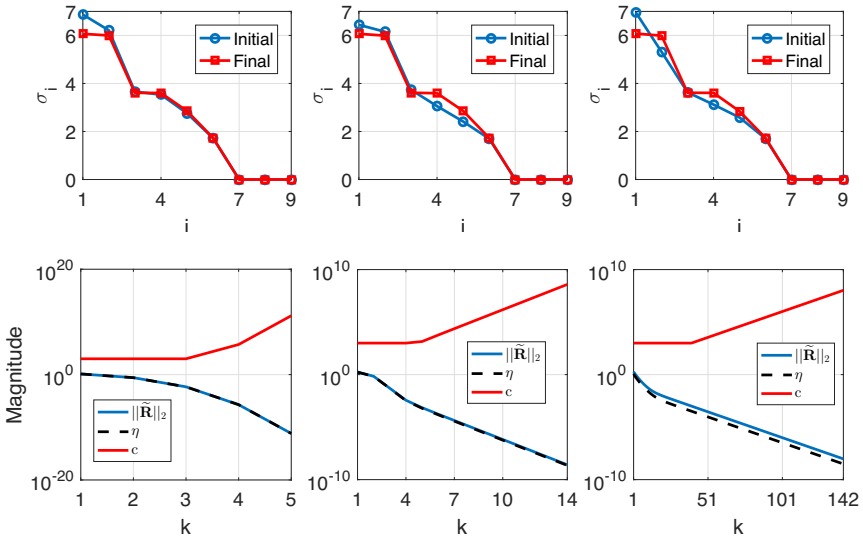

FIG. 1. *Singular values of Jacobian* $\widetilde{\boldsymbol{J}}$ *(cf.* (24)*) (top) and residual norm* $\|\widetilde{\boldsymbol{R}}\|_2$*, Newton decrement* η*, and penalty parameter* c *with respect to iterations* k *(bottom) for regularization parameter* $\lambda = 0.01$ *(left),* $\lambda = 1$ *(middle), and* $\lambda = 5$ *(right).*

To visualize the optimal points for this quadrature, we randomize the initial node positions and compute designed quadrature for 100 initializations. Plots of the ensemble of converged quadrature rules in two and three dimensions are shown in Figure 2. A set of 3 points (initial and final design) in each experiment forms a triangle; i.e., vertices of each triangle are the quadrature points where each triangle is visualized for differentiation. The cumulative time for 100 designs took ~ 6 sec with MATLAB on a single core personal desktop, and each design takes ~ 15 iterations with $\lambda = 1$.

4.2. Comparison with sparse grid quadrature. In this example we consider the number of nodes required to achieve exact polynomial accuracy on total degree spaces $\Lambda_{\mathcal{T}_r}$ of various orders and dimensions. Our goal is to compare designed quadrature against sparse grids. The number of nodes required for exact integration on a sparse grid is from [22]. Our tests fix dimension $d = 3$ and sweep values of the order r, and fix $r = 5$ and sweep values of dimension d. We present the nodal counts in Table 1 and in Figure 3. Table 1 shows that designed quadrature consistently results in fewer nodes than sparse grids for moderate values of r and d. We again emphasize that the weights for designed quadrature are all positive, unlike sparse grid quadrature.

Figure 3 compares various node counts: The number of nodes in the product rule is simply n^d, where n is the number of univariate Gauss quadrature nodes and the "lower bound" is the value $\mathcal{L}(\Lambda)$ determined from Theorem 2.4. Using Theorem 2.1 in [24], we can explicitly compute this as

$$\mathcal{L}\left(\Lambda_{\mathcal{T}_r}\right) = \left|\Lambda_{\mathcal{T}_{\lfloor r/2 \rfloor}}\right| = \begin{pmatrix} d + \lfloor r/2 \rfloor \\ d \end{pmatrix}. \tag{33}$$

Independently, we computed designed quadratures for $r = 2$ and $r = 3$ to confirm that the number of nodes for different dimensions d coincides with $d + 1$ and $2d$, respectively, as determined in [41] (not shown). Also, for $r = 5$ and $d = 3, 5$ we find the same number of nodes as those given by [40] with positive weights.

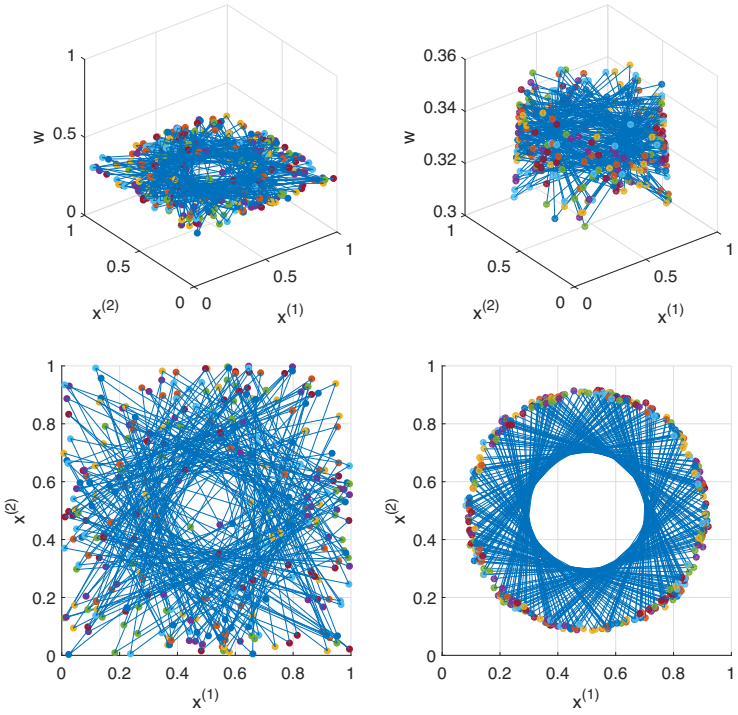

FIG. 2. *Ensemble of 3-point quadrature rules on $\Gamma = [0,1]^2$ found via designed quadrature ($d = r = 2$). Each 3-point nodal configuration has nodes connected with blue lines, forming a triangle. Left: initial guesses provided to the algorithm. Right: converged designed quadrature rules. Bottom: nodal configurations on Γ. Top: weight values plotted as z-coordinates. (Color available online.)*

TABLE 1

Number of node-n sparse grids and designed quadratures on total degree spaces $\Lambda_{\mathcal{T}_r}$ on $\Gamma = [0,1]^d$. Top: fixed $d = 3$ for various r. Bottom: fixed $r = 5$ for various d. The results in the top half of this table can be compared with Table 4 in [53]. Our quadrature rules have smaller or equal size compared with the results in [53], with the exception of $r = 8$, where we report a 43-point rule instead of a 42-point rule in [53].

$d = 3, r$	1	2	3	4	5	6	7	8	9	10	11
Sparse grid quadrature (nested)	1	-	7	-	19	-	39	-	87	-	135
Designed quadrature	1	4	6	10	13	22	26	43	51	74	84
$r = 5, d$	1	2	3	4	5	6	7	8	9	10	
Sparse grid quadrature (nested)	3	9	19	33	51	73	99	129	163	201	
Designed quadrature	3	7	13	21	32	44	63	88	114	148	

In Table 2 we show the performance of the scheme with respect to the number of nodes and iterations, CPU time (measured with tic-toc on MATLAB), and the achieved residual norm. To that end we consider $d = 4$ for different orders r and set the tolerance to $\epsilon = 10^{-12}$ in this example. It should be noted that these quantitative metrics can vary depending on the random initialization and regularization parameters throughout the algorithm; however, they provide a useful holistic measure for the method's performance.

To illustrate how the regularization parameter λ is chosen, we show the singular

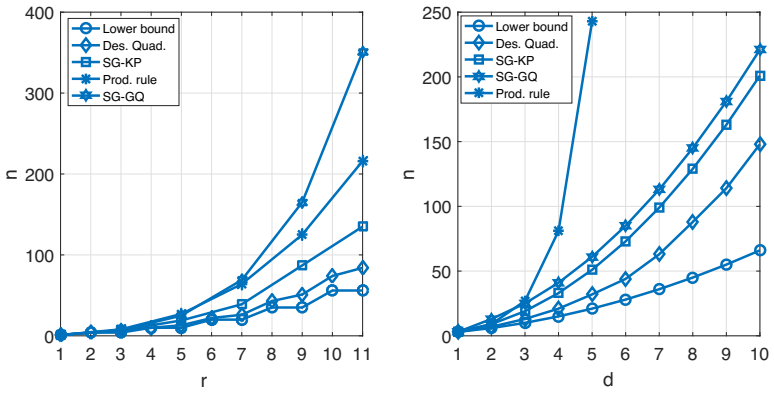

FIG. 3. *Number of nodes for fixed $d = 3$ (left) and fixed $r = 5$ (right) for total order index set $\Lambda_{\mathcal{T}_r}$. The lower bound is given in* (33)*, and "SG-KP" and "SG-GQ" are sparse grid constructions using nested Kronrod–Patterson and nonnested Gauss quadrature rules, respectively.*

TABLE 2

Performance of the scheme with respect to number of nodes and iterations, CPU time, and residual norm for $d = 4$ and various orders r with total order index set.

$d = 4, r$	1	2	3	4	5	6	7	8	9	10
Half-set size $\lvert\Lambda_{\mathcal{T}_{\lfloor r/2 \rfloor}}\rvert$	1	5	5	15	15	35	35	70	70	126
Number of nodes	1	5	8	16	21	43	55	103	138	207
Number of iterations	10	9	11	56	179	146	298	153	461	197
CPU time (sec)	0.09	0.21	0.25	0.91	4.68	8.45	30.66	33.06	183.67	160.44
Residual norm $\lvert\lvert\tilde{\boldsymbol{R}}\rvert\rvert_2$	1e-14	2e-14	4e-13	8e-13	4e-13	9e-13	9e-13	3e-13	9e-13	9e-13

values and regularization parameter choice for the case $r = 5$, $d = 7$. Figure 4 shows the regularization parameter selection for an iteration in the middle of the procedure. The regularized parameter is selected as $\lambda = 10$ by investigating the spectrum of singular values and its L-curve.

In practice, one could fix λ as a function of $\|\boldsymbol{R}\|_2$ (or $\|\widetilde{\boldsymbol{R}}\|_2$). In Figure 3, we have $\lambda = 10$ with $\|\boldsymbol{R}\|_2 = 40$. Then, for example, one could take $\lambda = 50$ for $200 \leq ||\boldsymbol{R}||_2 \leq 500$ and $\lambda = 10$ for $20 \leq ||\boldsymbol{R}||_2 \leq 200$. Such an a priori tabulation could be fixed for a variety of (d, r) values.

4.3. Interpolation with designed quadrature. Designed quadrature rules can be used to construct polynomial interpolants. Suppose we have a designed quadrature rule $(\boldsymbol{X}, \boldsymbol{w})$ of size n that matches moments for indices on Λ (up to the tolerance ϵ), and assume $n = |\Lambda|$.[1] For continuous function f, let $\mathcal{I}(f)$ denote the unique interpolant of f from Π_Λ at the locations $\boldsymbol{X}$. Lebesgue's lemma states

$$||f - \mathcal{I}(f)||_\infty \leq (L+1) \inf_{p \in \Pi_\Lambda} ||f - p||_\infty, \qquad L = \sup_{\|h\|_\infty = 1} \|\mathcal{I}(h)\|_\infty ,$$

where $\|\cdot\|_\infty$ is the maximum norm on Γ, and the supremum is taken over all functions h continuous on Γ. The constant L is the Lebesgue constant; small values indicate that interpolants are comparable to the best approximation measured in the maximum norm [27]. The Lebesgue constant can be computed explicitly: The interpolant $\mathcal{I}(f)$

[1]Designed quadrature rules achieve $n < |\Lambda|$, but in this section we will enforce $n = |\Lambda|$ for the purposes of forming an interpolant.

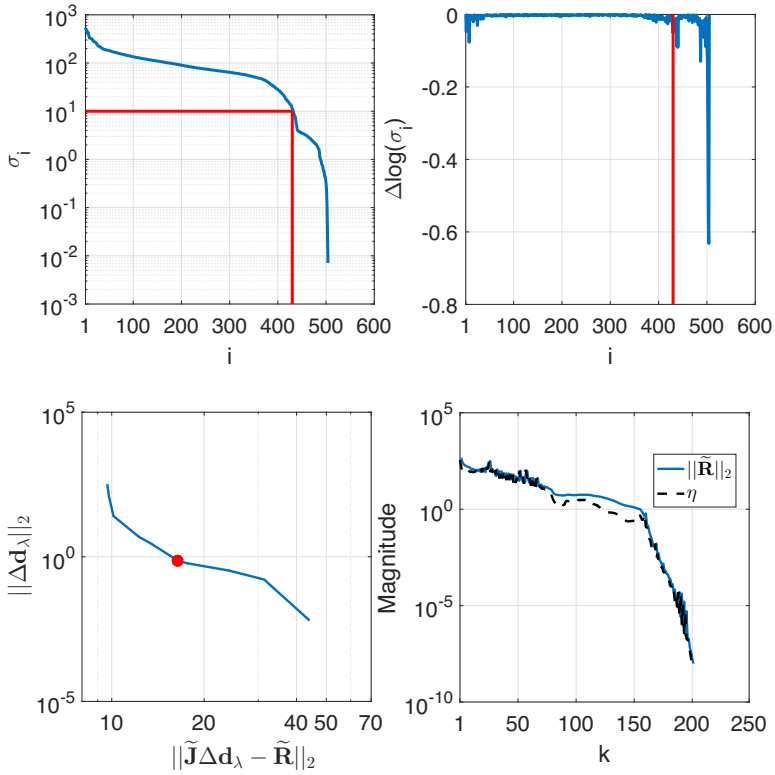

FIG. 4. *Singular values of* $\widetilde{\boldsymbol{J}}$ *with the chosen regularization parameter (top left); finite difference on* log *of singular values and the chosen singular value index (top right); L-curve for the given* $\widetilde{\boldsymbol{J}}$ *and* $\widetilde{\boldsymbol{R}}$*, where the circle indicates the point on the L-curve corresponding to the selected regularization parameter* λ *(bottom left); convergence of the scheme for* $d = 5, r = 7$ *(bottom right).*

can be expressed as

$$\mathcal{I}(f)(\boldsymbol{x}) = \sum_{j=1}^{n} \ell_j(\boldsymbol{x}) f(\boldsymbol{x}_j), \qquad \ell_j(\boldsymbol{x}_k) = \delta_{j,k},$$

where the ℓ_j are the cardinal interpolation functions. The Lebesgue function L_n and the Lebesgue constant L are, respectively,

$$L_n(\boldsymbol{x}) = \sum_{j=1}^{n} |\ell_j(\boldsymbol{x})|, \qquad L = \|L_n\|_\infty.$$

Finding a set of points with minimal Lebesgue constant is not trivial. In $d = 2$ dimensions the Padua points are essentially the only explicitly constructible set of nodes with provably minimal growth of Lebesgue constant on total degree spaces [5]. To compare designed quadrature with Padua points, we consider degree-5 Padua points, yielding $\dim \Pi_{\Lambda_{\mathcal{T}_5}} = 21$. These points, along with associated quadrature weights, integrate polynomials in $\Pi_{\Lambda_{\mathcal{T}_9}}$ exactly with respect to the product Chebyshev weight ω [5].

With designed quadrature we are able to find $17 < 21$ nodes and weights that integrate polynomials in $\Pi_{\Lambda_{\mathcal{T}_9}}$ exactly. However, for the purposes of interpolation

in this section, we enforce $n = 21$ nodes in the designed quadrature framework. To initialize the design we start from nodes that are close to Padua nodes. The Lebesgue function $L_n(\boldsymbol{x})$ for both cases is shown in Figure 5. The Lebesgue constant for Padua points and designed quadrature are $L = 4.9478$ and $L = 5.1553$, respectively. The similar small values of L suggest that the designed quadrature points and the Padua points are of comparable quality in terms of constructing interpolants. However, we reiterate that for quadrature we can use fewer nodes (17) than the Padua points (21).

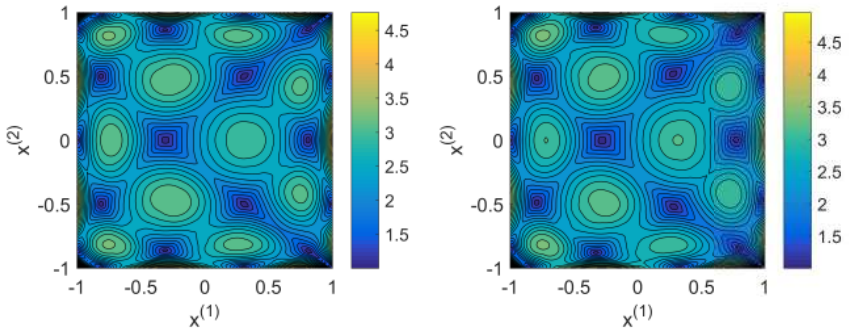

FIG. 5. *Contour plots of Lebesgue function L_n for Padua points (left) and designed quadrature (right).*

4.4. Designed quadrature: U. The formulation of designed quadrature allows Γ and ω to be of relatively general form, but we can construct quadrature rules in even more exotic situations. Let $\Gamma = [-1, 1]^2$ with ω the uniform weight. Instead of enforcing $\boldsymbol{x}_j \in \Gamma$ in (21), we enforce $\boldsymbol{x}_j \in \widetilde{\Gamma}$, where $\widetilde{\Gamma} \subset \Gamma$ is a "U" shape, mimicking the logo of the University of Utah; see Figure 6, left.

The penalty function for this problem has the same quadratic form as those discussed in section 3.1 and separate penalties are considered for violations in both the $x^{(1)}$ and $x^{(2)}$ directions. For example, we can model the infeasible rectangular region $\mathcal{S}_1$ between the two ascenders of the U with nonzero penalty in the $x^{(1)}$ direction and zero penalty in the $x^{(2)}$ direction as

$$\begin{cases} \mathcal{S}_1 : (0 \le |x_1^{(1)}| \le 0.4) \cap (-0.35 \le x^{(2)} \le 0.95), \\ P_1 = (x^{(1)} - 0.4)^2, P_2 = 0. \end{cases} \tag{34}$$

A similar method can be used to penalize the semicircular region below the rectangle where violations in both directions are penalized. The total penalty for infeasible regions then involves both P_1 and P_2, e.g., $P = 10\sqrt{P_1 + P_2}$, which is shown in Figure 6, right. We compute a designed quadrature rule for total degree $r = 2$, achieving residual tolerance of $\epsilon = 0.0098$ with $n = 150$. We need a relatively large number of nodes and achieve only a relatively large tolerance (compared to $\epsilon = 10^{-8}$ for previous examples). This is due to the difficulty of this problem: We want nodes to lie in $\widetilde{\Gamma}$ but want to achieve integration over Γ. We expect that convergence for larger r will require many iterations and may not be able to achieve arbitrarily small tolerances.

4.5. Integration in high dimensions. To demonstrate the capability of designed quadrature for integration in high dimensions, we consider $d = 100$ with hyperbolic cross index set $\Lambda_{\mathcal{H}_r}$. Figure 7 (top) shows different slices of the $d = 100$

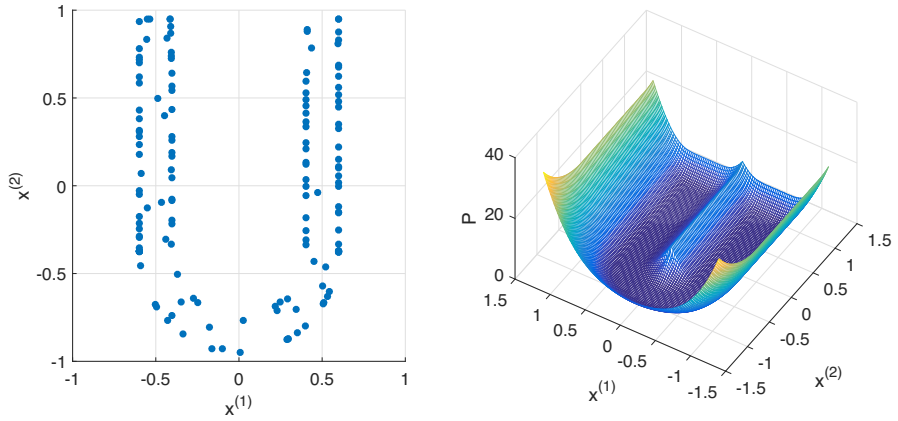

FIG. 6. *Designed quadrature for uniform weight and $d = r = 2$ with "U" shape indicating the University of Utah (left); the penalty function used in the scheme (right).*

nodal configuration generated by designed quadrature for the uniform weight on $\Gamma = [-1, 1]^{100}$ and $r = 4$, for which we have $n = 106$ and $|\Lambda_{\mathcal{H}_4}| = 5351$.

Figure 7 (bottom) shows the behavior of designed quadrature weights with respect to the Euclidean norm of the nodes (distance to the origin) for ω the Gaussian weight on $\Gamma = \mathbb{R}^{100}$ for total order $r = 2$ and hyperbolic cross orders $r = 3, 4$. As expected, the weights decay as the node norms increase. We find $n = 101$ for all these quadratures, again confirming the optimal $n = d + 1$ size for total order $r = 2$ [54]. It is also interesting to note that the minimum Euclidean norms of nodes for these cases are somewhat equal, viz. $||x|| = 8.89,\ 8.93,\ 8.85$, respectively.

Computing designed quadratures in high dimensions reveals computational challenges that are not present in small-to-moderate dimensions: Since the nonlinear system is quite large, we do not perform the SVD of the Jacobian in each iteration. Instead, we regularize the pseudoinverse matrix directly and compute the Newton step as $\Delta \boldsymbol{d} = (\boldsymbol{J}^T \boldsymbol{J} + \lambda \boldsymbol{I})^{-1} \boldsymbol{J}^T \boldsymbol{R}$. The parameter λ can be selected based on the residual norm value, as explained in the previous section. The designed quadrature algorithm for these cases in $d = 100$ took ~ 100 iterations and less than 30 minutes on a personal desktop in MATLAB.

4.6. High-dimensional integration: Linear elasticity problem. To investigate the performance of the high-dimensional quadrature points, we compute the mean and variance of compliance indicative of the elastic energy for a solid cantilever beam with uncertain material properties.

The compliance for the spatial domain Ω reads

$$C = \int_{\Omega} f u d\Omega,$$

where u is the displacement and f is the surface load on the structure. To find u, the equation of motion in linear elasticity $\nabla . \boldsymbol{\sigma} + f = 0$, where $\boldsymbol{\sigma}$ is the stress tensor and ∇ is the divergence operator solved via the finite element method. The global displacement is characterized with n_e finite elements

$$u = \sum_{i=1}^{n_e} u_i \Psi_i,$$

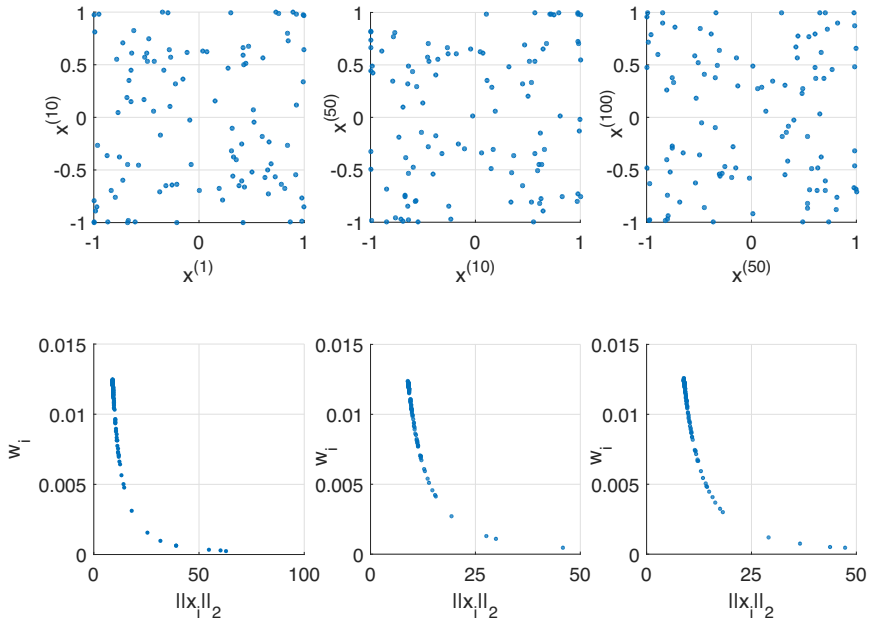

FIG. 7. *Top: different 2-dimensional slices of $d = 100$ designed quadrature for uniform weight and hyperbolic cross order $r = 4$. Bottom: formation of weights with respect to Euclidean norm of nodes for Gaussian weight in $d = 100$ for total order $r = 2$ (left) and hyperbolic cross orders $r = 3$ (middle) and $r = 4$ (right).*

where Ψ_i are finite element shape functions and u_i are nodal displacements. The nodal displacements $\boldsymbol{U} = \{u_i\}_{i=1}^{n_e}$ are solution of a linear system $\boldsymbol{KU} = \boldsymbol{F}$ (stems from the equation of motion), where

$$\boldsymbol{K} = \int_\Omega \frac{\partial \Psi^T}{\partial \boldsymbol{x}} \mathbb{C} \frac{\partial \Psi}{\partial \boldsymbol{x}} d\Omega,$$

$$\boldsymbol{F} = \int_\Omega f \Psi d\Omega,$$

with $\mathbb{C}$ being an elasticity matrix. We consider the plane stress condition in this example; hence for our 2-dimensional problem

$$\mathbb{C} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \dfrac{1-\nu}{2} \end{bmatrix},$$

where E is the modulus of elasticity and ν is Poisson's ratio.

The beam geometry is shown in Figure 8 and is modeled with 100 standard square finite elements, where each element has lognormal modulus of elasticity as $E_i = 10^{-9} + \exp(\xi^{(i)})$. The random variables $\xi^{(i)}|_{i=1}^{100}$ are independent standard random normal variables, and Poisson's ratio is $\nu = 0.3$. Our goal is to compute first- and second-order statistics of the compliance; these statistics are integrals with respect to the 100 variables $\xi^{(i)}$, and so we approximate these statistics via designed quadrature.

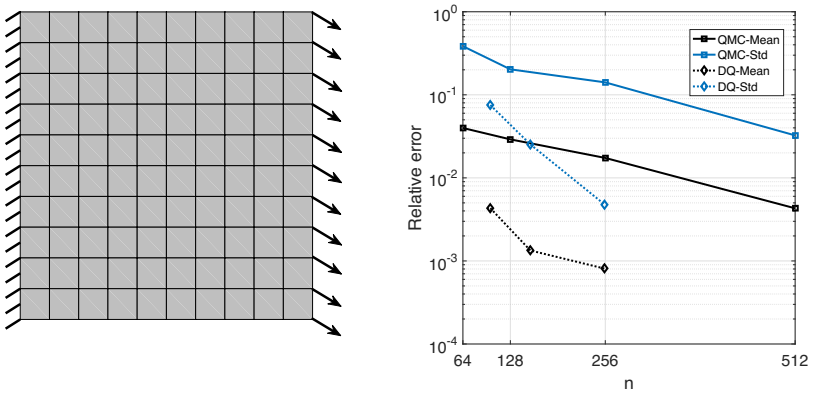

FIG. 8. *Finite element discretization for a linear elastic cantilever beam with random elastic modulus (left) convergence of compliance mean and variance with quasi-Monte Carlo (QMC) samples and designed quadrature (DQ) in $d = 100$ variables (right).*

For comparison against designed quadrature we use quasi-Monte Carlo (QMC) samples of size $n = 64, 128, 256, 512$ and $n = 1024$, where we treat the latter as the exact solution. The QMC samples are generated on $[0,1]^{100}$ and are mapped to $\mathbb{R}^{100}$ using inverse transform sampling for a 100-dimensional standard normal random vector. We have chosen the number of QMC samples so that they almost match the number of designed quadrature nodes computed from (i) the index sets for total order $r = 2$ and $n = 101$ nodes; (ii) the index set $\Lambda_{\mathcal{H}_4} \cup \Lambda_2$ with $n = 155$ nodes, where Λ_2 contains pairwise interactions of maximum univariate order 2; and (iii) the index set $\Lambda_{\mathcal{H}_4} \cup \Lambda_3$ with $n = 255$ nodes, where Λ_3 contains pairwise interactions of maximum univariate order 3.

Figure 8 compares errors in the computed mean and standard deviation of the compliance for QMC versus designed quadrature; we see that designed quadrature achieves significantly better errors. We believe QMC would be more effective if the problem involved many more variables, larger index sets, and/or nonsmooth quantities of interest, since in those cases the number of designed quadrature nodes is prohibitively large and finding a suitable quadrature rule is computationally challenging.

4.7. Designed quadrature for topology optimization under uncertainty. Our final example utilizes polynomial chaos (PC) methods [17] to build surrogates for topology optimization under geometric uncertainty [25]. Figure 9 shows the flowchart for design optimization under uncertainty. To build PC surrogates at each design iteration, n finite element analysis (FEA) and sensitivity analyses are performed in order to quantify the uncertainty associated with random variables $\xi^{(i)}$ as in the last example. This is the most costly step in the design process, and hence small n can result in significant computational savings.

The perturbation in the boundary of topology interfaces Z are modeled via a Karhunen–Loève random field with $d = 4$ significant modes as

$$Z(x, \boldsymbol{\xi}) = \sum_{i=1}^{4} \sqrt{\lambda_i} \gamma^{(i)}(x) \xi^{(i)}, \tag{35}$$

where $\xi^{(i)} \in U[-\sqrt{3}, \sqrt{3}]$ are independent uniform random variables. Sparse grids

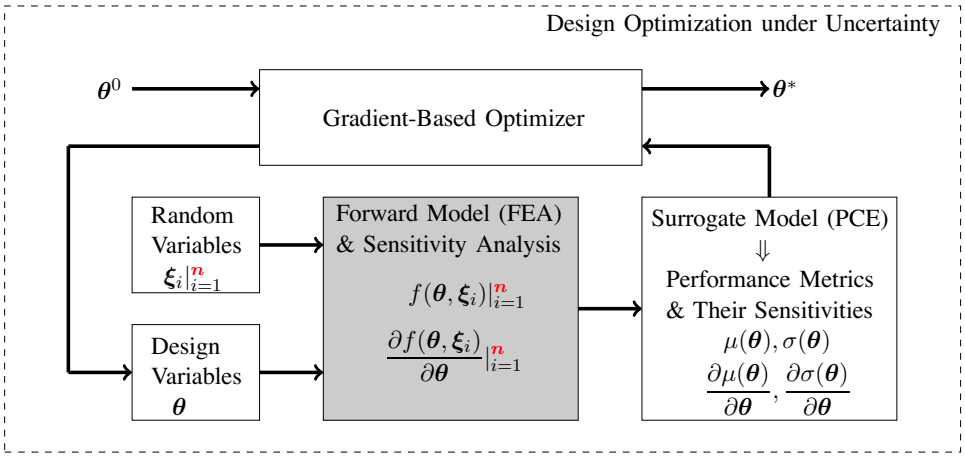

FIG. 9. *Design optimization under uncertainty flowchart.*

built from nested rules were utilized in [25] to develop a surrogate for total degree $r = 3$ in $d = 4$ dimensions. This requires a quadrature rule that can accurately integrate polynomials up to order $r = 6$ (see Proposition 2.3). A standard construction of sparse grid rules yields odd orders of polynomial accuracy, and hence a quadrature rule for $r = 7$ with $n = 81$ nodes is used. We observe that 33 out of these 81 nodes have negative weights [22]. On the other hand, we use designed quadrature constrained to integrate polynomials up to degree $r = 6$ and compute $n = 43$ nodes, almost half ($\sim 53\%$) the number of sparse grid points ($43/81 = 0.53$), and in which all nodes have positive weights. These nodes and weights are listed in Table 3.

We approximate the mean and variance for the final robust topology design of the Messerschmitt–Bölkow–Blohm (MBB) beam shown in Figure 10 (left) with both quadrature sets, and we use a sparse grid rule with $n = 641$ points ($r = 13$) as the "true" solution. The mean and standard deviation for the true solution are $\mu_{true} = 120.1032$, $\sigma_{true} = 0.7853$, respectively. The mean and standard deviation and the relative errors in mean $e_\mu = |(\mu - \mu_{true})/\mu_{true}|$ and in standard deviation $e_\sigma = |(\sigma - \sigma_{true})/\sigma_{true}|$ are listed in Table 4. We achieve higher accuracy with designed quadrature at nearly half the cost. Figure 10 also visually compares the probability density function (PDF) of compliance for both cases, and no substantial difference is observed.

5. Concluding remarks. We present a systematic approach, designed quadrature, for computing multivariate quadrature rules in generic settings. The framework uses penalty methods in constrained optimization to ensure positivity of the weights and feasible locations for the nodes. The Gauss–Newton algorithm is used to perform minimization of the penalty-augmented objective function. L^2 regularization is utilized to treat ill-conditioned systems encountered during Newton step updates. On regular domains such as hypercubes, our designed quadrature results in considerably fewer nodes (and guaranteed positive weights) compared to alternative multivariate quadrature rules, such as sparse grids, and hence is promising for computational science and engineering involving expensive simulations. When applied to a benchmark robust topology optimization problem, designed quadrature reduces requisite cost by nearly half compared with sparse grid rules, and achieves higher accuracy.

TABLE 3
Designed nodes and weights for uniform wight function associated with $d = 4, r = 6$.

$x^{(1)}$	$x^{(2)}$	$x^{(3)}$	$x^{(4)}$	w
0.257802083101815	-0.0703252346579532	0.962710865388279	0.430231485089995	0.0261815176727414
-0.816973940130726	-0.943714906761859	0.386890523282751	-0.999999000000035	0.00580285921526766
-0.947032046309044	0.989213881193871	0.936667215690650	-0.993786957614627	0.00249952956479966
-0.410640873236206	0.954255232732162	-0.147886263760743	0.759793319115318	0.0183482544886740
0.231143583536335	0.304638143131838	-0.528664154404583	0.0739711055845102	0.0410433205157124
-0.778491697308234	0.966653453252213	-0.383947457004414	-0.527873684868762	0.0139368201993277
-0.180274497367222	-0.0792041207370926	0.828782356307522	-0.777765421528772	0.0315300612985717
0.540286687837802	-0.208086450042028	-0.948367080027728	-0.501356657306036	0.0215578327462099
-0.683026871793909	0.647010501071650	-0.973583715205816	0.370583426726367	0.0161108573238146
-0.993338263553449	-0.368646129691905	-0.737371067829234	-0.348475820258963	0.0147573865118333
0.407947779669019	-0.815591287366472	-0.0796807556349552	0.378883833053981	0.0331223595693260
0.938583690749431	-0.673570409626489	0.750349855298878	0.515481332659911	0.0138830007891160
0.676361893172788	-0.0863458892282786	0.307033851877092	-0.222719471684161	0.0666800814123564
0.871158904206783	0.833748485858754	-0.0988153767941961	0.190418047822592	0.0231223581306122
-0.637055255430739	0.561219983742431	0.650334703413941	-0.0934633784282683	0.0464871742390699
-0.724596294549971	-0.469996083934142	0.597859508917895	0.576481467431314	0.0349899149656204
-0.0633965531889436	0.107495869838039	0.150581945391035	0.708624486280424	0.0579979230656018
0.447670409694807	0.676539689861564	-0.761277790685523	0.818329189952187	0.0206811254936501
-0.421094936258687	-0.384610025091731	-0.645174618242432	0.192061679511636	0.0430978935629120
0.144084034877935	0.920034465753694	0.719442066934779	0.265836622954943	0.0226831954639939
-0.511575703485635	-0.940233287027892	-0.881122093604340	-0.679836608445536	0.00914818101224769
0.903361192617833	-0.930618551587704	-0.556114178416505	-0.368802948869006	0.0133824642384102
-0.451300661973021	-0.678204534689092	0.109872507715082	-0.268415947407672	0.0469834603269315
0.570179507325998	0.946048343399397	-0.888875656537771	-0.502288128155230	0.0114411429728491
0.947673444496712	-0.0487093539097375	-0.745420922954487	0.483440076541598	0.0180891170799238
-0.151578782518484	0.536570691044281	-0.432491091797662	-0.296255229184200	0.0429298801103472
0.200254358361331	-0.574573679946025	-0.370917668694681	-0.848237515108908	0.0354514497467619
0.703268719144780	-0.538649441561851	-0.243402446610143	0.932031263765523	0.0178590105949260
-0.851069823343954	-0.999999000000274	-0.332766663879255	0.640694186502822	0.0109542526977389
0.778210999922193	-0.689412923038060	0.645938727211798	-0.866284593236453	0.0141564568596360
0.861114502173077	0.635646385445806	0.943650909951228	-0.537424461094664	0.0122992882684838
0.923895085294775	0.356961947592050	-0.390843097452625	-0.911346350395596	0.0147169035847508
0.224936370766468	0.759504704125777	0.295151117998272	-0.810538235949664	0.0343456718994955
0.753566197170202	0.547890728899287	0.582098594572620	0.865739837846072	0.0201259507621183
-0.999996169574511	0.450595235653401	-0.0687856135220421	0.522619830858851	0.0196754195407364
-0.740580150798581	0.648489590619417	0.938458459820395	0.997858479760182	0.00660961411039667
-0.641393244038470	-0.177020578876251	-0.678531778799864	0.993823966856282	0.0158119117410319
0.101688221877749	-0.925804660338255	0.769880631707084	-0.240864991838435	0.0208587330250274
0.183235217131754	-0.787109851166051	-0.944143000956805	0.613293643593588	0.0158948230283889
-0.0665844458312762	-0.866922211430191	0.698247471706727	0.981025573223252	0.0101514764249415
-0.494502217917924	0.382211470283539	-0.824748517891121	-0.935346887279527	0.0160972432916053
-0.973108310610189	-0.643578784698889	0.999996903007507	-0.263921398566453	0.00715355417527909
-0.800132635705771	0.0390203514319736	0.0939369967131938	-0.741463689395994	0.0313505282787613

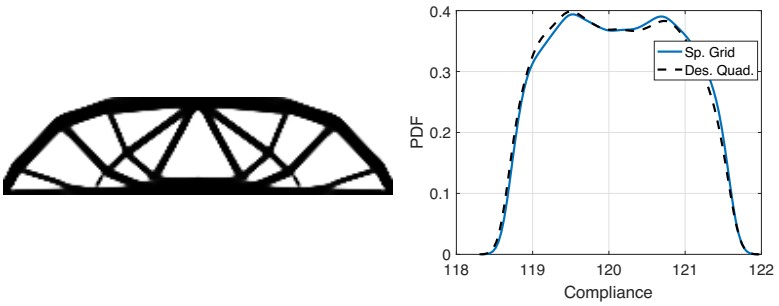

FIG. 10. *Robust topology design of the MBB beam (left) compliance PDF with sparse grid and designed quadrature (right).*

TABLE 4
Mean and standard deviation estimation for the robust topology design.

Quadrature rule	μ	e_μ	σ	e_σ	Cost
Sparse grid	120.1326	2.44e-04	0.7836	2.16e-03	81 simulations
Designed quadrature	120.1028	3.33e-06	0.7854	1.27e-04	43 simulations

REFERENCES

[1] N. AGARWAL AND N. R. ALURU, *A domain adaptive stochastic collocation approach for analysis of MEMS under uncertainties*, J. Comput. Phys., 228 (2009), pp. 7662–7688, https://doi.org/10.1016/j.jcp.2009.07.014.

[2] I. BABUŠKA, F. NOBILE, AND R. TEMPONE, *A stochastic collocation method for elliptic partial differential equations with random input data*, SIAM Rev., 52 (2010), pp. 317–355, https://doi.org/10.1137/100786356.

[3] C. BERNARDI AND Y. MADAY, *Spectral methods*, in Techniques of Scientific Computing (Part 2), Volume 5, P. Ciarlet and J. Lions, eds., Elsevier, 1997, pp. 209–485, https://doi.org/10.1016/S1570-8659(97)80003-8.

[4] D. BERTSEKAS, *Nonlinear Programming*, 2nd ed., Athena Scientific, 2008.

[5] L. BOS, M. CALIARI, M. VIANELLO, S. DE MARCHI, AND Y. XU, *Bivariate Lagrange interpolation at the Padua points: The generating curve approach*, J. Approx. Theory, 143 (2006), pp. 15–25.

[6] S. BOYD AND L. VANDENBERGHE, *Convex Optimization*, Cambridge University Press, 2004.

[7] H. BUNGARTZ AND M. GRIEBEL, *Sparse grids*, Acta Numer., 13 (2004), pp. 147–269.

[8] H.-J. BUNGARTZ AND S. DIRNSTORFER, *Multivariate quadrature on adaptive sparse grids*, Computing, 71 (2003), pp. 89–114, https://doi.org/10.1007/s00607-003-0016-4.

[9] M. CALIARI, S. DE MARCHI, AND M. VIANELLO, *Bivariate polynomial interpolation on the square at new nodal sets*, Appl. Math. Comput., 165 (2005), pp. 261–274, https://doi.org/10.1016/j.amc.2004.07.001.

[10] C. CANUTO, M. Y. HUSSAINI, A. QUARTERONI, AND T. A. ZANG, *Spectral Methods: Fundamentals in Single Domains*, 1st ed., Springer, Berlin, New York, 2006; corr. 4th printing, 2011.

[11] C. CANUTO, M. Y. HUSSAINI, A. QUARTERONI, AND T. A. ZANG, *Spectral Methods: Evolution to Complex Geometries and Applications to Fluid Dynamics*, Springer, Berlin, 2007; 2nd ed., 2014.

[12] A. COHEN, R. DEVORE, AND C. SCHWAB, *Convergence rates of best N-term Galerkin approximations for a class of elliptic sPDEs*, Found. Comput. Math., 10 (2010), pp. 615–646, https://doi.org/10.1007/s10208-010-9072-2.

[13] P. G. CONSTANTINE, M. S. ELDRED, AND E. T. PHIPPS, *Sparse pseudospectral approximation method*, Comput. Methods Appl. Mech. Engrg., 229 (2012), pp. 1–12, https://doi.org/10.1016/j.cma.2012.03.019.

[14] P. DAVIS AND P. RABINOWITZ, *Methods of Numerical Integration*, Courier Corporation, 2007, Chap. 2.

[15] W. GAUTSCHI, *Construction of Gauss-Christoffel quadrature formulas*, Math. Comp., 22 (1968), pp. 251–270.

[16] T. GERSTNER AND M. GRIEBEL, *Numerical integration using sparse grids*, Numer. Algorithms, 18 (1998), pp. 209–232, https://doi.org/10.1023/A:1019129717644.

[17] R. GHANEM AND P. SPANOS, *Stochastic Finite Elements: A Spectral Approach*, Dover, 2002.

[18] G. GOLUB AND J. WELSCH, *Calculation of Gauss quadrature rules*, Math. Comp., 23 (1969), pp. 221–230.

[19] G. H. GOLUB AND C. F. V. LOAN, *Matrix Computations*, 3rd ed., Johns Hopkins Stud. Math. Sci., The Johns Hopkins University Press, 1996.

[20] P. C. HANSEN, *Rank-Deficient and Discrete Ill-Posed Problems*, SIAM, 1998.

[21] P. C. HANSEN AND D. P. O'LEARY, *The use of the L-curve in the regularization of discrete ill-posed problems*, SIAM J. Sci. Comput., 14 (1993), pp. 1487–1503, https://doi.org/10.1137/0914086.

[22] F. HEISS AND V. WINSCHEL, *Quadrature on Sparse Grids*, http://www.sparse-grids.de/.

[23] F. HEISS AND V. WINSCHEL, *Likelihood approximation by numerical integration on sparse grids*, J. Econometrics, 144 (2008), pp. 62–80.

[24] J. JAKEMAN AND A. NARAYAN, *Generation and Application of Multivariate Polynomial Quadrature Rules*, preprint, https://arxiv.org/abs/1711.00506, 2017.

[25] V. Keshavarzzadeh, F. Fernandez, and D. Tortorelli, *Topology optimization under uncertainty via non-intrusive polynomial chaos expansion*, Comput. Methods Appl. Mech. Engrg., 318 (2017), pp. 120–147.
[26] M. Liu, Z. Gao, and J. S. Hesthaven, *Adaptive sparse grid algorithms with applications to electromagnetic scattering under uncertainty*, Appl. Numer. Math., 61 (2011), pp. 24–37, https://doi.org/10.1016/j.apnum.2010.08.002.
[27] D. Lubinsky, *A survey of weighted polynomial approximation with exponential weights*, Surv. Approx. Theory, 3 (2007), pp. 1–105.
[28] D. Luenberger and Y. Ye, *Linear and Nonlinear Programming*, Internat. Ser. Oper. Res. Management Sci. 116, Springer, 2008.
[29] J. Ma, V. Rokhlin, and S. Wandzura, *Generalized Gaussian quadrature rules for systems of arbitrary functions*, SIAM J. Numer. Anal., 33 (1996), pp. 971–996, https://doi.org/10.1137/0733048.
[30] S. E. Mousavi, H. Xiao, and N. Sukumar, *Generalized Gaussian quadrature rules on arbitrary polygons*, Internat. J. Numer. Methods Engrg., 82 (2010), pp. 99–113, https://doi.org/10.1002/nme.2759.
[31] A. Narayan and J. D. Jakeman, *Adaptive Leja sparse grid constructions for stochastic collocation and high-dimensional approximation*, SIAM J. Sci. Comput., 36 (2014), pp. A2952–A2983, https://doi.org/10.1137/140966368.
[32] A. Narayan, J. Jakeman, and T. Zhou, *A Christoffel function weighted least squares algorithm for collocation approximations*, Math. Comp., 86 (2017), pp. 1913–1947.
[33] H. Niederreiter, *Random Number Generation and Quasi-Monte Carlo Methods*, SIAM, 1992.
[34] A. B. Owen, *Quasi-Monte Carlo sampling*, in Monte Carlo Ray Tracing: SIGGRAPH 2003, Course 44, 2003, pp. 69–88.
[35] T. Patterson, *The optimum addition of points to quadrature formulae*, Math. Comp., 22 (1968), pp. 847–856.
[36] E. Ryu and S. Boyd, *Extensions of Gauss quadrature via linear programming*, Found. Comput. Math., 15 (2015), pp. 953–971.
[37] I. H. Sloan and H. Wozniakowski, *When are Quasi-Monte Carlo algorithms efficient for high dimensional integrals?*, J. Complexity, 14 (1998), pp. 1–33, https://doi.org/10.1006/jcom.1997.0463.
[38] S. Smolyak, *Quadrature and interpolation formulas for tensor products of certain classes of functions*, Soviet Math. Dokl., 4 (1963), pp. 240–243.
[39] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*, Texts Appl. Math. 12, Springer-Verlag, 2002.
[40] A. Stroud, *Some fifth degree integration formulas for symmetric regions* II, Numer. Math., 9 (1967), pp. 460–468.
[41] A. Stroud, *Approximate Calculation of Multiple Integrals*, Prentice-Hall, 1971.
[42] A. H. Stroud, *Remarks on the disposition of points in numerical integration formulas*, Math. Tables Other Aids Comput., 11 (1957), pp. 257–261.
[43] A. H. Stroud, *Numerical integration formulas of degree two*, Math. Comp., 14 (1960), pp. 21–26, https://doi.org/10.2307/2002981.
[44] G. Szegö, *Orthogonal Polynomials*, 4th ed., AMS, 1975.
[45] M. A. Taylor, B. A. Wingate, and L. P. Bos, *A cardinal function algorithm for computing multivariate quadrature points*, SIAM J. Numer. Anal., 45 (2007), pp. 193–205, https://doi.org/10.1137/050625801.
[46] M. A. Taylor, B. A. Wingate, and R. E. Vincent, *An algorithm for computing Fekete points in the triangle*, SIAM J. Numer. Anal., 38 (2000), pp. 1707–1720, https://doi.org/10.1137/S0036142998337247.
[47] R. Tibshirani, *Regression shrinkage and selection via the lasso*, J. Roy. Statist. Soc. Ser. B, 58 (1996), pp. 267–288.
[48] M. van Barel, M. Humet, and L. Sorber, *Approximating optimal point configurations for multivariate polynomial interpolation*, Electron. Trans. Numer. Anal., 42 (2014), pp. 41–63.
[49] E. Van den berg and M. Friedlander, *SPGL*1: *A Solver for Large-Scale Sparse Reconstruction*, http://www.cs.ubc.ca/~mpf/spgl1/.
[50] E. Van den berg and M. P. Friedlander, *Probing the Pareto frontier for basis pursuit solutions*, SIAM J. Sci. Comput., 31 (2008), pp. 890–912, https://doi.org/10.1137/080714488.
[51] E. Van den berg and M. P. Friedlander, *Sparse optimization with least-squares constraints*, SIAM J. Optim., 21 (2011), pp. 1201–1229, https://doi.org/10.1137/100785028.
[52] G. Wasilkowski and H. Wozniakowski, *Explicit cost bounds of algorithms for multivariate tensor product problems*, J. Complexity, 11 (1995), pp. 1–56.
[53] H. Xiao and Z. Gimbutasb, *A numerical algorithm for the construction of efficient quadrature*

rules in two and higher dimensions, Comput. Math. Appl., 59 (2010), pp. 663–676.
[54] D. Xiu, *Numerical integration formulas of degree two*, Appl. Numer. Math., 58 (2008), pp. 1515–1520, https://doi.org/16/j.apnum.2007.09.004.
[55] D. Xiu and J. S. Hesthaven, *High-order collocation methods for differential equations with random inputs*, SIAM J. Sci. Comput., 27 (2005), pp. 1118–1139, https://doi.org/10.1137/040615201.
[56] Y. Xu, *A characterization of positive quadrature formulae*, Math. Comp., 62 (1994), pp. 703–718, https://doi.org/10.1090/S0025-5718-1994-1223234-0.