ABSTRACT

MORROW, ZACHARY BENJAMIN. Sparse-Grid Surrogate Models in Computational Chemistry. (Under the direction of Carl T. Kelley.)

Molecules naturally adopt a geometry that at least locally minimizes potential energy. With a potential energy surface (PES), one can identify the minimizers, as well as how to move between local minima, by examining dynamics on the PES. The full PES of a molecule with N atoms is a function of $3N - 6$ coordinates. A relaxed PES is a function of fewer variables, but evaluations of it require an expensive optimization process. This thesis describes how to construct a surrogate for the relaxed PES with sparse interpolation. We present an algorithm that constructs an interpolant as a linear combination of sines and cosines [84]. This technique preserves the periodicity of the PES gradient for cases where the relaxed PES is a function of only full rotations. To our knowledge, this is a novel approach for PES surrogate modeling in computational chemistry. The author implemented this sparse interpolation algorithm, along with dimensionally adaptive refinement, in the Tasmanian package, an open-source sparse-grid tool developed at Oak Ridge National Laboratory [200]. We applied this all-periodic approximation to a reduced-dimensional representation of a tungsten molecule.

However, many chemically relevant systems involve both periodic and nonperiodic degrees of freedom. In those cases, we use a mixed-basis interpolation method that applies sparse trigonometric interpolation to the periodic components and sparse polynomial interpolation to the nonperiodic components. Using the azomethane molecule as a test case, this method needs fewer nodes than the previous state of the art (all-polynomial interpolation) to obtain a given level of accuracy and is an order of magnitude cheaper computationally. Perhaps most importantly, the mixed-basis interpolant accurately conserves total energy in microcanonical molecular dynamics simulation, whereas all-polynomial interpolation fails to conserve energy if a periodic component crosses the periodic boundary.

The final method presented in this thesis is an adaptation of Tully's fewest-switches surface hopping (FSSH) algorithm [214] in a reduced-dimensional setting. In addition to approximating the PES with mixed-basis sparse interpolation, this implementation performs expensive computations of the nonadiabatic coupling (NAC) corresponding to various geometries at the front end. At each time step, it approximates the NAC at an arbitrary geometry value as a weighted sum of the known NACs within a particular geometry radius. It therefore avoids the main expense of surface hopping: direct queries of the NACs and PESs. Using singlet-state photoexcitation of azomethane as an example, this method is able to replicate experimental results that have thus far eluded *ab initio* studies.

Sparse-Grid Surrogate Models in Computational Chemistry

by
Zachary Benjamin Morrow

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Applied Mathematics

Raleigh, North Carolina

2021

APPROVED BY:

DEDICATION

To the glory of God, and to Lauren and my parents for their steadfast support.

BIOGRAPHY

Zack Morrow was born and raised in the piedmont of South Carolina. After graduating from Andrew Jackson High School in 2012, he attended Wofford College. He finished in 2016 with degrees in Mathematics and Economics, *summa cum laude*. At that point, he started graduate school at North Carolina State University, earning his M.S. in 2018 and his Ph.D. in 2021. He held internships at the U.S. Army Engineer Research and Development Center (2015-2016) and Oak Ridge National Laboratory (2018-2019).

ACKNOWLEDGEMENTS

This thesis would not be possible without the help of several people who invested mightily in me and my research project.

The first to thank is my wife, Lauren. She has been wonderfully supportive during summers of internships and during the frustrations of slow research progress. I do not know where I would be without her by my side.

Next is my advisor, Tim Kelley. Tim's enthusiasm, drive, and characteristic bluntness encouraged me to be a better mathematician. I always looked forward to our research meetings.

Then there are Elena Jakubikova and Miroslav Stoyanov. Elena taught me everything I know about Gaussian 16 and computational chemistry. Miro walked with me as I learned to contribute to production DOE software.

This work has been funded by NSF DMS-1406349; an NSF Graduate Research Fellowship under NSF DGE-1746939; and two summers (2018 and 2019) of participation in the Oak Ridge National Laboratory (ORNL) ASTRO program, sponsored by the U.S. Department of Energy (DOE) and administered by the Oak Ridge Institute for Science and Education (ORISE). The views, findings, and opinions expressed in this dissertation are the author's and do not necessarily reflect the views of the NSF, ORNL, ORISE, DOE, or NC State University.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER

1 INTRODUCTION

Molecules naturally adopt a stable geometry that is a minimizer, at least locally, of a potential energy surface (PES). Different molecular geometries could potentially result in different physical and chemical properties, so there is much interest in determining useful geometries for various needs in scientific, engineering, and technological applications.

In a molecule with N atoms, the geometry $\boldsymbol{x}$ is completely determined by $3N-6$ internal coordinates: $N-1$ bond lengths, $N-2$ bond angles, and $N-3$ dihedral angles. For a fixed geometry $\boldsymbol{x}$, the Schrödinger equation (see Chapter 4) states

$$\hat{\mathcal{H}}\,\Psi_i = \mathcal{E}_i\,\Psi_i, \qquad\qquad i = 0, 1, 2, \ldots, \tag{1.1}$$

where $\hat{\mathcal{H}}$ is the molecular Hamiltonian operator, Ψ_i is the wavefunction of energy state i, $\mathcal{E}_i$ is the energy of state i, and all three depend parametrically on the full geometry $\boldsymbol{x}$. As a function of geometry, $\mathcal{E}_i(\boldsymbol{x})$ is the full PES for energy level i. Often there are many components of geometry that are not of interest in particular applications (or do not vary significantly), so we may partition

$$\boldsymbol{x} = (\boldsymbol{q}, \boldsymbol{\xi})\,.$$

Here, $\boldsymbol{q} \in \mathbb{R}^d$ are the *design variables*, $\boldsymbol{\xi}$ are the *remainder variables*, and typically $d \ll 3N-6$. In practice, one chooses the design variables based on chemical intuition or *a priori* knowledge of the system, though there have been recent studies on automated detection of $\boldsymbol{q}$ based on molecular dynamics trajectories [86]. Rather than considering the $(3N-6)$-dimensional PES, we consider the so-called "relaxed PES"[1]

$$E_i(\boldsymbol{q}) = \min_{\boldsymbol{\xi}} \mathcal{E}_i(\boldsymbol{q}, \boldsymbol{\xi}) \,. \tag{1.2}$$

While the relaxed PES's dependence on $\boldsymbol{q} \in \mathbb{R}^d$ does reduce the dimensionality of the problem, solving (1.1)–(1.2) is computationally intensive. There is not only an eigenvalue problem but also a multi-dimensional optimization. Each step of the optimization algorithm that solves (1.2) must compute a search direction by approximating a solution to (1.1). Though there are many approaches to approximating a solution to (1.1), we use density functional theory (DFT), whose computational cost scales like $\mathcal{O}(N^3)$.

The high cost of directly evaluating the PES is what makes it infeasible in most applications. For instance, suppose we are attempting to follow the reaction path

$$\frac{d\boldsymbol{q}}{dt} = -\nabla E_i(\boldsymbol{q}),$$

which is a common equation to solve in computational chemistry [144, 146, 147]. We would integrate the dynamics with some symplectic ODE integrator, which requires evaluations of E_i at a particular points $\{\boldsymbol{q}_j\}$ determined by the integrator, as well as any evaluations needed as part of a nonlinear solve. Integrating even this simple system would far too computationally expensive.

Surrogate modeling provide a means to keep the computational cost in check. Rather than directly evaluating E_i inside a chemistry simulation, we construct a surrogate PES $E_i^s(\boldsymbol{q})$ that is cheap to evaluate. To build the interpolant, we need evaluations of E_i only at a set of nodes $\{\boldsymbol{q}^k\}_{k=1}^{N_{data}}$. Here, N_{data} is the number of expensive PES evaluations, identical to the number of nodes, which represents an up-front cost. Famously, tensor-product grids have exponential scaling in the dimension d, known as the *curse of dimensionality* [17, 50, 68], and modern PES approximation uses methods that scale sub-exponentially or better.

While this thesis focuses on sparse interpolation as the method of choice, we will briefly describe other methods. Broadly, these methods fall into two categories: fitting and interpolative. Fitting methods include the (double) many-body expansion (MBE / DMBE) [181], permutationally invariant polynomials by least-squares fit (PIP) [2, 28, 29, 33, 39, 178], neural network PESs (nn-PES) [99, 130, 135], and the finite-element method [20–22]. Interpolative methods include sparse polynomial and trigonometric interpolation, modified Shepard interpolation, and (possibly local) interpolative moving least-squares (IMLS / L-IMLS) [132–134]. Both classes

[1]From now on, if we say "PES," it should be understood as the relaxed PES.

express the surrogate PES as

$$E^s(\boldsymbol{q}) = \sum_{j=1}^{M} c_j(\boldsymbol{q})\phi_j(\boldsymbol{q}), \tag{1.3}$$

where c_j are coefficients and ϕ_j are the basis functions in the expansion.[2] To determine the c_j, interpolative methods seek to create exact agreement between $E^s(\boldsymbol{q})$ and $E(\boldsymbol{q})$ at a set of nodes $\{\boldsymbol{q}^k\}$, while fitting methods minimize the discrepancy

$$D = \sum_{k=1}^{N_{data}} \sum_{j=1}^{M} \Big(c_j(\boldsymbol{q}^k)\phi_j(\boldsymbol{q}^k) - E(\boldsymbol{q}^k) \Big)^2$$

to a minimum that may be nonzero.To have a completely determined system of equations, interpolative methods require $N_{data} = M$, while for fitting methods, $N_{data} \geq M$ in general.

We now briefly describe two interpolative methods that are not the focus of this thesis. Modified Shepard interpolation takes ϕ_j to be the Taylor expansion of $E(\boldsymbol{q})$ about $\boldsymbol{q}^j$ and c_j to be a weighting function that peaks at $\boldsymbol{q}^j$ [68]. IMLS modifies the discrepancy to weight each difference:

$$D(\boldsymbol{q}) = \sum_{k=1}^{N_{data}} w_k(\boldsymbol{q}) \sum_{j=1}^{M} \Big(c_j(\boldsymbol{q}^k)\phi_j(\boldsymbol{q}^k) - E(\boldsymbol{q}^k) \Big)^2 \tag{1.4}$$

In IMLS, the optimal coefficients c_j depend on $\boldsymbol{q}$ due to the weighting function $w_k(\boldsymbol{q})$, which is chosen to be large near $\boldsymbol{q}^k$ and small otherwise. However, in IMLS, one must solve (1.4) at each evaluation point, which can become cost-prohibitive. The local variant (L-IMLS) [71, 83] solves (1.4) at each $\boldsymbol{q}^k$ to obtain $c_{kj} = c_j(\boldsymbol{q}^k)$ and then constructs the local approximations

$$v_k(\boldsymbol{q}) = \sum_{j=1}^{M} c_{kj}\phi_j(\boldsymbol{q})\,.$$

Then, the overall L-IMLS surrogate is

$$E^s(\boldsymbol{q}) = \sum_{k=1}^{N_{data}} \bar{w}_k(\boldsymbol{q}) v_k(\boldsymbol{q})$$

where the global weighting functions $\bar{w}_k$ need not be the same as the least-squares weighting functions w_k.

As noted before, our focus here is on sparse interpolation. This thesis explores an important extension of the work of previous group members [125, 144, 146, 147]. For cases where all components of $\boldsymbol{q}$ are periodic (e.g., full-rotation torsion angles), both the interpolant and its

[2]Without loss of generality, we suppress the electronic state in the PES notation and focus on the ground state, $i = 0$.

gradient must be periodic. Polynomial interpolation does not preserve periodicity of ∇E, which leads to nonphysical occurrences in some chemistry simulations, such as nonconservation of total energy. When each component of $\boldsymbol{q}$ is periodic, we construct the interpolant as a linear combination of sines and cosines. By design, this preserves periodicity of ∇E^s. When there is a mixture of periodic and nonperiodic components, we apply trigonometric interpolation to the periodic parts and polynomial interpolation to the nonperiodic parts.

We now describe the organization of this thesis. In Chapter 2, we develop sparse grids for both polynomial and trigonometric interpolation bases; we also present a novel adaptive refinement algorithm and mixed-basis interpolation method. In Chapter 3, we discuss the open-source sparse-grid tool Tasmanian, as well as the author's contributions to the package. In Chapter 4, we overview the quantum and computational chemistry underlying our PES computations. In Chapter 5, we apply sparse trigonometric interpolation to a tungsten molecule. In Chapter 6, we use mixed-basis interpolation to drive molecular dynamics (MD) simulations of azomethane and demonstrate that the mixed-basis method leads to energy conservation whereas all-polynomial interpolation does not. In Chapter 7, we use Tully's fewest-switches surface hopping (FSSH) algorithm in a reduced-dimensional setting, using the mixed-basis surrogate PES and an offline–online paradigm for computing the couplings between electronic states. The original work is as follows:

- Implementation of sparse trigonometric interpolation in Tasmanian (Chapters 2–3),
- Adaptive refinement algorithm for sparse trigonometric interpolation (Chapter 2),
- Approximation of PES with trigonometric interpolation (Chapter 5),
- Mixed-basis PES interpolation, which enforces energy conservation (Chapter 6), and
- Reduced-dimensional surface hopping and approximation of nonadiabatic coupling vectors (Chapter 7).

CHAPTER

2 SPARSE INTERPOLATION

2.1 Motivation and background

The chemistry simulations depend on a potential energy surface (PES), which, as a function of molecular geometry, is prohibitively expensive to evaluate directly during temporal integration. In order to maintain computational feasibility, we build a sparse interpolant of the true PES as a surrogate model. We use the surrogate PES within our time integration to drive the molecular dynamics.

In high-dimensional approximation, whether with interpolation or quadrature, the naïve approach is to take a tensor product of the same one-dimensional rules. For concreteness, a stereotypical picture of a full-tensor grid is shown in the lower left panel of Figure 2.3. In the full-tensor approach, if the one-dimensional approximation grid has N points, the d-dimensional grid will have N^d points. Thus, the total size of the grid increases exponentially in d. This phenomenon is called the *curse of dimensionality*, a term originally coined by Bellman [17].

Various techniques exist to mitigate, or in rare cases eliminate, the curse of dimensionality. Global and derivative-based sensitivity analysis enables the identification of noninfluential parameters, thereby reducing the effective dimensionality of the problem to include only inputs and directions that contribute towards the model output variability [87, 205]. Other methods seek to reduce the complexity of the target function f by approximating it with functions that are in some sense "simpler." This could be done, for instance, by projection onto or interpolation within a polynomial or trigonometric function space, both of which use samples, i.e., the values of the target function for a set of independent inputs. Sampling methods are attractive due to their non-intrusive nature, which can be easily wrapped around existing third-party or black-box models.

Let $\{\varphi_{\boldsymbol{\nu}}\}_{\boldsymbol{\nu}\in\mathbb{N}^d}$ be an orthonormal basis for the Hilbert space where f resides (e.g., L^2), and let $\Lambda \subset \mathbb{N}^d$ be finite.[1] We first discuss approximation by L^2 orthogonal projection. A well-known result states that orthogonal projection of f onto $\operatorname{span}\{\varphi_{\boldsymbol{\nu}}\}_{\boldsymbol{\nu}\in\Lambda}$ yields the optimal L^2 error [111]; that is,

$$c_{\boldsymbol{\nu}} = \langle f, \varphi_{\boldsymbol{\nu}}\rangle_{L^2} \quad \Rightarrow \quad \left\| f - \sum_{\boldsymbol{\nu}\in\Lambda} c_{\boldsymbol{\nu}}\varphi_{\boldsymbol{\nu}} \right\|_{L^2} = \min_{g\in\operatorname{span}\{\varphi_{\boldsymbol{\nu}}\}_{\boldsymbol{\nu}\in\Lambda}} \|f-g\|_{L^2}\,. \tag{2.1}$$

Here, $\{c_{\boldsymbol{\nu}}\}_{\boldsymbol{\nu}\in\Lambda}$ are the optimal expansion coefficients, and $\langle\cdot,\cdot\rangle_{L^2}$ and $\|\cdot\|_{L^2}$ denote the L^2 Hilbert space inner product and norm respectively. In general, the integral coefficients in (2.1) must be evaluated numerically, e.g., with a multidimensional numerical quadrature. Thus, projection methods often come at a high computational cost due to the large number of function samples necessary to approximate $c_{\boldsymbol{\nu}}$ to a sufficient accuracy [14, 206, 210] which can far exceed the number of basis functions. In contrast, interpolation methods require a single sample per basis function, although the resulting approximation is not Hilbert-optimal. The interpolation error in the L^∞ norm is bounded by the best L^∞ approximation error multiplied by a penalty term called the *Lebesgue constant*. However, the degradation in accuracy is usually offset by the reduction in complexity. In particular, sparse interpolation methods [192] often have better overall convergence rate with respect to the number of samples [206].

Given a target function f, one chooses an appropriate sparse-grid method based on (an upper bound of) the decay rates of the orthonormal expansion coefficients $c_{\boldsymbol{\nu}}$, which may be Legendre or Fourier coefficients, for example. If f has an analytic extension, then the expansion coefficients decay as

$$|c_{\boldsymbol{\nu}}| \leq C \exp(-\boldsymbol{\alpha}\cdot\boldsymbol{\nu})$$

where $\boldsymbol{\alpha}$ is related to the size of the region of analyticity, and the bound is asymptotically sharp. So the indices of largest coefficients can be characterized by

$$\Lambda^{\boldsymbol{\alpha}}_{tot}(L) = \{\boldsymbol{\nu}\in\mathbb{N}^d \ : \ \boldsymbol{\alpha}\cdot\boldsymbol{\nu}\leq L\}, \tag{2.2}$$

commonly called a "total-degree space." If f is differentiable and periodic up to a certain finite

[1]This thesis adopts the convention $\mathbb{N} = \{0, 1, 2, \dots\}$.

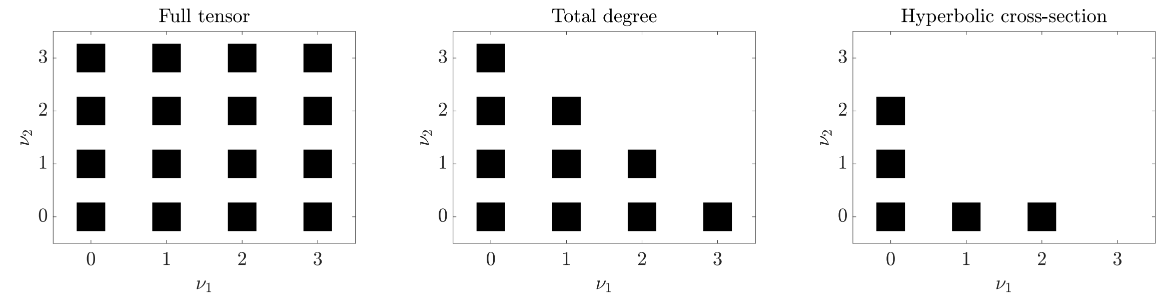


Figure 2.1 Different choices of Λ.

order, then the Fourier coefficients decay as

$$|c_{\boldsymbol{\nu}}| \leq C \prod_{k=1}^{d} (\nu_k + 1)^{-\alpha_k},$$

where α_k is related to the smoothness in dimension k, and the bound is again asymptotically sharp. Similarly to before, the set of indices for the largest coefficients is

$$\Lambda_{hyp}^{\boldsymbol{\alpha}}(L) = \left\{ \boldsymbol{\nu} \in \mathbb{N}^d \ : \ \prod_{k=1}^{d} (\nu_k + 1)^{\alpha_k} \leq L \right\}, \tag{2.3}$$

which is called a "hyperbolic cross-section space."

Armed with these two spaces Λ, we can begin to see why rigid full-tensor grids result in oversampling. In Figure 2.1, the dominant coefficients associated with total-degree and hyperbolic cross-section spaces are in the center and right panels. Full-tensor approaches, in contrast, use far more multi-indices than those associated with the dominant coefficients, leading to function evaluations that do not significantly contribute to the approximation accuracy. Sparse grids use a superposition of smaller full-tensor approximations to match the dominant coefficients of the target function better than a single rigid full-tensor.

To improve the convergence rate of sparse-grid interpolation further, many methods gauge the approximation error in order to determine the most important directions and spatial locations in which to sample next. This process is known as *adaptive refinement*, and the overall goal is to select the samples that would result in the fastest convergence rate. Bungartz and Griebel formulated this procedure as a knapsack problem in which they maximize the added accuracy subject to cost constraints at each refinement iteration until the interpolation error reaches a desired accuracy [34].

In this chapter, we will develop the machinery of sparse grids for interpolation problems. We have deployed both polynomial and trigonometric interpolation bases in the chemistry surrogate model. Accordingly, our initial presentation of sparse grids is basis-independent for generality. In this way, we may discuss the most general construction of a sparse operator free from the details associated with a specific basis or set of coarse full-tensor operators. All that is needed is a set Λ of the multi-indices associated with the dominant coefficients of f. This general approach broadly adheres to Stoyanov's presentation of sparse grids in [200].

Consider the d-dimensional domain $\Gamma = [a_1, b_1] \times \cdots \times [a_d, b_d]$, where a_k and b_k are finite for all $1 \le k \le d$. If $[a_k, b_k]$ is not the canonical interpolation interval (usually $[-1, 1]$ or $[0, 1]$), then we map $[a_k, b_k]$ linearly to the canonical domain. Given a function $f : \Gamma \to \mathbb{R}$, the most general formulation of an interpolation grid (one-dimensional, full-tensor, sparse, or otherwise) requires a set of basis functions $\{\phi_j(\boldsymbol{x})\}_{j=1}^N$, interpolation coefficients $\{c_j\}_{j=1}^N$, and nodes $\{\boldsymbol{x}_j\}_{j=1}^N \subset \Gamma$.[2] The interpolant $\tilde{f}(\boldsymbol{x})$ satisfies the interpolation conditions

$$\tilde{f}(\boldsymbol{x}_j) = f(\boldsymbol{x}_j) \quad \forall\, 1 \le j \le N$$

and may be expressed as

$$\tilde{f}(\boldsymbol{x}) = \sum_{j=1}^{N} c_j \phi_j(\boldsymbol{x})\,. \tag{2.4}$$

Furthermore, we define the interpolation operator as $\mathcal{U}[f] = \tilde{f}$. We will now develop sparse interpolation in detail.

2.2 General construction

We begin by considering the one-dimensional case, $f : [a, b] \to \mathbb{R}$. We use $\mathbb{N}$ to denote the set of natural numbers including zero. Let $m : \mathbb{N} \to \mathbb{Z}_+$ be a strictly increasing function such that $m(l)$ is the number of one-dimensional interpolation nodes on level l.[3] We denote by $\mathcal{U}^{m(l)}$ the interpolation operator associated with the (distinct) level-l nodes $\{x_j^l\}_{j=1}^{m(l)} \subset [a, b]$. That is,

$$\mathcal{U}^{m(l)}[f](x) = \tilde{f}^{(l)}(x) = \sum_{j=1}^{m(l)} c_j^l \phi_j^l(x) \tag{2.5}$$

where c_j^l and $\phi_j^l(x)$ are the interpolation coefficients and basis functions of level l, respectively. If

$$\{x_j^l\}_{j=1}^{m(l)} \subset \{x_j^{l+1}\}_{j=1}^{m(l+1)} \quad \forall l \ge 0,$$

then we say that the one-dimensional rule is *nested*; otherwise, we say it is *non-nested.* For a given choice of basis functions $\phi_j^l(x)$, one may find the interpolation coefficients c_j^l by imposing

[2]The N here has nothing to do with the N from Chapter 1.

[3]We use the concept of *level* to keep track of how nodes are added and when nestedness occurs.

the interpolation conditions

$$\mathcal{U}^{m(l)}[f](x_j^l) = f(x_j^l), \qquad j = 1, 2, \ldots, m(l)\,.$$

One way of gauging the accuracy of interpolation at level l is to look at the difference between levels l and $l-1$. Accordingly, we define the surplus operator

$$\Delta^{m(l)} = \mathcal{U}^{m(l)} - \mathcal{U}^{m(l-1)}, \quad \mathcal{U}^{m(-1)} \equiv 0\,. \tag{2.6}$$

Note that from (2.6), we have the telescoping sum

$$\mathcal{U}^{m(l)} = \Delta^{m(l)} + \Delta^{m(l-1)} + \cdots + \Delta^{m(0)}\,. \tag{2.7}$$

A natural way to represent $\Delta^{m(l)}$ is with so-called hierarchical functions [34]. We denote the level-l hierarchical functions as $\{h_j(x)\}_{j=1}^{m(l)}$, and they have the property that, for each $l \geq 1$,

$$h_j(x_p) = 0 \qquad \forall j \geq m(l) + 1, \quad \forall 1 \leq p \leq m(l)\,. \tag{2.8}$$

Informally, Property (2.8) states that the level-l nodes are zeros of the *additional* hierarchical functions at level $l+1$ and above. Because of the hierarchical property (2.8), we may express $\mathcal{U}^{m(l)}[f]$ as

$$\mathcal{U}^{m(l)}[f](x) = \sum_{j=1}^{m(l)} \check{c}_j h_j(x) \tag{2.9}$$

where the hierarchical interpolation coefficients $\check{c}_j$ do not vary with l. This allows us to write (2.6) as

$$\Delta^{m(l)}[f](x) = \sum_{j=m(l-1)+1}^{m(l)} \check{c}_j h_j(x)\,, \tag{2.10}$$

which will be useful in the context of error estimation later in this chapter. In Example 1, we provide a specific choice of $h_j(x)$ as well as plots to demonstrate how the hierarchical construction is useful for gauging interpolation error.

Example 1. *The most accessible example of hierarchical functions are the piecewise-linear, locally defined hat functions $\{h_j(x)\}_{j=1}^{m(l)}$ on the canonical domain $[-1,1]$. We show these functions in the left column of Fig. 2.2. In order to reduce clutter, we only plot the* ***additional*** *hierarchical functions at each level. The hat function $h_j(x)$ is 0 outside the cone corresponding to the node x_j. From the left column, we see that the hierarchical property* (2.8) *holds. From the right column, we can visually see the contribution to interpolation accuracy of the additional hierarchical functions.*

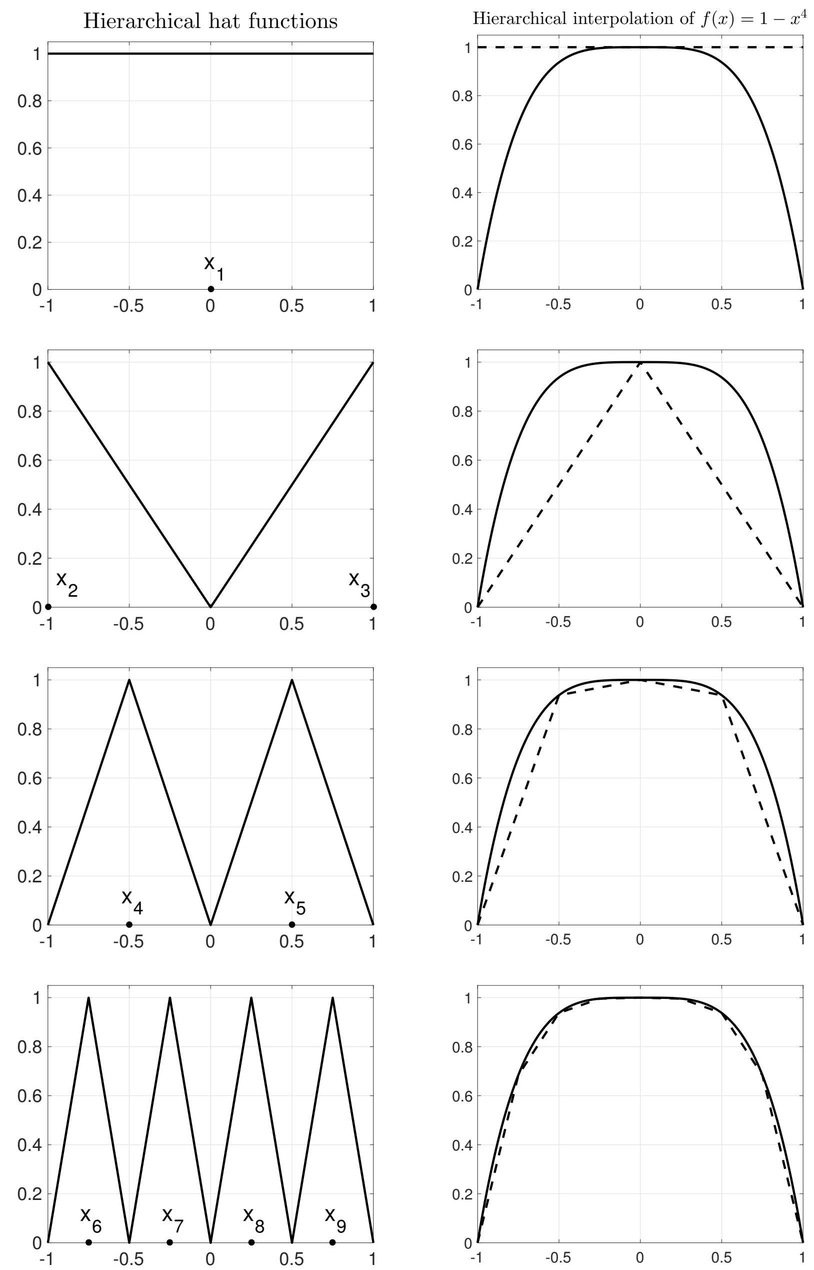


Figure 2.2 Left: Row $i = 0, 1, 2, 3$ shows the additional hierarchical functions of level i. Right: interpolant (dashed line) using the hierarchical functions of level i (i.e., includes all of the basis functions in the left column up to and including row i).

We now consider the multi-dimensional case. Let $f : [a_1, b_1] \times \cdots \times [a_d, b_d] \to \mathbb{R}$. We use the standard multi-index notation:

$$
\begin{aligned}
&\boldsymbol{i} \le \boldsymbol{j} \iff i_k \le j_k, \quad \forall 1 \le k \le d, \\
&\boldsymbol{x}^{\boldsymbol{j}} = \prod_{k=1}^{d} x_k^{j_k}, \text{ with the convention } 0^0 = 1, \\
&\boldsymbol{i} \cdot \boldsymbol{j} = \sum_{k=1}^{d} i_k j_k \,, \\
&\|\boldsymbol{i}\|_1 = \sum_{k=1}^{d} i_k \,, \\
&\|\boldsymbol{i}\|_{\max} = \max_{1 \le k \le d} i_k \,.
\end{aligned}
$$

We now define the d-dimensional tensors of growth functions and nodes

$$
\boldsymbol{m}(\boldsymbol{i}) = [m(i_1), \ldots, m(i_d)], \quad \boldsymbol{x}_{\boldsymbol{j}}^{\boldsymbol{i}} = [x_{j_1}^{i_1}, \ldots, x_{j_d}^{i_d}],
$$

along with the full-tensor interpolation operator

$$
\mathcal{U}^{\boldsymbol{m}(\boldsymbol{i})}[f](\boldsymbol{x}) = \left(\bigotimes_{k=1}^{d} \mathcal{U}^{m(i_k)} \right)[f](\boldsymbol{x}) = \sum_{j_1=1}^{m(i_1)} \cdots \sum_{j_d=1}^{m(i_d)} c_{j_1 \cdots j_d}^{i_1 \ldots i_d} \prod_{k=1}^{d} \phi_{j_k}^{i_k}(x_k) \,. \tag{2.11}
$$

Here, $c_{j_1 \cdots j_d}^{i_1 \cdots i_d}$ are the d-dimensional full-tensor interpolation coefficients, and $\phi_{j_k}^{i_k}$ is evaluated at the k-th component of $\boldsymbol{x}$.

From this point forward, $\boldsymbol{i}$ and $\boldsymbol{j}$ are multi-indices in $\mathbb{N}^d$. We will use $\boldsymbol{i}$ to denote a tensor product of one-dimensional interpolation operators (i.e., a multi-dimensional interpolation level). Following Stoyanov in [200, 206], we will refer to $\boldsymbol{i}$ as a *tensor*.[4] On the other hand, $\boldsymbol{j}$ denotes the indexing of specific points or basis functions, given a particular interpolation tensor $\boldsymbol{i}$. If we break from this convention, we will note it explicitly.

In the full-tensor case, the grid is the Cartesian product of the constituent one-dimensional grids, so we may explicitly write the corresponding interpolation grid $\{\boldsymbol{x}_{\boldsymbol{j}}^{\boldsymbol{i}}\}$ as

$$
\{\boldsymbol{x}_{\boldsymbol{j}}^{\boldsymbol{i}}\}_{\boldsymbol{1} \le \boldsymbol{j} \le \boldsymbol{m}(\boldsymbol{i})} = \{x_{j_1}^{i_1}\}_{j_1=1}^{m(i_1)} \times \cdots \times \{x_{j_d}^{i_d}\}_{j_d=1}^{m(i_d)} \,. \tag{2.12}
$$

Once one chooses the basis functions $\phi_{\boldsymbol{j}}(\boldsymbol{x})$, then the full-tensor interpolation coefficients $c_{\boldsymbol{j}}^{\boldsymbol{i}}$ will

[4] A *tensor* generalizes the notion of interpolation level. The k-th component a tensor is the interpolation level in dimension k.

follow from imposing the interpolation conditions

$$\mathcal{U}^{\boldsymbol{m}(\boldsymbol{i})}[f](\boldsymbol{x}_{\boldsymbol{j}}^{\boldsymbol{i}}) = f(\boldsymbol{x}_{\boldsymbol{j}}^{\boldsymbol{i}}) \qquad \forall \boldsymbol{x}_{\boldsymbol{j}}^{\boldsymbol{i}} \text{ with } \boldsymbol{1} \leq \boldsymbol{j} \leq \boldsymbol{m}(\boldsymbol{i})\,.$$

Lastly, we define d-dimensional surplus operators as

$$\Delta^{\boldsymbol{m}(\boldsymbol{i})} = \bigotimes_{k=1}^{d} \Delta^{m(i_k)} = \bigotimes_{k=1}^{d} \left(\mathcal{U}^{m(i_k)} - \mathcal{U}^{m(i_k-1)}\right), \tag{2.13}$$

where the tensor product of the one-dimensional operators $\mathcal{U}^{m(i_k)}$ is defined in (2.11). Similarly to Equation (2.10), we have

$$\Delta^{\boldsymbol{m}(\boldsymbol{i})}[f](\boldsymbol{x}) = \sum_{\boldsymbol{m}(\boldsymbol{i}-\boldsymbol{1})+\boldsymbol{1} \leq \boldsymbol{j} \leq \boldsymbol{m}(\boldsymbol{i})} \check{c}_{\boldsymbol{j}} h_{\boldsymbol{j}}(\boldsymbol{x})\,, \tag{2.14}$$

where $\check{c}_{\boldsymbol{j}}$ is the full-tensor hierarchical coefficient. The multidimensional hierarchical functions $h_{\boldsymbol{j}}(\boldsymbol{x})$ are the product of the one-dimensional functions:

$$h_{\boldsymbol{j}}(\boldsymbol{x}) = \prod_{k=1}^{d} h_{j_k}(x_k)\,.$$

We note that the number of points required in a full-tensor interpolation scheme is $\prod_{k=1}^{d} m(i_k)$, leading to a large number of interpolation nodes as d grows. When $m(i_k) = N$ for all $1 \leq k \leq d$, then we observe the well-known exponential dependence on the dimension d:

$$\#\{\boldsymbol{x}_{\boldsymbol{j}}^{\boldsymbol{i}}\}_{\boldsymbol{1} \leq \boldsymbol{j} \leq \boldsymbol{m}(\boldsymbol{i})} = N^d\,,$$

where $\#S$ denotes the cardinality of the set S. This phenomenon is known as the *curse of dimensionality*, which sparse grids are able mitigate through a clever selection of full-tensor operators as building blocks.

We define a generic d-dimensional sparse grid operator G_{Θ}^{d} as

$$G_{\Theta}^{d}[f] = \sum_{\boldsymbol{i} \in \Theta} \Delta^{\boldsymbol{m}(\boldsymbol{i})}[f], \tag{2.15}$$

where $\Theta \subset \mathbb{N}^d$ is a lower set.[5] The set Θ may have parameter dependence, which we will express when necessary but omit here for notational simplicity. Given a target function space described by the lower set Λ, we define $\Theta_{opt}(\Lambda)$ as the smallest Θ such that G_{Θ}^{d} is exact for every function in the target space [206]. Because the definition of Θ_{opt} is different for polynomial and

[5]We say $\Lambda \subset \mathbb{N}^d$ is *lower* (or *admissible*) if $\boldsymbol{\nu} \in \Lambda$ implies $\{\boldsymbol{i} \in \mathbb{N}^d : \boldsymbol{i} \leq \boldsymbol{\nu}\} \subset \Lambda$.

trigonometric interpolation, we will define it separately in later sections. Since the full-tensor operators $\mathcal{U}^{\boldsymbol{m}(\boldsymbol{i})}$ are more intuitive than the surplus operators $\Delta^{\boldsymbol{m}(\boldsymbol{i})}$, we wish to rewrite (2.15) in the form

$$G_{\Theta}^{d} = \sum_{\boldsymbol{i}\in\Theta} \Delta^{\boldsymbol{m}(\boldsymbol{i})} = \sum_{\boldsymbol{j}\in\Theta} t_{\boldsymbol{j}} \mathcal{U}^{\boldsymbol{m}(\boldsymbol{j})} \tag{2.16}$$

for some weights $t_{\boldsymbol{j}}$, to be determined. Here, $\boldsymbol{j} \in \Theta$ is an interpolation tensor and not an index of a specific node. Now, from (2.7), we can write each full-tensor operator $\mathcal{U}^{\boldsymbol{m}(\boldsymbol{j})}$ as

$$\begin{aligned}\mathcal{U}^{\boldsymbol{m}(\boldsymbol{j})} &= \bigotimes_{k=1}^{d} \mathcal{U}^{m(j_k)} \\ &= \bigotimes_{k=1}^{d} \left(\Delta^{m(j_k)} + \Delta^{m(j_k-1)} + \cdots + \Delta^{m(0)}\right) \\ &= \sum_{\boldsymbol{\nu}\leq\boldsymbol{j}} \Delta^{\boldsymbol{m}(\boldsymbol{\nu})} \,. \end{aligned} \tag{2.17}$$

By substituting (2.17) into (2.16), we have

$$\sum_{\boldsymbol{i}\in\Theta} \Delta^{\boldsymbol{m}(\boldsymbol{i})} = \sum_{\boldsymbol{j}\in\Theta} t_{\boldsymbol{j}} \sum_{\boldsymbol{\nu}\leq\boldsymbol{j}} \Delta^{\boldsymbol{m}(\boldsymbol{\nu})} \tag{2.18}$$

By equating the coefficients of each $\Delta^{\boldsymbol{m}(\boldsymbol{i})}$ in (2.18), we see that to satisfy (2.16), it must hold that

$$\sum_{\boldsymbol{j}\in\Theta,\ \boldsymbol{i}\leq\boldsymbol{j}} t_{\boldsymbol{j}} = 1 \qquad \forall \boldsymbol{i} \in \Theta \,. \tag{2.19}$$

We may express (2.19) as the linear system $\boldsymbol{M}\boldsymbol{t} = \boldsymbol{1}$. Since $\boldsymbol{M}$ is an upper triangular matrix with a diagonal of all ones, then (2.19) admits a solution, and the reformulation in (2.16) is valid for weights $t_{\boldsymbol{j}}$ satisfying (2.19). Additionally, since the entries of $\boldsymbol{M}$ are zeros and ones, the weights $t_{\boldsymbol{j}}$ are integers. Importantly, the weights $t_{\boldsymbol{j}}$ will depend on the choice of lower set Θ. Rather than using explicit expressions for $t_{\boldsymbol{j}}$ shown in (for example) Theorem 1, the Tasmanian package [200, 203] uses Equation (2.19) to solve for $t_{\boldsymbol{j}}$. From the reformulation (2.16), one can see that G_{Θ}^{d} is a linear combination of the more intuitive full-tensor operators $\mathcal{U}^{\boldsymbol{m}(\boldsymbol{i})}$ defined in Equation (2.11).

Moreover, the grid associated with (2.16) is the union of the constituent full-tensor grids corresponding to $\mathcal{U}^{\boldsymbol{m}(\boldsymbol{i})}$, where $\boldsymbol{i} \in \Theta$. That is, the sparse grid nodes are $\{\boldsymbol{x}_{\boldsymbol{j}}^{\boldsymbol{i}}\}_{(\boldsymbol{i},\boldsymbol{j})\in\bar{X}(\Theta)}$, where

$$\bar{X}(\Theta) = \bigcup_{\boldsymbol{i}\in\Theta} \{(\boldsymbol{i},\boldsymbol{j}) : \boldsymbol{1} \leq \boldsymbol{j} \leq \boldsymbol{m}(\boldsymbol{i})\} \,. \tag{2.20}$$

In the case where the one-dimensional rule is nested, then for tensors $\boldsymbol{i}$ and $\boldsymbol{\nu}$ such that $\boldsymbol{i} \leq \boldsymbol{\nu}$, the full-tensor nodes of tensor $\boldsymbol{i}$ are a subset of the nodes of tensor $\boldsymbol{\nu}$. Thus, for nested rules, we do not need to express explicit dependence on the tensor $\boldsymbol{i}$ and may write the set of sparse nodes as $\{\boldsymbol{x}_{\boldsymbol{j}}\}_{\boldsymbol{j}\in X(\Theta)}$, where

$$X(\Theta) = \bigcup_{\boldsymbol{i}\in\Theta} \{\boldsymbol{j} \in \mathbb{N}^d : \mathbf{1} \leq \boldsymbol{j} \leq \boldsymbol{m}(\boldsymbol{i})\}\,. \tag{2.21}$$

At this point, the sparse operator G^d_Θ in (2.16) can take many forms depending on the interpolation basis $\phi_{\boldsymbol{j}}(\boldsymbol{x})$ and the lower set Θ of admissible interpolation tensors. We will briefly summarize some possible choices for Θ in this section, along with concrete examples in Example 2, and then we will discuss the Lebesgue constant $\mathbb{L}_\Theta$ of sparse interpolation. Afterwards, we will begin new sections to discuss different interpolation bases. Importantly, the approach of much early work on sparse grids started with Θ and then derived the space of exactness Λ, including results on node growth and error estimation. Following recent approaches in sparse grids [201, 205, 206], we will begin with Λ and use $\Theta = \Theta_{opt}$ unless noted otherwise.

For nested rules,[6] it is possible to write full-tensor grids in our Θ notation as

$$\Theta_{\text{full}}(L) = \{\boldsymbol{i} \in \mathbb{N}^d : \|\boldsymbol{i}\|_{\max} \leq L\} \tag{2.22}$$

where $L \in \mathbb{N}$. However, from the tensor-product construction in Equation (2.11), there is no reason that the interpolation level must be the same in each dimension. In order to allow for the possibility that different directions may have different numbers of points, we introduce the weighting vector $\boldsymbol{\xi} \in \mathbb{R}^d$. Now we may define

$$\Theta^{\boldsymbol{\xi}}_{\text{full}}(L) = \{\boldsymbol{i} \in \mathbb{N}^d : \boldsymbol{i} \leq L\boldsymbol{\xi}\} \tag{2.23}$$

where $L\boldsymbol{\xi}$ is the usual componentwise product of a scalar and vector.

The usefulness of introducing $\boldsymbol{\xi}$ is that different dimensions may have different numbers of points, which is useful when the underlying function f is more regular in certain dimensions than in others. In such cases, where not all dimensions have the same number of nodes, the grid is called *anisotropic*; grids that have the same number of points in each dimension are termed *isotropic*. We will present a non-exhaustive summary of popular forms of Θ using the weighting vector $\boldsymbol{\xi}$, with the understanding that the choice $\boldsymbol{\xi} = \mathbf{1}$ produces an isotropic grid. For notational convenience, we will not write a superscript on Θ when $\boldsymbol{\xi} = \mathbf{1}$.

[6]For non-nested rules, using a lower set to describe tensor-product grids is invalid since the points at level $i_k - 1$ are not a subset of those at level i_k. For instance, if the one-dimensional rules are non-nested, then the grid for $\mathcal{U}^{\boldsymbol{m}((3,2))}$ may contain points that $\mathcal{U}^{\boldsymbol{m}((3,3))}$ does *not* contain, even though $(3,2) \leq (3,3)$.

We now introduce the anisotropic total-degree space

$$\Theta_{\mathrm{tot}}^{\boldsymbol{\xi}}(L) = \{\boldsymbol{i} \in \mathbb{N}^d : \boldsymbol{\xi} \cdot \boldsymbol{i} \leq L\}\,. \tag{2.24}$$

Much of the classical work on sparse grids uses (2.24) with $\boldsymbol{\xi} = \mathbf{1}$. Bungartz and Griebel provide a comprehensive overview of the history and development of sparse-grid theory in [34]. In [223], Wasilkowski and Wozniakowski found an explicit representation of the coefficients $t_{\boldsymbol{i}}$. In [155, 156], Novak and Ritter proved results on polynomial exactness and the asymptotic growth of the number of sparse nodes. Barthelmann, Novak, and Ritter derived an explicit error bound on polynomial interpolation using (2.24) in [11]. Nance used (2.24) to construct a sparse polynomial surrogate for potential energy surfaces in the context of molecular dynamics [144, 146, 147].

Another popular choice of tensor-selection strategy is the hyperbolic cross, given by

$$\Theta_{\mathrm{hyp}}^{\boldsymbol{\xi}}(L) = \{\boldsymbol{i} \in \mathbb{N}^d : (\boldsymbol{i} + \mathbf{1})^{\boldsymbol{\xi}} \leq L\}\,. \tag{2.25}$$

Babenko [8] and Korobov [110] noted the hyperbolic cross-section space for its usefulness in approximating multivariable periodic functions with bounded mixed derivatives. Griebel and Hamaekers used a sparse-grid method based on (2.25) to approximate solutions to the many-particle Schrödinger equation. Similarly, Shen and Wang [188] and Shen and Yu [189] used (2.25) to approximate solutions to partial differential equations.

Example 2. *Figure 2.3 displays the admissible $\boldsymbol{i}$ for different choices of $\Theta(L)$, where $L = 3$, along with the corresponding grids. The one-dimensional rule for these grids is nested and uses equidistant nodes; specifically, for $l \in \mathbb{N}$,*

$$m(l) = 3^l, \qquad x_j^l = \frac{j-1}{3^l}, \quad j = 1, 2, \ldots, m(l)\,.$$

By looking at the top-middle panel of Figure 2.3 and temporarily interpreting i_k as the degree of a polynomial, we see that $\Theta_{tot}(3)$ yields the set of all multivariable polynomials of total degree at most 3. The advantage of such a space is that terms such as $x_1^2 x_2^3$ (present in the full-tensor grid) are not relevant if the goal is interpolation of functions of total degree at most 3. We also see that the grid corresponding to $\Theta_{tot}(3)$ uses far fewer points than the tensor-product grid. Compared to $\Theta_{tot}(L)$, $\Theta_{hyp}(L)$ favors tensors containing relatively small mixed terms. However, it is a coincidence that there are no mixed terms in this example; nothing in Equation (2.25) prevents them.

We will now discuss the Lebesgue constant for interpolation. Let $\Lambda \subset \mathbb{N}^d$ be a lower set, and let the generalized interpolation operator G_Θ^d be exact for all f in a space S_Λ. Following

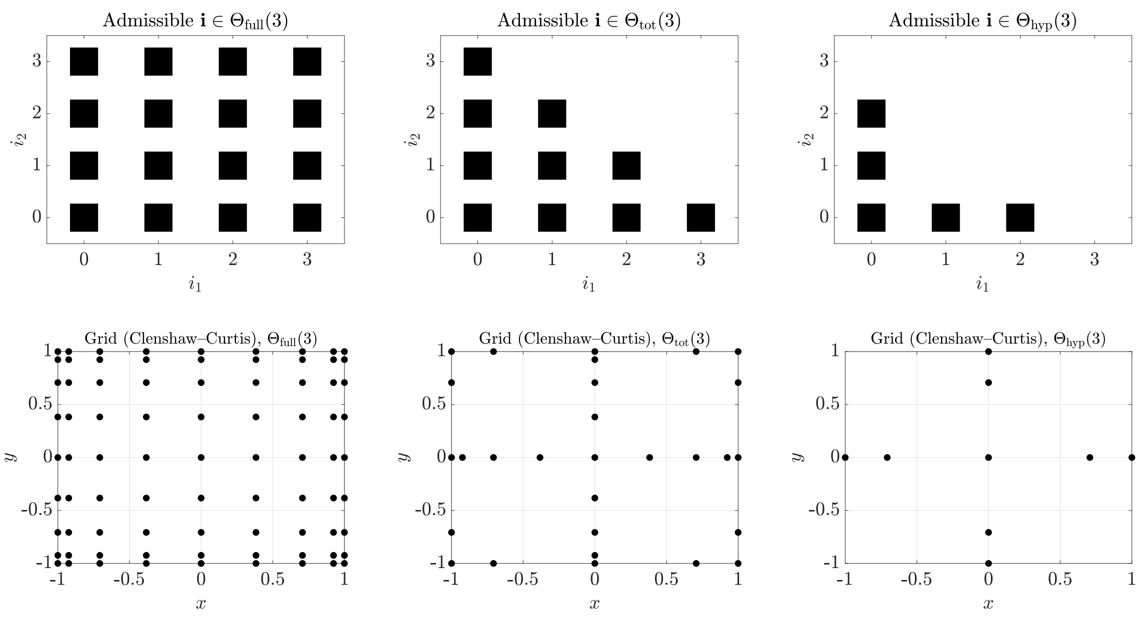


Figure 2.3 Top: Structure of different choices for isotropic $\Theta(L)$, where $L = 3$. Bottom: Grids arising from the different definitions of Θ. The Clenshaw–Curtis points are developed in Section 2.3.1.

classical results in interpolation (e.g., [76]), for all $g \in S_\Lambda$, we have

$$\begin{aligned} \|f - G_\Theta^d[f]\|_\infty &\le \|f - g + g - G_\Theta^d[f]\|_\infty \\ &= \|f - g\|_\infty + \|G_\Theta^d[f - g]\|_\infty \\ &\le \|f - g\|_\infty + \|G_\Theta^d\| \, \|f - g\|_\infty \\ &= (1 + \|G_\Theta^d\|) \, \|f - g\|_\infty \end{aligned}$$

where the operator norm is given by

$$\|G_\Theta^d\|_{\infty,\infty} = \sup_{\|f\|_\infty = 1} \|G_\Theta^d[f]\|_\infty \,. \tag{2.26}$$

Taking the infimum over all $g \in S_\Lambda$, we get

$$\|f - G_\Theta^d[f]\|_\infty \le (1 + \mathbb{L}_\Theta) \inf_{g \in S_\Lambda} \|f - g\|_\infty \tag{2.27}$$

where $\mathbb{L}_\Theta = \|G_\Theta^d\|_{\infty,\infty}$ is the *Lebesgue constant.* Note that the term "constant" is a misnomer in our context since $\mathbb{L}_\Theta$ depends heavily on Θ and the location of the interpolation nodes $\boldsymbol{x_j}$. In general, sharp estimates of the Lebesgue constant for sparse interpolation are not known. However, Chkifa, Cohen, and Schwab showed in [40] that, if the Lebesgue constants of one-dimensional rules λ_l are bounded polynomially by

$$\lambda_l = C_\beta (l+1)^\beta$$

for some $\beta \geq 1$, then

$$\mathbb{L}_\Theta \leq C_\beta^d \, (\#\Theta)^{\beta+1} \tag{2.28}$$

where $\#\Theta$ is the cardinality of the set Θ. That is, if the one-dimensional Lebesgue constants grow polynomially, then the sparse-grid Lebesgue constants also grow polynomially, possibly with a slight increase in the power of the polynomial.

The final result we will present in this section is a closed-form expression for the coefficients $t_{\boldsymbol{i}}$ in Equation (2.16) under the historically popular choice $\Theta = \Theta^{\text{tot}}(L)$. In [223], Wasilkowski and Wozniakowski showed that

$$G^d_{\Theta^{\text{tot}}(L)} = \sum_{\|\boldsymbol{i}\|_1 \leq L} \Delta^{\boldsymbol{m}(\boldsymbol{i})} \tag{2.29}$$

$$= \sum_{\|\boldsymbol{i}\|_1 \leq L} (-1)^{L-\|\boldsymbol{i}\|_1} \binom{d-1}{L-\|\boldsymbol{i}\|_1} \mathcal{U}^{\boldsymbol{m}(\boldsymbol{i})} \,. \tag{2.30}$$

Importantly, we make no assumption here on the choice of basis going into the $\mathcal{U}^{\boldsymbol{m}(\boldsymbol{i})}$. We will now prove the above equality using Wasilkowski and Wozniakowski's approach, which we adapt since multi-indices in our notation are drawn from $\{0,1,2,\dots\}^d$ rather than $\{1,2,\dots\}^d$.

Theorem 1. *Equation* (2.30) *is an equivalent formulation of Equation* (2.29)*.*

Proof. In this proof, we only utilize the fact that $\mathcal{U}^{\boldsymbol{m}(\boldsymbol{i})}$ is a linear operator. Here, $\mathbb{N}$ includes 0. Following Wasilkowski and Wozniakowski in [223], we first define the set

$$P(L,d) = \left\{ \boldsymbol{i} \in \mathbb{N}^d : \|\boldsymbol{i}\|_1 \leq L \right\} .$$

Note that $P(L,d)$ has cardinality $\binom{L+d}{d}$ and contains all of the indices within the sum of Equation (2.29). We now wish to write (2.29) involving a tensor product of only the $\mathcal{U}^{m(i_k)}$. Observe that by multiplying out, we have

$$\bigotimes_{k=1}^{d} \left(\mathcal{U}^{m(i_k)} - \mathcal{U}^{m(i_k-1)}\right) = \sum_{\boldsymbol{\alpha} \in \{0,1\}^d} (-1)^{\|\boldsymbol{\alpha}\|_1} \bigotimes_{k=1}^{d} \mathcal{U}^{m(i_k-\alpha_k)} \,. \tag{2.31}$$

So, given a multi-index $\boldsymbol{j} \in \mathbb{N}^d$, $\bigotimes_{k=1}^d \mathcal{U}^{m(j_k)}$ appears in Equation (2.29) for each $\boldsymbol{i} \in \mathbb{N}^d$ such that $\boldsymbol{j} = \boldsymbol{i} - \boldsymbol{\alpha}$, where $\boldsymbol{\alpha} \in \{0,1\}^d$ and $\|\boldsymbol{\alpha}\|_1 \leq L - \|\boldsymbol{j}\|_1$. From (2.31), the sign of $\bigotimes_{k=1}^d \mathcal{U}^{m(j_k)}$

is $(-1)^{\|\boldsymbol{\alpha}\|_1}$.

We define

$$b(z,d) = \sum_{\boldsymbol{\alpha}\in\{0,1\}^d,\ \|\boldsymbol{\alpha}\|_1 \le z} (-1)^{\|\boldsymbol{\alpha}\|_1},$$

which, combined with Equation (2.29) and the preceding paragraph, yields

$$G_\Theta^d = \sum_{\boldsymbol{j}\in P(L,d)} b(L - \|\boldsymbol{j}\|_1, d) \bigotimes_{k=1}^{d} \mathcal{U}^{m(j_k)}. \tag{2.32}$$

To compute $b(i,d)$, we sum with respect to $\|\boldsymbol{\alpha}\|_1 = 0,1,\ldots,d$. For $i \ge d$, we have

$$b(i,d) = \sum_{j=0}^{d} (-1)^j \binom{d}{j} = 0,$$

and for $i \le d-1$, we have

$$b(i,d) = \sum_{j=0}^{i} (-1)^j \binom{d}{j} = (-1)^i \binom{d-1}{i}.$$

By setting $i = L - \|\boldsymbol{j}\|_1$, then Equation (2.32) becomes

$$G_{\Theta(L)}^d = \sum_{\|\boldsymbol{i}\|_1 \le L} (-1)^{L-\|\boldsymbol{i}\|_1} \binom{d-1}{L-\|\boldsymbol{i}\|_1} \mathcal{U}^{\boldsymbol{m}(\boldsymbol{i})}$$

as required. □

2.3 Polynomial basis

In this section, we will summarize the basic theory of polynomial interpolation with a Lagrange basis. In terms of the general sparse grid construction, this is one specific choice of basis functions $\phi_j(x)$ and corresponding interpolation coefficients c_j. After developing the one-dimensional and full-tensor interpolation operators, we will present the classical Smolyak construction of a sparse operator, first developed in [192]. Other than being the original (and thoroughly studied) sparse operator, the Smolyak construction is merely one choice for the lower set Θ of admissible tensors.

2.3.1 One dimension and full-tensor extension

In terms of Equation (2.5), this section develops one particular choice of basis functions $\phi_j(x)$ and interpolation nodes c_j for the polynomial case. Polynomial interpolation in one dimension is a classical staple in Numerical Analysis texts, e.g. [76]. Given n distinct nodes $\{x_j\}_{j=1}^n \subset [a,b]$

and associated function values $\{f(x_j)\}_{j=1}^n$, we first introduce the n-point interpolation operator $\mathcal{U}^n : C([a,b]) \to \mathbb{P}_{n-1}$. Here, $\mathbb{P}_k$ is the space of all univariate degree-k polynomials. Our goal is to construct a polynomial $\mathcal{U}^n[f] \in \mathbb{P}_{n-1}$ such that

$$\mathcal{U}^n[f](x_j) = f(x_j), \quad j = 1, 2, \ldots, n\,.$$

Here, we use the notation $\mathcal{U}^n$ to maintain consistency with $\mathcal{U}^{m(l)}$ from Section 2.2; that is, we take $m(l) = n$. Note that it is not necessary to know, *a priori*, the exact form of f, in which case we may view $\{f(x_j)\}_{j=1}^n$ as data. In order to construct $\mathcal{U}^n[f](x)$, we introduce the elementary Lagrange polynomials of degree $n-1$ (associated with an n-point grid):

$$\ell_j^n(x) = \prod_{r=1,\ r\neq j}^{n} \frac{x - x_r}{x_j - x_r}, \quad j = 1, 2, \ldots, n\,.$$

Observe that $\ell_i^n(x_j) = \delta_{ij}$. We define $\mathcal{U}^n[f] : [a,b] \to \mathbb{R}$ as

$$\mathcal{U}^n[f](x) = \sum_{j=1}^{n} f(x_j)\ell_j^n(x),$$

which gives $\mathcal{U}^n[f](x_j) = f(x_j)$ for all $1 \leq j \leq n$. Furthermore, one can easily show that the interpolating polynomial $\mathcal{U}^n[f]$ is unique [76]. This equation is nothing more than (2.5) with the choice $c_j = f(x_j)$ and $\phi_j^l(x) = \ell_j^n(x)$.

In order to select a one-dimensional rule for the nodes $\{x_j\}_{j=1}^n$, we want our choice of nodes to yield a reasonable bound on our interpolation error. Accordingly, we will now devote some time toward error estimation for one-dimensional polynomial interpolation. Gautschi shows in [76] that

$$\|\mathcal{U}^n\| = \mathbb{L}_n = \|\lambda_n\|_\infty, \qquad \lambda_n(x) = \sum_{j=1}^{n} \left|\ell_j^n(x)\right|,$$

where $\mathbb{L}_n$ and $\lambda_n(x)$ are known as the *Lebesgue constant* and *Lebesgue function*, respectively. The operator norm is given by (2.26). Following [76], we can derive from $\mathbb{L}_n$ the rough error estimate

$$\|f - \mathcal{U}^n[f]\|_\infty \leq (1 + \mathbb{L}_n) \min_{p\in\mathbb{P}_{n-1}} \|f - p\|_\infty\,. \tag{2.33}$$

So we see that the worst-case error depends on the Lebesgue constant $\mathbb{L}_n$ (which in turn depends on the choice of nodes), as well as the best approximation of f in the L^∞ norm by polynomials of degree $n-1$. Unfortunately, as $n \to \infty$, $\mathbb{L}_n \to \infty$ as well; however, some choices of nodes cause

$\mathbb{L}_n$ to grow more slowly. Furthermore, Jackson's Theorem [7] states that, for $f \in C^k([a,b])$,

$$\min_{p \in \mathbb{P}_{n-1}} \|f - p\|_\infty \leq \frac{\alpha_{k,f}\|f\|_\infty}{n^k}, \tag{2.34}$$

where $\alpha_{k,f}$ is a constant that depends on k and f. Thus, interpolation error decreases as f becomes smoother. We now repeat a well-known one-dimensional pointwise error estimate in the following theorem, highlighting the impact of node choice on error estimation.

Theorem 2. *Let $f \in C^n([a,b])$ and $\{x_j\}_{j=1}^n \subset [a,b]$ be a set of n distinct nodes. For each $t \in [a,b]$, there exists some ξ_t such that*

$$f(t) - \mathcal{U}^n[f](t) = \frac{f^{(n)}(\xi_t)}{n!} \prod_{j=1}^n (t - x_j)\,. \tag{2.35}$$

Proof. This is a classical proof in many Numerical Analysis textbooks, e.g. [76]. For $t = x_j$, the result holds trivially. Let $t \in [a,b]$ be fixed but arbitrary, such that for all $j \in \{1, \dots, n\}$, $t \neq x_j$. Define the functions

$$\begin{aligned} E(x) &= f(x) - \mathcal{U}^n[f](x), \\ \Psi(x) &= \prod_{j=1}^n (x - x_j), \\ G(x) &= E(x) - \frac{\Psi(x)}{\Psi(t)} E(t) \end{aligned}$$

where $x \in [a,b]$. Since $f \in C^n([a,b])$ and $\mathcal{U}^n[f], \Psi \in \mathbb{P}_{n-1}$, then G is n times continuously differentiable. Note that

$$\begin{aligned} G(x_j) &= 0, \qquad j = 1, \dots, n, \\ G(t) &= 0\,. \end{aligned}$$

So G has $n+1$ distinct roots in (a,b). Applying Rolle's theorem once to G, we see that G' has n distinct roots in (a,b). Applying Rolle's theorem to G', we see that $G^{(2)}$ has $n-1$ distinct roots in (a,b). Continuing along these lines, when we apply Rolle's theorem n times to G, we see that there exists some $\xi_t \in (a,b)$ such that $G^{(n)}(\xi_t) = 0$. Differentiating G with respect to x, we have

$$G^{(n)}(x) = f^{(n)}(x) - \frac{n!}{\Psi(t)} E(t)\,.$$

Since $G^{(n)}(\xi_t) = 0$, then

$$\frac{f^{(n)}(\xi_t)}{n!} \prod_{j=1}^{n} (x - x_j) = E(t) = f(t) - \mathcal{U}^n[f](t),$$

as required. $\square$

From (2.35), one can see that the choice of nodes $\{x_j\}_{j=1}^{n}$ is critically important for small interpolation error. We wish to choose the nodes in such a way that the Lebesgue constant grows slowly and the points are nested. Compared to the non-nested case, nestedness results in fewer required evaluations of f as we increase the interpolation level l. Following [11, 147], we will use the Clenshaw–Curtis points, which are the extrema of the Chebyshev polynomials and a popular choice for polynomial interpolation.[7] For a level $l \in \mathbb{N}$, we have $m(l)$ points $\{x_j^l\}_{j=1}^{m(l)}$, where

$$m(l) = \begin{cases} 1, & l = 0 \\ 2^l + 1, & l > 0 \end{cases}, \tag{2.36}$$

$$x_j^l = \begin{cases} 0, & l = 0 \\ -\cos\left(\frac{j-1}{m(l)-1}\pi\right), \quad 1 \le j \le m(l), & l > 0 \end{cases}. \tag{2.37}$$

In Fig. 2.4, we show the one-dimensional Clenshaw–Curtis points for different values of l.

As stated previously, a primary goal when choosing nodes is that our choice yields respectable error estimates. From [64, 65, 226], we indeed see that the Lebesgue constant for Clenshaw–Curtis points grows, like the Chebyshev nodes (roots of the Chebyshev polynomials), at most logarithmically in the number of points:

$$\mathbb{L}_{m(l)} \le \frac{2}{\pi}\log(m(l) - 1) + 1 = \frac{2}{\pi}\log(2^l) + 1, \quad l \ge 1\,. \tag{2.38}$$

From (2.38), linear growth of the Lebesgue constant with respect to l follows immediately, so (2.28) holds with $\beta = 1$.

Extending one-dimensional interpolation for $d > 1$, we let $f : [a_1, b_1] \times \cdots \times [a_d, b_d] \to \mathbb{R}$. We use the typical multi-index notation from Section 2.2. We define the multivariate interpolation

[7]For a detailed overview of different choices for one-dimensional polynomial interpolation nodes, see [200].

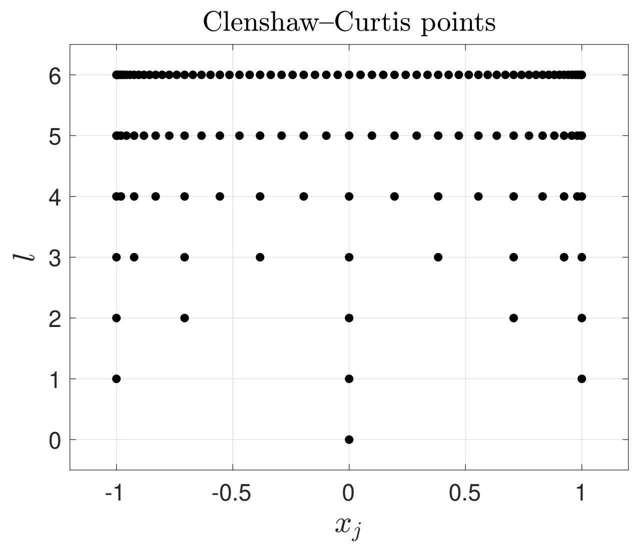


Figure 2.4 Examples of Clenshaw–Curtis points for different levels l. Notice the nestedness of the nodes and how they tend to cluster around the boundary.

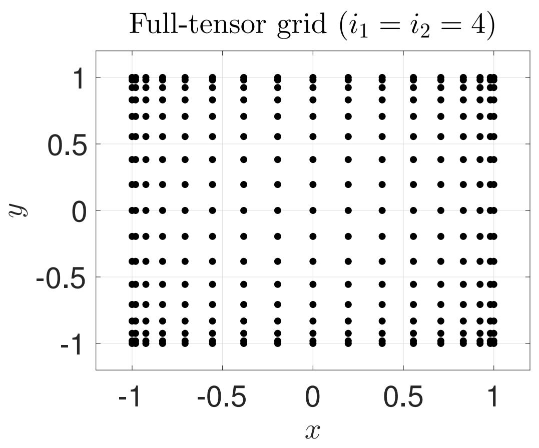
operator $\mathcal{U}^{\boldsymbol{m}(\boldsymbol{i})}$ as the tensor product of the one-dimensional interpolation operators $\mathcal{U}^{m(i_k)}$:

$$\begin{aligned}\mathcal{U}^{\boldsymbol{m}(\boldsymbol{i})}[f](\boldsymbol{x}) &= \left(\bigotimes_{k=1}^{d} \mathcal{U}^{m(i_k)}\right)[f](\boldsymbol{x}) \\ &= \sum_{j_1=1}^{m(i_1)} \cdots \sum_{j_d=1}^{m(i_d)} f\left(x_{j_1}^{i_1}, \ldots, x_{j_d}^{i_d}\right) \prod_{k=1}^{d} \ell_{j_k}^{i_k}(x_k)\,. \end{aligned} \tag{2.39}$$

Here, $\mathcal{U}^{m(i_k)}$ uses a Clenshaw–Curtis grid of level i_k, and the multivariate elementary Lagrange polynomials are the products of the corresponding one-dimensional polynomials. The set of full-tensor nodes $\{\boldsymbol{x}_{\boldsymbol{j}}^{\boldsymbol{i}}\}_{\boldsymbol{1}\le\boldsymbol{j}\le\boldsymbol{m}(\boldsymbol{i})}$ needed in (2.39) is given by

$$\{\boldsymbol{x}_{\boldsymbol{j}}^{\boldsymbol{i}}\}_{\boldsymbol{1}\le\boldsymbol{j}\le\boldsymbol{m}(\boldsymbol{i})} = \{x_{j_1}^{i_1}\}_{j_1=1}^{m(i_1)} \times \cdots \times \{x_{j_d}^{i_d}\}_{j_d=1}^{m(i_d)} \tag{2.40}$$

where $\{x_{j_k}^{i_k}\}$ is defined in (2.37). Fig. 2.5 shows an example of isotropic and anisotropic full-tensor grids for $d = 2$. As noted in Section 2.2, Equation (2.39) requires a number of nodes that grows exponentially in the dimension. To interpolate a d-variate polynomial of degree $N-1$ in the full-tensor construction, one needs N^d points. For high-dimensional problems, the full-tensor construction is infeasible due to the large number of points required, but smaller full-tensor operators serve as the building blocks of sparse grids, which we consider in the next subsection.



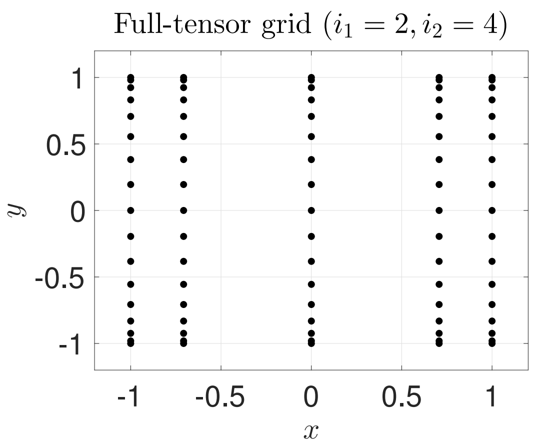


Figure 2.5 Examples of tensor-product Clenshaw–Curtis grids for the $d = 2$ case. Left: $\boldsymbol{i} = (4,4)$ (isotropic). Right: $\boldsymbol{i} = (2,4)$ (anisotropic).

2.3.2 Sparse polynomial interpolation

Let $f : \mathbb{R}^d \to \mathbb{R}$ be given. Smolyak gave his original construction of a sparse operator in [192] as

$$G^d_{\Theta(L)}[f] = \sum_{\|\boldsymbol{i}\|_1 \le L} \Delta^{\boldsymbol{m}(\boldsymbol{i})}[f], \tag{2.41}$$

which is the Smolyak operator in Equation (2.29) of Section 2.2. Here, $\Delta^{\boldsymbol{m}(\boldsymbol{i})}$ is defined by Equation (2.13), $\Theta(L)$ is the total-degree space given by (2.24), and the basis functions are Lagrange polynomials with the Clenshaw–Curtis points (2.37). Note that this operator is merely a specific instance of the general sparse operator in Equation (2.15) for the Clenshaw–Curtis polynomial basis and the Smolyak tensor space. For notational convenience, we suppress the subscripting on $\Theta(L)$ for this section. For now, we treat L simply as a parameter; we will later show that L is the polynomial degree of exactness. For this choice of $\Theta(L)$, we showed in Theorem 1 of Section 2.2 that $G^d_{\Theta(L)}$ has the equivalent formulation

$$G^d_{\Theta(L)}[f] = \sum_{\|\boldsymbol{i}\|_1 \le L} (-1)^{L - \|\boldsymbol{i}\|_1} \binom{d-1}{L - \|\boldsymbol{i}\|_1} \mathcal{U}^{\boldsymbol{m}(\boldsymbol{i})}[f]\,, \tag{2.42}$$

where $\mathcal{U}^{\boldsymbol{m}(\boldsymbol{i})}$ is defined by (2.39).

We have the set of sparse nodes (i.e., the *sparse grid*)

$$\mathcal{H}_{\Theta(L)} = \bigcup_{\|\boldsymbol{i}\|_1 \le L} \{\boldsymbol{x}^{\boldsymbol{i}}_{\boldsymbol{j}}\}_{\boldsymbol{1} \le \boldsymbol{j} \le \boldsymbol{m}(\boldsymbol{i})}\,, \tag{2.43}$$

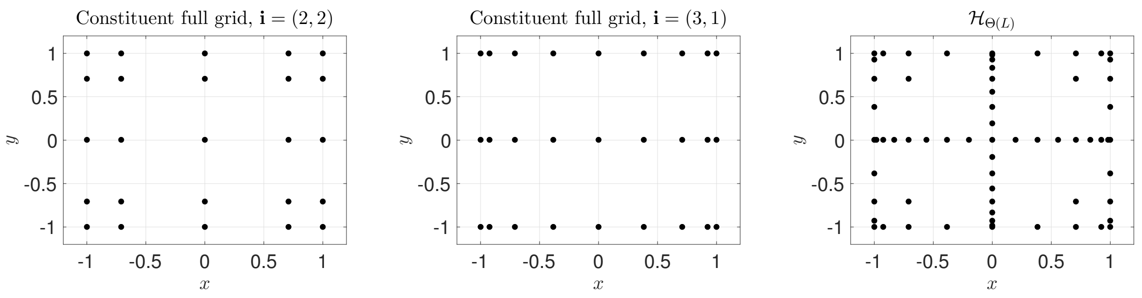


Figure 2.6 Left, middle: constituent full-tensor grids. Right: sparse grid for $d = 2$, $L = 4$. Note that the sparse grid contains the left and middle full grids.

where $\{\boldsymbol{x}_{\boldsymbol{j}}^{\boldsymbol{i}}\}_{\mathbf{1}\le \boldsymbol{j} \le \boldsymbol{m}(\boldsymbol{i})}$ is defined by Equation (2.40). Equation (2.43) is merely (2.21) for a particular choice of interpolation basis and $\Theta(L)$. We have the useful property that, for a given dimension d, $\mathcal{H}_{\Theta(L+1)} \subset \mathcal{H}_{\Theta(L)}$ since the one-dimensional rule is nested. We illustrate the node selection procedure for Equation (2.43) in the following example.

Example 3. *Let $d = 2$ and $L = 4$. We will show how to construct the sparse grid $\mathcal{H}_{\Theta(L)}$ from the constituent full-tensor grids. From Equation (2.24), we calculate*

$$\begin{aligned}\Theta_{tot}(L) = \{&(0,0),(0,1),(0,2),(0,3),(0,4),\\ &(1,0),(1,1),(1,2),(1,3),\\ &(2,0),(2,1),(2,2),\\ &(3,0),(3,1),\\ &(4,0)\}\,.\end{aligned}$$

We focus on the full-tensor grids grids corresponding to $\boldsymbol{i} = (2, 2), (3, 1)$. We calculate $\{\boldsymbol{x}_{\boldsymbol{j}}^{\boldsymbol{i}}\}$ for these choices of $\boldsymbol{i}$ according to Equation (2.40). Moreover, the sparse grid $\mathcal{H}_{\Theta(L)}$ is the set of all nodes corresponding to the full-tensor operators in $\Theta_{\mathrm{Smol}}(L)$.

Fig. 2.6 shows the full-tensor grids for $\boldsymbol{i} = (2, 2)$ and $\boldsymbol{i} = (3, 1)$, and the set of sparse points $\mathcal{H}_{\Theta(L)}$. Note that these full-tensor grids are subsets of $\mathcal{H}_{\Theta(L)}$.

Next, we turn our attention to the growth of the number of points $n(L, d)$ that Equation (2.42) requires. We use the notation

$$a_n \approx b_n \quad \Leftrightarrow \quad \lim_{n\to\infty} \frac{a_n}{b_n} = 1\,. \tag{2.44}$$

Novak and Ritter [156] showed that in the context of quadrature formulas, for fixed L, $n(L, d)$ grows polynomially in d. In [147], Nance showed a similar result for sparse polynomial interpolation. We state the specific result in the following theorem.

Theorem 3. *Let $n(L,d)$ be the number of points required by Smolyak's algorithm in Equation (2.42). Then, for fixed L and as $d \to \infty$,*

$$n(L,d) \approx \frac{2^L}{L!}\, d^L, \tag{2.45}$$

where the relation $\approx$ is defined by (2.44).

Proof. See Nance [147] and Novak and Ritter [156]. We present a similar proof similar proof in the context of trigonometric interpolation in Theorem 7 of Section 2.4.2. □

2.3.3 Exactness and error estimation

We begin by presenting a result on polynomial exactness, based on the work of Nance [147] and Novak and Ritter [155].

Theorem 4. $G^d_{\Theta(L)}[p] = p$ *for all p in the space*

$$\sum_{\|\boldsymbol{i}\|_1 = L} \left(\mathbb{P}_{m(i_1)-1} \otimes \cdots \otimes \mathbb{P}_{m(i_d)-1} \right) . \tag{2.46}$$

Proof. See Nance [147], based on a similar result from Novak and Ritter [155]. Again, we present quite a similar proof in the context of trigonometric interpolation in Theorem 8 of Section 2.4.3. □

From Equation (2.46), we have, via a counting argument, that L is the polynomial degree of exactness. That is, L is the largest integer n such that, for all $d \geq 1$,

$$G^d_{\Theta(L)}[p] = p \qquad \text{for all } d\text{-variate } p \text{ of degree } n\,.$$

To see this, take $d > L$ in Equation (2.46). First, note that every d-dimensional polynomial of degree L is in (2.46). However, $x_1 \cdots x_L x_{L+1}$ is a polynomial of degree $L+1$ that is not in (2.46), so L is the degree of exactness.

To develop a general error estimate for sparse polynomial interpolation, we first introduce the notation

$$D^{\boldsymbol{k}} f = \frac{\partial^{\|\boldsymbol{k}\|_1} f}{\partial x_1^{k_1} \cdots \partial x_d^{k_d}}, \qquad \boldsymbol{k} \in \mathbb{N}_0^d\,.$$

From Equations (2.33), (2.34), and (2.38), we get the following one-dimensional error estimate for $f \in C^r([-1,1])$:

$$\|f - \mathcal{U}^{m(l)}[f]\|_\infty \leq \frac{\alpha_{r,f} \log(m(l))}{m(l)^r} \|f\|_\infty \tag{2.47}$$

where $\mathcal{U}^{m(l)}$ is the one-dimensional polynomial interpolation operator on level l.

Now we define the more general space

$$F_d^r = \Big\{ f : [-1,1]^d \to \mathbb{R} \ : \ D^{\boldsymbol{k}} f \text{ is continuous } \forall \boldsymbol{k} \in \mathbb{N}_0^d \text{ with } \boldsymbol{k} \leq r \cdot \mathbf{1} \Big\} \tag{2.48}$$

with the norm

$$\|f\|_{F_d^r} = \max\{\|D^{\boldsymbol{k}} f\|_\infty \ : \ \boldsymbol{k} \in N_0^d, \ \boldsymbol{k} \leq r \cdot \mathbf{1}\}\,.$$

Following Barthelmann, Novak, and Ritter in [11], we let I_d denote the embedding

$$(F_d^r, \|\cdot\|_{F_d^r}) \hookrightarrow (C([-1,1]^d), \|\cdot\|_\infty)$$

with the operator norm

$$\|I_d\|_{F_d^r,\infty} = \sup\{\|f\|_\infty \ : \ f \in F_d^r, \ \|f\|_{F_d^r} = 1\}\,.$$

From (2.47), we have the error estimate

$$\|I_1 - \mathcal{U}^{m(l)}\|_{F_d^r,\infty} \leq c_{r,1} \cdot \log(m(l)) \cdot m(l)^{-r} \tag{2.49}$$

where $c_{r,d}$ denotes constants that depend only on r and d. Barthelmann et al. [11] used Equation (2.49) to prove a d-dimensional error bound for $G^d_{\Theta(L)}$, which we state in the following theorem.

Theorem 5. *Let $n = n(L,d)$ be the number of points required by Smolyak's algorithm in* (2.42)*. Then, taking the space F_d^r in Equation* (2.48) *as the domain of the operators I_d and $G^d_{\Theta(L)}$, we have*

$$\|I_d - G^d_{\Theta(L)}\|_{F_d^r,\infty} \leq c_{r,d} \cdot n^{-r} \cdot (\log n)^{(d-1)(r+2)+1} \tag{2.50}$$

Proof. See Barthelmann, Novak, and Ritter in [11]. We prove an error estimate for sparse trigonometric interpolation in Theorem 9 of Section 2.4.4. □

Combining the results on exactness and polynomial growth of $n(L,d)$, we compare the number of points required to exactly interpolate a d-variate polynomial of degree 4 with both tensor-product grids and sparse grids. Table 2.1 displays the results. We note that for large d, Smolyak's algorithm requires dramatically fewer points even though the polynomial exactness is the same.

Finally, let $\Lambda \subset \mathbb{N}^d$ be lower and $m(l)$ be the growth function of the interpolation rule. Consider the target function space

$$\mathbb{P}_\Lambda = \bigcup_{\boldsymbol{\nu} \in \Lambda} \bigotimes_{k=1}^{d} \mathbb{P}_{\nu_k}\,.$$

Table 2.1 Growth of number of points for Equation (2.39) and for Equation (2.42), using the Clenshaw–Curtis points to build $G^d_{\Theta(L)}$. Both methods are exact on polynomials of degree 4.

d	Full-tensor ($n = 5$)	Sparse C-C grid ($L = 4$)
1	5	17
2	25	65
3	125	177
5	3,125	801
10	9,765,625	8,801
15	30,517,578,125	40,001

Stoyanov and Webster [206] showed that the smallest Θ such that G^d_Θ is exact for all $f \in \mathbb{P}_\Lambda$ is

$$\Theta^{opt}_{poly} = \{\boldsymbol{i} \in \mathbb{N}^d \ : \ \boldsymbol{m}(\boldsymbol{i} - \boldsymbol{1}) \in \Lambda\}\,. \tag{2.51}$$

2.4 Trigonometric basis

We now turn our attention to trigonometric interpolation. Our application requires a periodic interpolation basis in the angular components of molecular geometry in order to enforce periodicity of the surrogate potential energy surface (PES). If we lose periodicity, we will fail to conserve the total energy of our system, and our results would therefore be unreliable.

Much of the literature on sparse interpolation of periodic functions actually predates the major works on sparse polynomial interpolation for non-periodic functions. In 1989, Baszenski and Delvos [12] and Delvos and Schempp [51] studied Boolean sums of trigonometric operators on Korobov function spaces. In 1992, Hallatschek described a high-dimensional fast Fourier transform based on the sparse-grid approach [84]. Hallatschek's work also includes an interpolation scheme based on the sparse FFT as well as interpolation error estimates. Pöplau and Sprengel described error estimates for periodic interpolation on both full and sparse grids in 1994 [171]. Later, in 1998, Sprengel developed a sparse-grid approach to wavelet approximation [195]. In 2000, she presented a unified approach to the estimation of periodic interpolation error on full and sparse grids, for functions from Sobolev spaces, Korobov spaces, and the space of functions with absolutely convergent Fourier coefficients [196]. In 2014, Griebel and Hamaekers generalized Hallatschek's work by parametrizing the structure of a sparse grid and deriving error bounds associated with the parameter [81]. In 2020, the author and Stoyanov implemented a nested trigonometric interpolation rule in the Tasmanian sparse grid package, as well as a novel dimensionally adaptive refinement algorithm in the context of sparse trigonometric interpolation [140].

2.4.1 One dimension and full-tensor extension

Our approach is similar that of Hallatschek [84] and Griebel and Hamaekers [81]. Without loss of generality, we consider periodic functions $f : [0, 1] \to \mathbb{R}$. We first define the complex exponential functions $\varphi_j(x)$ as [8]

$$\varphi_j(x) = \exp\left(2\pi \mathrm{i} \cdot \sigma(j) \cdot x\right), \tag{2.52}$$

where $\mathrm{i}^2 = -1$ and

$$\sigma(j) = \begin{cases} (1-j)/2, & j \text{ odd} \\ j/2, & j \text{ even} \end{cases}. \tag{2.53}$$

We introduce $\sigma(j)$ because trigonometric interpolation involves a sum over both positive and negative powers of $\mathrm{e}^{2\pi \mathrm{i} x}$. Furthermore, the one-dimensional equidistant interpolation nodes are

$$x_j^l = \frac{j-1}{m(l)}, \quad j = 1, \ldots, m(l), \tag{2.54}$$

$$m(l) = 3^l. \tag{2.55}$$

Fig. 2.7 displays the periodic interpolation points for different l. We note that our grids are nested with respect to l.

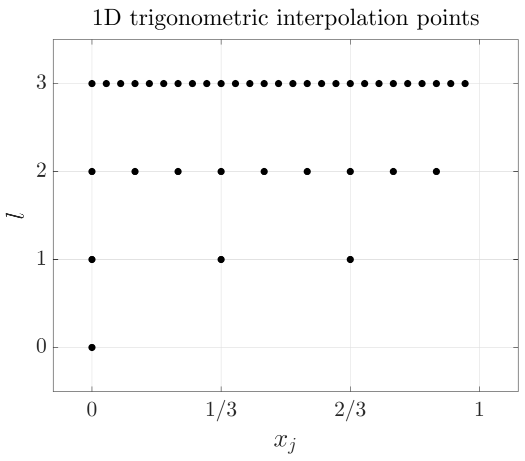


Figure 2.7 Trigonometric interpolation points for $l = 0, \ldots, 3$.

In order to find the interpolation coefficients c_j^l in Equation (2.5), we temporarily interpret $\varphi_j^l(x)$ as our basis functions. Note that $\varphi_j^l(x)$ is complex-valued, so we must take real parts at

[8]To distinguish $\mathrm{i} = \sqrt{-1}$ from an index i, we use the Roman typeface.

the end. The $m(l)$ interpolation conditions are

$$f(x_j^l) = \tilde{\mathcal{U}}^{m(l)}[f](x_j^l) \qquad j = 1, 2, \ldots, m(l), \tag{2.56}$$

where $\tilde{\mathcal{U}}^{m(l)}$ is the (complex-valued) trigonometric interpolation operator on level l. In [81, 84], we find that the (again, complex-valued) interpolant satisfying (2.56) has the form

$$\tilde{\mathcal{U}}^{m(l)}[f](x) = \sum_{j=1}^{m(l)} \hat{f}_j^l \varphi_j(x)\,, \tag{2.57}$$

where the normalized discrete Fourier coefficients $\hat{f}_j^l$ are given by

$$\hat{f}_j^l = \frac{1}{m(l)} \sum_{p=1}^{m(l)} f(x_p^l)\varphi_j^*(x_p^l)\,. \tag{2.58}$$

Here, φ_j^* denotes the complex conjugate of φ_j. We take the real part of (2.57) to obtain the real-valued interpolant

$$\mathcal{U}^{m(l)}[f](x) = \sum_{j=1}^{m(l)} \mathrm{Re}(\hat{f}_j^l)\mathrm{Re}(\varphi_j(x)) - \mathrm{Im}(\hat{f}_j^l)\mathrm{Im}(\varphi_j(x))\,. \tag{2.59}$$

In one dimension, we can furthermore write (2.59) as

$$\mathcal{U}^{m(l)}[f](x) = \sum_{j=1}^{m(l)} \mathrm{Re}(\hat{f}_j^l)\cos(2\pi\,\sigma(j)\,x) - \mathrm{Im}(\hat{f}_j^l)\sin(2\pi\,\sigma(j)\,x)\,. \tag{2.60}$$

By examining (2.60) and $\sigma(j)$, we see that, in terms of Equation (2.5), the $2m(l)$ basis functions and interpolation coefficients are

$$\begin{aligned} \phi_j^l(x) &= \cos(2\pi\,\sigma(j)\,x), \quad c_j^l = \mathrm{Re}(\hat{f}_j^l), \quad j = 1, 2, \ldots, m(l), \\ \phi_j^l(x) &= \sin(2\pi\,\sigma(j)\,x), \quad c_j^l = -\mathrm{Im}(\hat{f}_j^l), \quad j = m(l)+1, \ldots, 2m(l)\,. \end{aligned}$$

It is an established result that trigonometric interpolation with $2n+1$ points can resolve all modes up to frequency n [95]. Since we have $m(l) = 3^l$ points, then $\mathcal{U}^{m(l)}[u] = u$ for all $u \in \mathbb{T}_{(m(l)-1)/2}$, where

$$\mathbb{T}_n = \mathrm{span}\,\{1, \cos(2\pi x), \ldots, \cos(2\pi n x), \sin(2\pi x), \ldots, \sin(2\pi n x)\}\,. \tag{2.61}$$

We will use $\mathbb{T}_n$ in an analogous manner to $\mathbb{P}_n$ from the previous section; specifically, we refer to $\mathbb{T}_n$ as the space of all trigonometric functions of degree at most n. Furthermore, in [94], Jackson proved a classic estimate for the error of the best approximation of $f : [0, 2\pi] \to \mathbb{R}$ by trigonometric functions (in the L^∞ norm), which we will state in the following theorem.

Theorem 6. (Jackson) *Let $f : [0, 2\pi] \to \mathbb{R}$ and each of its derivatives up to order r be continuous and 2π-periodic. Then there exists some constant $\alpha_{r,f}$ with the following property: for any n, there exists $T_n \in \mathbb{T}_n$ such that*

$$\|f - T_n\|_\infty \leq \frac{\alpha_{r,f}}{n^r}\,. \tag{2.62}$$

With Theorem 6, we may continue along similar lines to Equation (2.33) to arrive at the rough estimate for trigonometric interpolation error

$$\begin{aligned}\left\|f - \mathcal{U}^{m(l)}[f]\right\|_\infty &\leq \left(1 + \mathbb{L}_{m(l)}\right) \min_{T \in \mathbb{T}_{(m(l)-1)/2}} \|f - T\|_\infty \\ &\leq \left(1 + \mathbb{L}_{m(l)}\right) \frac{\alpha_{r,f} 2^r}{(3^l - 1)^r}\end{aligned} \tag{2.63}$$

for $f \in C^r([0, 1])$, with the function values and each derivative periodic on the interval $[0, 1]$. Here, the constant $\alpha_{r,f}$ may include a correction for the changed interval, and the Lebesgue constant is $\mathbb{L}_{m(l)} = \|\mathcal{U}^{m(l)}\|_{\infty,\infty}$, with the operator norm from (2.26). In order to determine if interpolation converges as $l \to \infty$, we need to bound $\mathbb{L}_{m(l)}$. From [65, 182, 226], we have the bound

$$\mathbb{L}_{m(l)} \leq \frac{2}{\pi} \log(m(l)) + 1 = \frac{2\log(3)}{\pi}\, l + 1\,. \tag{2.64}$$

Substituting (2.64) into (2.63), we indeed see that

$$\lim_{l \to \infty} \|f - \mathcal{U}^{m(l)}[f]\|_\infty = 0$$

for $f \in C^r([0, 1])$, $r \geq 1$, where f and each of its first r derivatives are periodic. Furthermore, since the one-dimensional Lebesgue constants grow linearly in the level, then the bound (2.28) on the sparse-grid Lebesgue constant holds with $\beta = 1$.

Let us now remark how Equation (2.61) has informed our choice of one-dimensional rule (2.54). Hallatschek [84] and Griebel and Hamaekers [81] have used a one-dimensional rule with growth $m(l) = 2^l$, the optimal choice for performing FFTs. But with such a rule, we would only obtain exactness for $\mathbb{T}_{\lfloor (m(l)-1)/2 \rfloor}$, where $\lfloor \cdot \rfloor$ denote the round-down function. That is, for $l \geq 1$, we would have *only the positive* exponential for mode 2^{l-1}, which does not provide enough resolution to resolve the entire highest-frequency mode.

In order to avoid unnecessary basis functions and keep the growth like 2^l, one could either discard an exponential and get $m(l) = 2^l - 1$ or add an exponential to obtain $m(l) = 2^l + 1$. Note, however, that our interpolation algorithm requires a Fourier transform (2.58) of the data. The premise of an FFT is to break the data down recursively into smaller segments and then perform the transform; the computational cost of an FFT algorithm is $O(n \log n)$, where n is the number of data points. With $m(l) = 2^l - 1$, our number of data points will hit all of the Mersenne primes, in which case we would be stuck with a prime-numbered FFT, which is more costly than $O(n \log n)$. For $m(l) = 2^l + 1$ (and also for composite numbers of the form $2^l - 1$), the integer may be factorizable, but one of the factors may be a very large prime [32], leaving us again stuck performing a costly computation on the prime-numbered data segment. Furthermore, the equidistant rule in Equation (2.54) is not nested for $m(l) = 2^l \pm 1$. All of these considerations lead us to choose $m(l) = 3^l$, which induces a nested rule and allows naturally for an $O(n \log n)$ Fourier transform. We describe this FFT in detail in Section 3.3.

We now transition to the full-tensor case. For $d > 1$, the (complex-valued) trigonometric interpolant on tensor $\boldsymbol{i} \in \mathbb{N}^d$ is

$$\tilde{\mathcal{U}}^{\boldsymbol{m}(\boldsymbol{i})}[f](\boldsymbol{x}) = \left(\bigotimes_{k=1}^{d} \mathcal{U}^{m(i_k)} \right) [f](\boldsymbol{x}) = \sum_{\boldsymbol{1} \leq \boldsymbol{j} \leq \boldsymbol{m}(\boldsymbol{i})} \hat{f}_{\boldsymbol{j}}^{\boldsymbol{i}} \varphi_{\boldsymbol{j}}(\boldsymbol{x}) \,, \tag{2.65}$$

where $\otimes$ denotes the tensor product and

$$\varphi_{\boldsymbol{j}}(\boldsymbol{x}) = \prod_{k=1}^{d} \varphi_{j_k}(x_k) \,, \tag{2.66}$$

$$\hat{f}_{\boldsymbol{j}}^{\boldsymbol{i}} = \frac{1}{m(i_1)} \cdots \frac{1}{m(i_d)} \sum_{\boldsymbol{1} \leq \boldsymbol{p} \leq \boldsymbol{m}(\boldsymbol{i})} f(\boldsymbol{x}_{\boldsymbol{p}}) \varphi_{\boldsymbol{j}}^{*}(\boldsymbol{x}_{\boldsymbol{p}}) \,. \tag{2.67}$$

Furthermore, the full-tensor grid $\{\boldsymbol{x}_{\boldsymbol{j}}\}_{\boldsymbol{1} \leq \boldsymbol{j} \leq \boldsymbol{m}(\boldsymbol{i})}$ has the Cartesian product structure of Equation (2.12) on top of the one-dimensional rule (2.54). We note that Equation (2.67) is a d-dimensional discrete Fourier transform, normalized by the total number of points, so we use the d-dimensional FFT in our implementation.

To construct the real-valued interpolant $\mathcal{U}^{\boldsymbol{m}(\boldsymbol{i})}$ from the complex-valued interpolant $\tilde{\mathcal{U}}^{\boldsymbol{m}(\boldsymbol{i})}$, we take the real part of (2.65):

$$\mathcal{U}^{\boldsymbol{m}(\boldsymbol{i})}[f](\boldsymbol{x}) = \sum_{\boldsymbol{1} \leq \boldsymbol{j} \leq \boldsymbol{m}(\boldsymbol{i})} \mathrm{Re}(\hat{f}_{\boldsymbol{j}}^{\boldsymbol{i}}) \mathrm{Re}(\varphi_{\boldsymbol{j}}(\boldsymbol{x})) - \mathrm{Im}(\hat{f}_{\boldsymbol{j}}^{\boldsymbol{i}}) \mathrm{Im}(\varphi_{\boldsymbol{j}}(\boldsymbol{x})) \,, \tag{2.68}$$

similarly to the one-dimensional case. We may express the real and imaginary parts of $\varphi_{\boldsymbol{j}}(\boldsymbol{x})$ as

$$\mathrm{Re}(\varphi_{\boldsymbol{j}}(\boldsymbol{x})) = \cos\left(2\pi \sum_{k=1}^{d} \sigma(j_k)\, x_k \right), \qquad \mathrm{Im}(\varphi_{\boldsymbol{j}}(\boldsymbol{x})) = \sin\left(2\pi \sum_{k=1}^{d} \sigma(j_k)\, x_k \right).$$

To find the space for which $\mathcal{U}^{\boldsymbol{m}(\boldsymbol{i})}$ is exact, we use a similar argument to the one-dimensional case along with repeated applications of the trigonometric identities

$$\cos(\alpha+\beta) = \cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta)\,,$$
$$\sin(\alpha+\beta) = \sin(\alpha)\cos(\beta) + \cos(\alpha)\sin(\beta)\,.$$

With this approach, we obtain that $\mathcal{U}^{\boldsymbol{m}(\boldsymbol{i})}[u] = u$ for all u in the space

$$\mathbb{T}_{(\boldsymbol{m}(\boldsymbol{i})-\boldsymbol{1})/2} = \bigotimes_{k=1}^{d} \mathbb{T}_{(m(i_k)-1)/2}\,, \tag{2.69}$$

where $\mathbb{T}_{(m(i_k)-1)/2}$ is defined according to Equation (2.61).

2.4.2 Sparse trigonometric interpolation

We proceed along similar lines to the sparse polynomial interpolation discussion in Section 2.3.2. We consider $f : [0,1]^d \to \mathbb{R}$. The general approach of Section 2.2 is basis-agnostic, so we again get

$$G^d_{\Theta(L)}[f] = \sum_{\boldsymbol{i}\in\Theta} \Delta^{\boldsymbol{m}(\boldsymbol{i})}[f]\,. \tag{2.70}$$

Here, we have chosen $\Theta(L)$ as the Smolyak space in Equation (2.24), $\Delta^{\boldsymbol{m}(\boldsymbol{i})}$ is given by Equation (2.13), and the interpolation operators are given by Equations (2.60) and (2.68). We suppress subscripting on $\Theta(L)$ right now for notational simplicity. Following Theorem 1 of Section 2.2, we may rewrite Equation (2.70) as

$$G^d_{\Theta(L)}[f] = \sum_{\|\boldsymbol{i}\|_1\le L} (-1)^{L-\|\boldsymbol{i}\|_1} \binom{d-1}{L-\|\boldsymbol{i}\|_1} \mathcal{U}^{\boldsymbol{m}(\boldsymbol{i})}[f]\,. \tag{2.71}$$

Here, we define the full-tensor trigonometric interpolation operator $\mathcal{U}^{\boldsymbol{m}(\boldsymbol{i})}$ from Equation (2.68). In terms of the general reformulation of a sparse operator in Equation (2.16), Equation (2.71) is a closed-form expression for the coefficients $t_{\boldsymbol{j}}$ under a specific choice of Θ.

Analogously to the polynomial case in (2.43), the set of all points utilized in (2.71) is

$$\mathcal{H}_{\Theta(L)} = \bigcup_{\boldsymbol{i}\in\Theta(L)} \{\boldsymbol{x}_{\boldsymbol{j}}\}_{\boldsymbol{1}\le\boldsymbol{j}\le\boldsymbol{m}(\boldsymbol{i})} = \bigcup_{\|\boldsymbol{i}\|\le L} \{\boldsymbol{x}_{\boldsymbol{j}}\}_{\boldsymbol{1}\le\boldsymbol{j}\le\boldsymbol{m}(\boldsymbol{i})} \tag{2.72}$$

where the full-tensor grids $\{\boldsymbol{x}_{\boldsymbol{j}}\}_{\boldsymbol{1}\le\boldsymbol{j}\le\boldsymbol{m}(\boldsymbol{i})}$ use the one-dimensional rule (2.54). In Figure 2.8, we display an example of $\mathcal{H}_{\Theta(L)}$ for $d = 2$, $L = 3$.

Hallatschek proved a result on the growth of the number of nodes required for his sparse

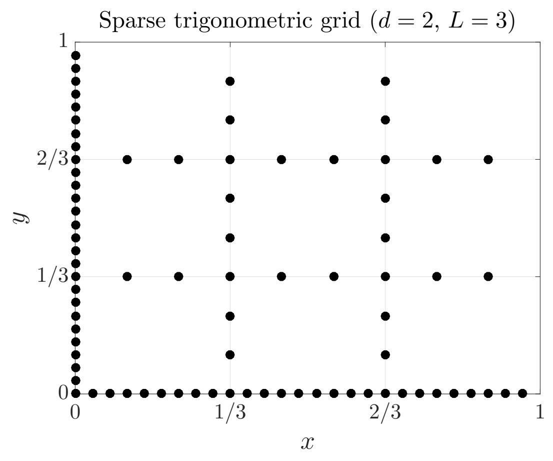


Figure 2.8 Sparse trigonometric interpolation points $\mathcal{H}_{\Theta(L)}$ for $d = 2$, $L = 3$. Here, we have 81 points.

interpolation scheme that used $m(l) = 2^l$. Following his approach in [84], we now provide a result on the cardinality of $\mathcal{H}_{\Theta(L)}$ for large d in the trigonometric case, stated in the following theorem.

Theorem 7. *Let $n(L, d) = \#\mathcal{H}^{\text{trig}}_{\Theta(L)}$ be the number of points required by the sparse trigonometric interpolation algorithm in Equation* (2.71)*, i.e. the cardinality of $\mathcal{H}^{\text{trig}}_{\Theta(L)}$. For fixed L and as $d \to \infty$,*

$$n(L, d) \approx \frac{2^L}{L!}\, d^L \tag{2.73}$$

where the relation $\approx$ is defined by Equation (2.44)*.*

Proof. We follow the approach of Hallatschek in [84] with slight modifications in light of $m(l) = 3^l$. For $L > 0$ and $d > 1$, we can write the recursion relation

$$n(L, d) = n(L, d-1) + 2 \sum_{j=0}^{L-1} 3^{L-1-j}\, n(j, d-1)\,. \tag{2.74}$$

Geometrically, this relation decomposes a sparse grid of level l into the portion along the $x_d = 0$ plane and the grids corresponding to $(d-1)$-dimensional grids of level j lying along the planes

$$x_d \in \left\{0, \frac{1}{3^j}, \ldots, \frac{3^j - 1}{3^j}\right\} \setminus \left\{0, \frac{1}{3^{j-1}}, \ldots, \frac{3^{j-1} - 1}{3^{j-1}}\right\}.$$

For each sparse grid of level j in dimension $d - 1$, there are $2 \cdot 3^{L-1-j}$ such planes; Figure 2.8

may be helpful for visualization. We also have

$$n(0,d) = 1, \qquad n(j,1) = 3^j\,, \tag{2.75}$$

for all $d \geq 1$ and $j \geq 0$. Using (2.75) in (2.74) and unwinding the recursion gives

$$n(L,d) = \sum_{j=0}^{\min(L,d-1)} 3^{L-j}\, 2^j \binom{L}{j}\binom{d-1}{j}. \tag{2.76}$$

for all $L \geq 0$, $d \geq 1$. Since we want to fix L and take $d \to \infty$, we consider the case $d > L$. We can bound $\binom{d-1}{j}$ as

$$\begin{aligned}\binom{d-1}{j} &= \frac{(d-1)!}{j!\,(d-1-j)!} = \frac{(d-1)\cdots(d-j)}{j!} \leq \frac{(d-1)^j}{j!}\,,\\ \binom{d-1}{j} &= \frac{(d-1)\cdots(d-j)}{j!} \geq \frac{(d-j)^j}{j!}\,.\end{aligned}$$

Substituting these bounds into (2.76) gives

$$\begin{aligned}n(L,d) &\leq \frac{(d-1)^L}{L!}2^L + \sum_{j=0}^{L-1} 3^{L-j}\, 2^j \binom{L}{j}\frac{(d-1)^j}{j!}\,,\\ n(L,d) &\geq \frac{(d-L)^L}{L!}2^L + \sum_{j=0}^{L-1} 3^{L-j}\, 2^j \binom{L}{j}\frac{(d-j)^j}{j!}\,,\end{aligned}$$

which implies $\lim_{d\to\infty} \dfrac{n(L,d)}{(2^L d^L)/L!} = 1$, as required. □

After discussing exactness, we will provide a concrete example of the asymptotic node growth of the sparse operator defined in Equation (2.71).

2.4.3 Exactness

In one dimension, trigonometric interpolation with 3^l equidistant nodes will exactly reproduce trigonometric functions of degree $(3^l - 1)/2$, where degree is understood to be the largest frequency. We use this one-dimensional relationship to prove a related result for sparse multivariate trigonometric interpolation. Our proof is similar to [155] and [147], which gave proofs for quadrature formulas and sparse polynomial interpolation, respectively.

Theorem 8. *$G^d_{\Theta(L)}$ will exactly reproduce all trigonometric functions of the form*

$$\sum_{\|\boldsymbol{i}\|_1=L} \left(\mathbb{T}_{(m(i_1)-1)/2} \otimes \cdots \otimes \mathbb{T}_{(m(i_d)-1)/2}\right), \tag{2.77}$$

where we define $\mathbb{T}_n$ by Equation (2.61)*.*

Proof. Let

$$f \in \sum_{\|\boldsymbol{i}\|_1=L} \left(\mathbb{T}_{(m(i_1)-1)/2} \otimes \cdots \otimes \mathbb{T}_{(m(i_d)-1)/2}\right).$$

We proceed by induction on d. For $d = 1$, we get $G^1_{\Theta(L)} = \mathcal{U}^{m(L)}$, which interpolates f exactly. Now, suppose $d \geq 1$ and f is a tensor product of univariate trigonometric functions:

$$f = f_{i_1} \otimes \cdots \otimes f_{i_d} \otimes f_{i_{d+1}}$$

where $\boldsymbol{i} \in \mathbb{N}^{d+1}$, $\|\boldsymbol{i}\|_1 = L$, and f_{i_r} has trigonometric degree at most $(m(i_r)-1)/2$. Also, assume as our inductive hypothesis that $G^d_{\Theta(L)}$ is exact on (2.77). We will show that $G^{d+1}_{\Theta(L)}$ is exact on (2.77).

We set $M = \sum_{k=1}^d i_k$, so that $M + i_{d+1} = L$. Recall our convention that $\mathcal{U}^{m(-1)} \equiv 0$. Following [155], we can manipulate Smolyak's original construction in Equation (2.70) to express $G^{d+1}_{\Theta(L)}$ in terms of $G^d_{\Theta(\ell)}$:

$$G^{d+1}_{\Theta(L)} = \sum_{\ell=0}^{L} G^d_{\Theta(\ell)} \otimes (\mathcal{U}^{m(L-\ell)} - \mathcal{U}^{m(L-\ell-1)}),$$

which gives

$$G^{d+1}_{\Theta(L)}[f] = \sum_{\ell=0}^{L} G^d_{\Theta(\ell)}[f_{i_1} \otimes \cdots \otimes f_{i_d}] \otimes (\mathcal{U}^{m(L-\ell)} - \mathcal{U}^{m(L-\ell-1)})[f_{i_{d+1}}].$$

By our inductive hypothesis, $G^d_{\Theta(L)}$ is exact, so for each ℓ with $0 \leq \ell \leq L$,

$$G^d_{\Theta(\ell)}[f_{i_1} \otimes \cdots \otimes f_{i_d}] = f_{i_1} \otimes \cdots \otimes f_{i_d}.$$

If $M \geq 1$, then for $0 \leq \ell \leq M-1$, we have $L - \ell - 1 \geq L - M = i_{d+1}$, so

$$\mathcal{U}^{m(L-\ell)}[f_{i_{d+1}}] = \mathcal{U}^{m(L-\ell-1)}[f_{i_{d+1}}] = f_{i_{d+1}}, \qquad 0 \leq \ell \leq M-1.$$

If $M = 0$, the above case does not arise. In either case, however, we obtain the telescoping sum

$$\begin{aligned} G^{d+1}_{\Theta(L)}[f] &= \sum_{\ell=M}^{L} (f_{i_1} \otimes \cdots \otimes f_{i_d}) \otimes (\mathcal{U}^{m(L-\ell)} - \mathcal{U}^{m(L-\ell-1)})[f_{i_{d+1}}] \\ &= (f_{i_1} \otimes \cdots \otimes f_{i_d}) \otimes \mathcal{U}^{m(L-M)}[f_{i_{d+1}}] \\ &= (f_{i_1} \otimes \cdots \otimes f_{i_d}) \otimes \mathcal{U}^{m(i_{d+1})}[f_{i_{d+1}}] \\ &= f_{i_1} \otimes \cdots \otimes f_{i_d} \otimes f_{i_{d+1}} = f \,, \end{aligned}$$

which completes the proof. □

Theorem 8 will allow us to find the degree of exactness for sparse trigonometric interpolation. We defined $\mathbb{T}_n$ in Equation (2.61) as the space of all trigonometric functions of degree at most n. We will now define precisely what we mean by trigonometric degree in the multidimensional case. For a one-dimensional trigonometric function f, we define

$$\deg_{1\mathrm{D}}(f) = \min\{n \;:\; f \in \mathbb{T}_n\}.$$

For a general d-variate, n-term trigonometric function of the form $g = \prod_{k=1}^{d} g_{1_k} + \cdots + \prod_{k=1}^{d} g_{n_k}$, the degree is

$$\deg(g) = \max_{1 \le i \le n} \sum_{k=1}^{d} \deg_{1\mathrm{D}}(g_{i_k})\,.$$

This is analogous to total degree for multivariate polynomials. Using Theorem 8 and similar reasoning to the polynomial case (e.g., taking $d > L$), we see that L is the trigonometric degree of exactness for the sparse operator in Equation (2.71).

Since L is the trigonometric degree of exactness, we may now study the number of nodes required to reproduce a d-variate trigonometric function of some fixed degree (say 3). In the one-dimensional case, we need $2(3)+1 = 7$ points, so the full-tensor construction will have 7^d points. Table 2.2 compares the number of points required in the full-tensor and sparse constructions for values of d ranging from 1 to 20. With the sparse algorithm, there is a dramatic reduction in computational complexity for large d versus the full-tensor case, even though the degree of exactness is the same.

2.4.4 Error estimation

Following Hallatschek in [84], we present an error estimate for sparse trigonometric interpolation. Here, we will work with the complex-valued interpolants based on the one-dimensional and full-

Table 2.2 Number of interpolation nodes needed to reproduce any degree-3 trigonometric function in the full-tensor and the sparse-grid construction (2.71) with $L = 3$. Both methods reproduce all frequencies up to the third mode.

d	Full-tensor	Sparse
1	7.00×10^0	27
2	4.90×10^1	81
3	3.43×10^2	171
5	1.68×10^4	491
10	2.83×10^8	2481
15	4.75×10^{12}	6971
20	7.98×10^{16}	14961

tensor operators defined by Equations (2.57) and (2.65). First, we let $f : [0,1]^d \to \mathbb{C}$ and denote

$$\hat{\omega}_j(x) = \exp(2\pi \mathrm{i}\, j\, x)\,, \qquad \omega_{\boldsymbol{j}}(\boldsymbol{x}) = \prod_{k=1}^{d} \hat{\omega}_{j_k}(x_k)\,,$$

where $\mathrm{i} = \sqrt{-1}$. Since $\{\omega_{\boldsymbol{j}}\}_{\boldsymbol{j}\in\mathbb{Z}^d}$ is a complete orthonormal set in $L^2([0,1]^d)$, then every $f \in L^2([0,1]^d)$ has the unique expansion

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{j}\in\mathbb{Z}^d} \hat{f}'_{\boldsymbol{j}}\, \omega_{\boldsymbol{j}}(\boldsymbol{x})\,, \tag{2.78}$$

where the continuous Fourier coefficients $\hat{f}_{\boldsymbol{j}}$ are given by

$$\hat{f}'_{\boldsymbol{j}} = \langle \omega_{\boldsymbol{j}}, f\rangle_{L^2} = \int_{[0,1]^d} \omega^*_{\boldsymbol{j}}(\boldsymbol{x}) f(\boldsymbol{x})\; d\boldsymbol{x}\,. \tag{2.79}$$

We use the prime on $\hat{f}'_{\boldsymbol{j}}$ to distinguish from the discrete Fourier coefficients $\hat{f}^{\boldsymbol{i}}_{\boldsymbol{j}}$ in Equation (2.67). We now introduce the Korobov space of parameter $a > 1$:

$$\begin{aligned} E^d_a = \Bigg\{ f : &[0,1]^d \to \mathbb{C}, f \in L^2([0,1]^d) \ : \ \exists C \in \mathbb{R}^+ \text{ s.t. } \forall \boldsymbol{j} \in \mathbb{Z}^d, \\ &|f_{\boldsymbol{j}}| \le \frac{C}{((1+|j_1|)(1+|j_2|)\cdots(1+|j_d|))^a} \Bigg\}\,. \end{aligned} \tag{2.80}$$

Intuitively, the parameter a controls the regularity of functions in E^d_a. Roughly speaking, as a gets larger, the functions in E^d_a become more regular because the bound on $|f_{\boldsymbol{j}}|$ gets tighter. Intriguingly, the denominator in Equation (2.80) is quite similar to the membership criterion for $\Theta^{\boldsymbol{1}}_{\text{hyp}}(L)$ in Equation (2.25) of Section 2.2.

In order to prove an error estimate, we will work with the surplus operators $\Delta^{\boldsymbol{m}(\boldsymbol{i})}$, which gauge the successive improvement of interpolation accuracy as we increase the interpolation level. Accordingly, we wish to express the one-dimensional Fourier interpolation operator $\mathcal{U}^{m(l)}$ as

$$\mathcal{U}^{m(l)}[f](x) = \sum_{j=1}^{m(l)} \check{f}_{\sigma(j)} h_j(x)\,. \tag{2.81}$$

Here, the $h_j(x)$ terms are hierarchical functions (recall Section 2.2), and $\check{f}_{\sigma(j)}$ are the hierarchical coefficients, to be determined. With some modifications to Hallatschek's approach [84] due to our using $m(l) = 3^l$ rather than $m(l) = 2^l$, we arrive at

$$h_j = \begin{cases} 0, & j = 1 \\ \varphi_j - \varphi_{2\cdot 3^{l-1}-j}, & \exists l \text{ s.t. } 3^{l-1}+1 \le j \le 2\cdot 3^{l-1} \\ \varphi_j - \varphi_{j-2\cdot 3^{l-1}-1}, & \exists l \text{ s.t. } 2\cdot 3^{l-1}+1 \le j \le 3^l \end{cases}. \tag{2.82}$$

We show examples of the hierarchical functions arising from Equation (2.82) in Table 2.3, along with the nodes of level $l-1$ for verification of the hierarchical property (2.8). With this machinery, we may write the one-dimensional surplus operator $\Delta^{m(l)}$ as

$$\Delta^{m(l)}[f](x) = \sum_{j=m(l-1)+1}^{m(l)} \check{f}_{\sigma(j)} h_j(x)\,, \tag{2.83}$$

which is merely a specific case of Equation (2.10). For dimensions $d > 1$, we have

$$h_{\boldsymbol{j}}(\boldsymbol{x}) = \prod_{k=1}^{d} h_{j_k}(x_k), \tag{2.84}$$

$$\Delta^{\boldsymbol{m}(\boldsymbol{i})}[f](\boldsymbol{x}) = \sum_{\boldsymbol{m}(\boldsymbol{i}-\boldsymbol{1})+\boldsymbol{1}\le\boldsymbol{j}\le\boldsymbol{m}(\boldsymbol{i})} \check{f}_{\boldsymbol{\sigma}(\boldsymbol{j})} h_{\boldsymbol{j}}(\boldsymbol{x})\,, \tag{2.85}$$

which is a specific form of Equation (2.14) for the Fourier basis.

To derive an error estimate, we must connect the hierarchical coefficients $\check{f}_{\sigma(j)}$ to the continuous Fourier coefficients f'_j in (2.79). Now, in Equation (2.81), at the highest hierarchical level (i.e., for j with $m(l-1)+1 \le j \le m(l)$), the non-hierarchical basis function φ_j appears only once. Thus, by equating the coefficients of φ_j, then

$$\check{f}_{\sigma(j)} = \hat{f}^l_j\,, \qquad m(l-1)+1 \le j \le m(l)\,.$$

where $\hat{f}^l_j$ is the discrete Fourier coefficient given by (2.58). Using the Fourier expansion (2.78)

Table 2.3 Examples of hierarchical functions $h_j(x)$ and the corresponding nodes where $h_j(x)$ is 0.

l	j	$h_j(x)$	Nodes at level $l-1$
0	1	1	—
1	2	$\exp(2\pi\mathrm{i}x) - 1$	$x = 0$
	3	$\exp(-2\pi\mathrm{i}x) - 1$	
2	4	$\exp(-2\pi\mathrm{i}x)(\exp(6\pi\mathrm{i}x) - 1)$	$x = 0, \frac{1}{3}, \frac{2}{3}$
	5	$\exp(2\pi\mathrm{i}x)(\exp(-6\pi\mathrm{i}x) - 1)$	
	6	$\exp(6\pi\mathrm{i}x) - 1$	
	7	$\exp(-6\pi\mathrm{i}x) - 1$	
	8	$\exp(2\pi\mathrm{i}x)(\exp(6\pi\mathrm{i}x) - 1)$	
	9	$\exp(-2\pi\mathrm{i}x)(\exp(-6\pi\mathrm{i}x) - 1)$	

in the formula for the discrete Fourier coefficients (2.58) along with the grid (2.54) gives

$$\begin{aligned}\check{f}_{\sigma(j)} &= \frac{1}{m(l)} \sum_{p=1}^{m(l)} \exp\left(\frac{-2\pi\mathrm{i}\,\sigma(j)\,p}{m(l)}\right) \sum_{q=-\infty}^{\infty} \hat{f}'_q \exp\left(\frac{2\pi\mathrm{i}\,q\,p}{m(l)}\right) \\ &= \frac{1}{m(l)} \sum_{q=-\infty}^{\infty} \hat{f}'_q \sum_{p=1}^{m(l)} \exp\left(\frac{2\pi\mathrm{i}\,(q-\sigma(j))\,p}{m(l)}\right) .\end{aligned}$$

But by the unitarity of the discrete Fourier transform operator, we have

$$\sum_{p=1}^{m(l)} \exp\left(\frac{2\pi\mathrm{i}\,(q-\sigma(j))\,p}{m(l)}\right) = \begin{cases} m(l), & \text{if } q = n\,m(l) + \sigma(j),\ n \in \mathbb{Z} \\ 0, & \text{otherwise} \end{cases}$$

so we get the aliasing formula (cf. [81, 84])

$$\check{f}_{\sigma(j)} = \sum_{n=-\infty}^{\infty} \hat{f}'_{\sigma(j)+n\,3^l}, \qquad 3^{l-1} + 1 \le j \le 3^l . \tag{2.86}$$

We are now prepared to state and prove an error estimate for the sparse interpolation operator (2.70).

Theorem 9. *Let $d > 0$, $a > 1$, $L > 0$. Let $f \in E_a^d$, and let $G_{\Theta(L)}^d$ be defined by Equation (2.70). There exists some constant $c_{a,d}$ (independent of L) such that*

$$\left\| G_{\Theta(L)}^d[f] - f \right\|_\infty \le c_{a,d}\,(L+1)^{d-1}\,3^{(1-a)L} \tag{2.87}$$

where $\|\cdot\|_\infty$ is the sup-norm on $[0,1]^d$.

Proof. This proof follows the approach of Hallatschek in [84] with slight modifications due to

$m(l) = 3^l$. First, for $f \in E_a^d$, we may write

$$f = \sum_{\boldsymbol{i} \in \mathbb{N}^d} \Delta^{\boldsymbol{m}(\boldsymbol{i})}[f] \, .$$

Let $\boldsymbol{x} \in [0,1]^d$ be arbitrary. We have

$$\begin{aligned} \left| f(\boldsymbol{x}) - G^d_{\Theta(L)}[f](\boldsymbol{x}) \right| &= \left| \sum_{\boldsymbol{i} \in \mathbb{N}^d, \ \|\boldsymbol{i}\|_1 > L} \Delta^{\boldsymbol{m}(\boldsymbol{i})}[f](\boldsymbol{x}) \right| \\ &\le \sum_{\boldsymbol{i} \in \mathbb{N}^d, \ \|\boldsymbol{i}\|_1 > L} \left| \Delta^{\boldsymbol{m}(\boldsymbol{i})}[f](\boldsymbol{x}) \right| \\ &\le \sum_{\boldsymbol{i} \in \mathbb{N}^d, \ \|\boldsymbol{i}\|_1 > L} \ \sum_{\boldsymbol{m}(\boldsymbol{i}-\boldsymbol{1})+\boldsymbol{1} \le \boldsymbol{j} \le \boldsymbol{m}(\boldsymbol{i})} \left| \check{f}_{\boldsymbol{\sigma}(\boldsymbol{j})} h_{\boldsymbol{j}}(\boldsymbol{x}) \right| \end{aligned} \tag{2.88}$$

From this, we see that we must bound $|h_{\boldsymbol{j}}(\boldsymbol{x})|$ and $|\check{f}_{\boldsymbol{\sigma}(\boldsymbol{j})}|$. For the first, we have

$$|h_{\boldsymbol{j}}(\boldsymbol{x})| \le \prod_{k=1}^{d} (1+1) = 2^d \, . \tag{2.89}$$

For the second, we set $\boldsymbol{\beta} = \boldsymbol{\sigma}(\boldsymbol{j})$ with $\boldsymbol{m}(\boldsymbol{i}-\boldsymbol{1}) + \boldsymbol{1} \le \boldsymbol{j} \le \boldsymbol{m}(\boldsymbol{i})$. We note that the aliasing formula (2.86) gives

$$\check{f}_{\boldsymbol{\beta}} = \sum_{\boldsymbol{\mu} \in \mathbb{Z}^d} \hat{f}'_{\boldsymbol{\beta} + (3^{i_1}\mu_1, \ldots, 3^{i_d}\mu_d)} \, .$$

But since $f \in E_a^d$, then by (2.80) there exists $C \in \mathbb{R}^+$, independent of $\boldsymbol{\beta}$, such that

$$\left| \check{f}_{\boldsymbol{\beta}} \right| \le C \prod_{k=1}^{d} \sum_{n=-\infty}^{\infty} (1 + |\beta_k + 3^{i_k} n|)^{-a} \, .$$

The reverse triangle inequality gives

$$|\beta_k + 3^{i_k} n| \ge \left| 3^{i_k} |n| - |\beta_k| \right| \, .$$

Since $\boldsymbol{\beta} = \boldsymbol{\sigma}(\boldsymbol{j})$, then

$$\frac{3^{i_k - 1} + 1}{2} \le |\beta_k| \le \frac{3^{i_k} - 1}{2} \, ,$$

so we have

$$\begin{aligned}\left|\check{f}_{\boldsymbol\beta}\right| &\le C\prod_{k=1}^{d}\left((1+|\beta_k|)^{-a}+2(1+3^{i_k}-|\beta_k|)^{-a}+2\sum_{n=2}^{\infty}(1+3^{i_k}n-|\beta_k|)^{-a}\right)\\ &\le C\prod_{k=1}^{d}\left(3(1+|\beta_k|)^{-a}+2\sum_{n=2}^{\infty}(1+3^{i_k}n-|\beta_k|)^{-a}\right)\end{aligned}$$

because $3^{i_k}-|\beta_k|\ge|\beta_k|$. We can bound the remaining sum by the improper integral

$$\begin{aligned}\sum_{n=2}^{\infty}(1+3^{i_k}n-|\beta_k|)^{-a} &\le \int_2^{\infty}(1+3^{i_k}\,x-|\beta_k|)^{-a}\;dx\\ &= 3^{-i_k}\,\frac{(1+3^{i_k}\,x-|\beta_k|)^{1-a}}{1-a}\Bigg|_{x=2}^{x\to\infty}\\ &\le 3^{-i_k}\,\frac{(1+3^{i_k}-|\beta_k|)^{1-a}}{a-1}\\ &\le 3^{-i_k}\,\frac{(1+|\beta_k|)^{1-a}}{a-1}\end{aligned}$$

since $a>1$. This results in

$$\left|\check{f}_{\boldsymbol\beta}\right| \le C\prod_{k=1}^{d}\left(3(1+|\beta_k|)^{-a}+2\cdot 3^{-i_k}\,\frac{(1+|\beta_k|)^{1-a}}{a-1}\right).$$

But since $|\beta_k|\le\frac{3^{i_k}-1}{2}$, then $1+|\beta_k|\le 3^{i_k}$, so $3^{-i_k}\le(1+|\beta_k|)^{-1}$. Therefore, along with $D_a=2C\cdot\max(3,\frac{2}{a-1})$, we get

$$\left|\check{f}_{\boldsymbol\beta}\right| \le \frac{D_a}{\prod_{k=1}^{d}(1+|\beta_k|)^a}. \tag{2.90}$$

Using (2.89) and (2.90) in the second sum of (2.88), we obtain

$$\begin{aligned}\sum_{\boldsymbol m(\boldsymbol i-\boldsymbol 1)+\boldsymbol 1\le \boldsymbol j\le \boldsymbol m(\boldsymbol i)}\left|\check{f}_{\boldsymbol\sigma(\boldsymbol j)}h_{\boldsymbol j}(\boldsymbol x)\right| &\le 2^d\sum_{\boldsymbol m(\boldsymbol i-\boldsymbol 1)+\boldsymbol 1\le \boldsymbol j\le \boldsymbol m(\boldsymbol i)}\frac{D_a}{\prod_{k=1}^{d}(1+|\beta_k|)^a}\\ &\le 2^d\sum_{\boldsymbol m(\boldsymbol i-\boldsymbol 1)+\boldsymbol 1\le \boldsymbol j\le \boldsymbol m(\boldsymbol i)}\frac{D_a}{\prod_{k=1}^{d}(1+3^{i_k-2})^a}\\ &\le 2^d\cdot\left(2^d\cdot 3^{\|\boldsymbol i\|_1}\right)\cdot D_a\cdot 3^{(2d-\|\boldsymbol i\|_1)a}\\ &\le \hat{C}_{a,d}\,3^{(1-a)\,\|\boldsymbol i\|_1}.\end{aligned}$$

The above inequality, combined with (2.88), yields

$$\begin{aligned}\left|f(\boldsymbol{x}) - G^d_{\Theta(L)}[f](\boldsymbol{x})\right| &\le \sum_{\boldsymbol{i}\in\mathbb{N}^d,\ \|\boldsymbol{i}\|_1 > L} \hat{C}_{a,d}\, 3^{(1-a)\,\|\boldsymbol{i}\|_1} \\ &= \hat{C}_{a,d} \sum_{p=L+1}^{\infty} 3^{(1-a)\,p} \sum_{\boldsymbol{i}\in\mathbb{N}^d,\ \|\boldsymbol{i}\|_1 = p} 1 \\ &\le \hat{C}_{a,d} \sum_{p=L+1}^{\infty} 3^{(1-a)\,p}\, p^{d-1} \\ &\le \hat{C}_{a,d}\, 3^{(1-a)L}\, (L+1)^{d-1} \sum_{n=1}^{\infty} 3^{(1-a)\,n}\, n^{d-1}\,. \qquad (2.91)\end{aligned}$$

Finally, by the Ratio Test, we get

$$\lim_{n\to\infty} \left| \frac{3^{(1-a)(n+1)}(n+1)^{d-1}}{3^{(1-a)n} n^{d-1}} \right| = \lim_{n\to\infty} \left| 3^{(1-a)} \left(\frac{n+1}{n}\right)^{d-1} \right| < 1$$

since $a > 1$. Therefore, the infinite sum in (2.91) converges to a value which depends on a and d (but not L), and the inequality in (2.87) follows. □

Finally, we present the optimal set of tensors Θ^{opt}_{trig}. Consider a lower set $\Lambda \subset \mathbb{N}^d$ and a target function space

$$\mathbb{T}_\Lambda = \bigcup_{\boldsymbol{\nu}\in\Lambda} \mathbb{T}_{\boldsymbol{\nu}} = \bigcup_{\boldsymbol{\nu}\in\Lambda} \bigotimes_{k=1}^{d} \mathbb{T}_{\nu_k}\,. \qquad (2.92)$$

Recalling that trigonometric interpolation with $2n+1$ points is exact up to mode n and using a similar result in [206], we get

$$\Theta^{opt}_{trig} = \left\{ \boldsymbol{i} \in \mathbb{N}^d \ :\ (\boldsymbol{m}(\boldsymbol{i}-\boldsymbol{1})+\boldsymbol{1})/2 \in \Lambda \right\}\,. \qquad (2.93)$$

2.4.5 Adaptive refinement

First, we consider the space of multidimensional periodic functions. We will approximate functions in this space *quasi-optimally*, meaning we will use sharp upper bounds on the Fourier coefficients to choose the approximation space, e.g. [206, 213]. Using upper bounds on the Fourier coefficients, we derive the quasi-optimal approximation space in the context of L^2 projection. From projection we proceed to interpolation and derive the quasi-optimal interpolation space. We conclude by discussing how to estimate the anisotropic coefficients of the target function *on-the-fly*. This section is based heavily on the author's work with Stoyanov in [140].

We let $[0,1]$ represent a one-dimensional torus. Let $H^n([0,1]) \subset C^n([0,1])$, with $n \ge 0$, denote the space of n-times continuously differentiable functions $f : [0,1] \to \mathbb{R}$ such that f has

n periodic derivatives and $f^{(n+1)}$ is piece-wise continuous with only isolated jump discontinuities. In the d-dimensional case, for $\boldsymbol{n} = (n_1, n_2, \cdots, n_d)$, we define

$$H^{\boldsymbol{n}}([0,1]^d) = H^{n_1}([0,1]) \otimes \cdots \otimes H^{n_d}([0,1])$$

so that for any $f \in H^{\boldsymbol{n}}([0,1]^d)$, $1 \le k \le d$, and $(x_1, x_2, \cdots, x_d) \in [0,1]^d$

$$f(x_1, \cdots, x_{k-1}, x, x_{k+1}, \cdots x_d) \in H^{n_k}([0,1]),$$

i.e., restricting f to a single dimension yields a function in $H^{n_k}([0,1])$. Here, without loss of generality, we consider the canonical torus $[0,1]^d$ since any arbitrary torus $\Gamma = \bigotimes_{k=1}^d [a_k, b_k]$ can be translated to $[0,1]^d$ with a simple affine transformation.

The coefficients of the Fourier expansion of $f \in H^{\boldsymbol{n}}$ are defined as

$$c_{\boldsymbol{j}}(f) = \int_{[0,1]^d} \exp(-2\pi \mathrm{i}\, \boldsymbol{j} \cdot \boldsymbol{x}) f(\boldsymbol{x})\, d\boldsymbol{x}, \qquad \boldsymbol{j} \in \mathbb{Z}^d, \tag{2.94}$$

with $\mathrm{i}^2 = -1$ and $\boldsymbol{j} \cdot \boldsymbol{x} = \sum_{k=1}^d j_k x_k$. In a single dimensional context, using Theorems 1.6, 4.4, and 4.5 from [102, pp. 4, 25] and trivial re-indexing, we obtain

$$|c_j(f)| \le \frac{C}{(1+|j|)^{n+2}}\,, \qquad j \in \mathbb{Z}\,,\ f \in H^n([0,1]), \tag{2.95}$$

for some constant $C > 0$. Furthermore, since $f^{(n+1)}$ has jump discontinuities, the bound in (2.95) is sharp [80, p. 200]. In a multidimensional context, using the tensor-product structure of the space, we have

$$|c_{\boldsymbol{j}}(f)| \le \frac{C}{\prod_{k=1}^d (1+|j_k|)^{n_k+2}}, \qquad \boldsymbol{j} \in \mathbb{Z}^d,\ f \in H^{\boldsymbol{n}}([0,1]^d)\,. \tag{2.96}$$

Function spaces similar to $H^{\boldsymbol{n}}([0,1]^d)$ have appeared in the literature as weighted Korobov spaces. In an early work on sparse trigonometric interpolation, Hallatschek [84] considered the isotropic Korobov space in (2.80), where $a > 1$ is a smoothness parameter. In general, a may take on any real value greater than one, but integer values have an interpretation in terms of the order of differentiability [160]. One may directly connect $a \in \{2, 3, \dots\}$ back to $H^{\boldsymbol{n}}([0,1]^d)$ and (2.96) by taking $\boldsymbol{n} = (a-2, \dots, a-2)$. However, there is precedent in the literature for our consideration of anisotropic, rather than isotropic, approximations for functions obeying (2.96). Authors have recently studied the complexity of approximation algorithms in anisotropic Korobov spaces [113, 157, 160] in addition to approximation in anisotropic Sobolev and Besov spaces [190]. There is also a long tradition of dimensionally and spatially adaptive refinement within the context of sparse-grid interpolation with a piece-wise or Lagrange polynomial basis, e.g. [34, 82, 96–98, 106, 107, 131, 148, 152, 153, 165, 166, 202, 204, 206].

Let $\Lambda \subset \mathbb{N}^d$ be lower. Consider the best approximation to f, denoted f_Λ, within the space $\mathbb{T}_\Lambda$ (2.92). From e.g. [111], the L^2 approximation error is

$$\|f - f_\Lambda\|_{L^2} = \sum_{\boldsymbol{\nu} \notin \Lambda} |c_{\boldsymbol{\nu}}|^2$$

where the $c_{\boldsymbol{\nu}}$ are defined in (2.94). It follows that the best M-term approximation uses only the largest M Fourier coefficients of f. Since the Fourier coefficients obey (2.96), then the quasi-optimal projection space is the hyperbolic cross-section $\Lambda^{\boldsymbol{\alpha}}_{hyp}(L)$ in (2.3), where $\boldsymbol{\alpha} = \boldsymbol{n} + \mathbf{2}$:

$$\Lambda^{\boldsymbol{\alpha}}_{hyp}(L) = \left\{ \boldsymbol{i} \in \mathbb{N}^d \ : \ \prod_{k=1}^{d} (i_k + 1)^{\alpha_k} \leq L \right\} \tag{2.97}$$

Now, let f_Λ denote an interpolatory, rather than projective, approximation in $\mathbb{T}_\Lambda$ to f. Interpolation error in the max norm is bounded by

$$\|f - f_\Lambda\|_\infty \leq (1 + \mathbb{L}_\Lambda) \inf_{T \in \mathbb{T}_\Lambda} \|f - T\|_\infty \tag{2.98}$$

where $\mathbb{L}_\Lambda$ is the Lebesgue constant of interpolation in $\mathbb{T}_\Lambda$ with a given choice of interpolation rule. We observe that

$$\inf_{T \in \mathbb{T}_\Lambda} \|f - T\|_\infty \leq \left\| f - \sum_{\boldsymbol{\nu} \in \Lambda} c_{\boldsymbol{\nu}} T_{\boldsymbol{\nu}} \right\|_\infty$$

where the $T_{\boldsymbol{\nu}}$ are trigonometric polynomials and the $c_{\boldsymbol{\nu}}$ are the best L^2 expansion coefficients. So, as in [140, 206], we may chain these inequalities together and note that the only difference versus the optimal L^2 approximation comes from the Lebesgue constant and from using the L^∞ rather than L^2 norm. Since the Lebesgue constant for one-dimensional trigonometric interpolation (2.64) grows logarithmically in the number of points, the effects of the Lebesgue constant are negligible compared to the denominator in (2.96), and we end up with the same quasi-optimal approximation space $\Lambda^{\boldsymbol{\alpha}}_{hyp}(L)$.

The specific values of the entries in the anisotropy vector $\boldsymbol{\alpha}$, while critical for constructing a quasi-optimal approximation, are seldom known *a priori*. In this section, we describe a method for estimating the anisotropy from an already constructed approximation $f_{\Lambda(L)}$ for some lower set $\Lambda(L)$. By definition, since $f_{\Lambda(L)} \in \mathbb{T}_{\Lambda(L)}$

$$f_{\Lambda(L)}(\boldsymbol{x}) = \sum_{\boldsymbol{\nu} \in \Lambda(L)} \hat{c}_{\boldsymbol{\nu}} \, T_{\boldsymbol{\nu}}(\boldsymbol{x})$$

where $\hat{c}_{\boldsymbol{\nu}}$ are either the projection coefficients or a corresponding set of interpolation weights. Often times, the $\hat{c}_{\boldsymbol{\nu}}$ are explicitly computed as part of the respective projection or interpolation procedure and hence available at no additional cost. If the estimate in (2.96) bounds the decay of $\hat{c}_{\boldsymbol{\nu}}$ sharply, then

$$|\hat{c}_{\boldsymbol{\nu}}| \approx \tilde{C} \prod_{k=1}^{d} (1 + |\nu_k|)^{-\alpha_k} \tag{2.99}$$

and the rates can be inferred from only two samples in each direction. If $\Lambda(L)$ is defined by (2.97) and $L \geq 2$, we can replace the approximate sign in (2.99) by an equal sign and solve the system of equations; however, in practice, the estimate is only an upper bound and the individual coefficients can vary, which gives an effect similar to noise. Thus, in order to cancel the noise, we take more samples in each direction and solve for the effective rates of decay from an over-determined set of equations, as in [140, 206]. Taking the log of both sides and changing signs, we obtain

$$-\log(|\hat{c}_{\boldsymbol{\nu}}|) \approx -C + \boldsymbol{\alpha} \cdot \log(\boldsymbol{\nu} + \mathbf{1}), \qquad \forall \boldsymbol{\nu} \in \Lambda(L). \tag{2.100}$$

for some constant C (different from the constant in (2.96)), where

$$\log(\boldsymbol{i}) = \bigotimes_{k=1}^{d} \log(i_k)\,.$$

To average out the effects of the “noise,” we take the least-squares solution, i.e., the solution that minimizes the ℓ^2 norm

$$\min_{\boldsymbol{\alpha}, C} \ \frac{1}{2} \sum_{\boldsymbol{\nu} \in \Lambda(L)} \left(C + \boldsymbol{\alpha} \cdot \log(\boldsymbol{\nu} + \mathbf{1}) + \log(|\hat{c}_{\boldsymbol{\nu}}|)\right)^2. \tag{2.101}$$

This can be written in a matrix form

$$\min_{\boldsymbol{v}} \frac{1}{2} \|\boldsymbol{A}\boldsymbol{v} - \boldsymbol{b}\|_2, \tag{2.102}$$

where the rows of $\boldsymbol{A}$ are $(1, \log(\nu_1 + 1), \log(\nu_2 + 1), \cdots, \log(\nu_d + 1))$, the solution is

$$\boldsymbol{v} = (C, \alpha_1, \cdots, \alpha_d)^T,$$

and $\boldsymbol{b}$ holds the corresponding entries of $\log(\hat{c}_{\boldsymbol{\nu}})$. Equation (2.102) admits a unique solution so long as $\boldsymbol{A}$ has full column rank, i.e., so long as we have at least two coefficients in each direction to estimate the corresponding decay rate. However, the accuracy of the estimated rates is also heavily dependent on the condition number of $\boldsymbol{A}$, and since (2.99) is only an approximation, more than a couple of coefficients are required. Also note that the constant C does not enter into the estimate of $\boldsymbol{\alpha}$ except as a dummy variable and a regularizer within the least-squares problem.

Equation (2.102) admits a unique solution so long as $\boldsymbol{A}$ has full column rank, i.e., so long as we have at least two coefficients in each direction to estimate the corresponding decay rate. However, since the coefficients may not decay monotonically and since the accuracy of the solution heavily depends on the condition number of the matrix $\boldsymbol{A}$, the approximation with only two coefficients will not suffice. The estimated $\boldsymbol{\alpha}$ may be too inaccurate or even yield negative decay rates, which according to (2.97) results in $\Lambda(L)$ with infinitely many multi-indexes. Nevertheless, we employ the estimate in an adaptive refinement strategy presented in Algorithm 1, and in Remark 1 we propose an ad-hoc strategy, specific to the refinement procedure, that would allow us to move forward with the adaptive steps even if some of the computed α_k are negative.

Example 4. *The motivation for the least-squares fitting* (2.101) *can be demonstated in the following one-dimensional example*

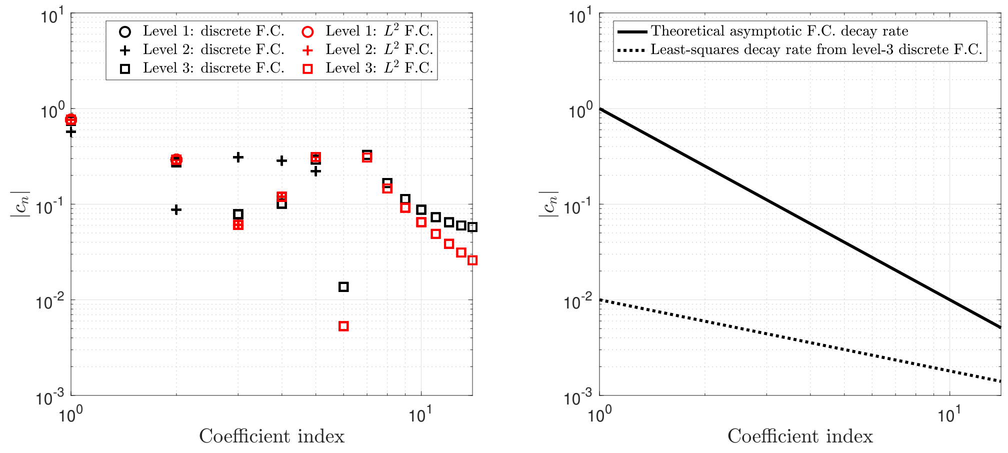
$$f(x) = x\sin(\pi x) + x\sin(5\pi x), \qquad x \in [-1, 1], \tag{2.103}$$

where $f \in H^0([0,1])$. *In Figure 2.9, we show the computed continuous and discrete Fourier coefficients, the theoretical decay rate according to* (2.96)*, and the decay rate estimate coming from* (2.101)*. We observe that on each level, for the largest indexes in* Λ*, the discrete Fourier coefficients differ systematically from the continuous Fourier coefficients. This can be explained by observing that the discrete Fourier transform* (2.58) *is a left-hand Riemann sum discretization of* (2.94)*; therefore, the largest indexes in* Λ *correspond to the highest frequencies, and the discretization is not able to resolve those to the same degree of accuracy. A refinement criterion could be based on the surplus or the correction introduced by the high frequencies, e.g., similar to the greedy knapsack problem [34], but such refinement would be guided by the least accurate coefficients. This phenomenon is not present in the methods using hierarchical Lagrange approximation where adding more indexes to* Λ *would not alter the current set of polynomial coefficients. Furthermore, there are unpredictable fluctuations in the preasymptotic low-frequencies. Since the breaking point between the two regimes is unknown, we use the least-squares approach defined in* (2.101) *to incorporate all coefficients and balance out the "noise-like" effects.*

We aim to estimate anisotropy *on the fly* in the course of an interpolatory approximation that is iteratively refined. Using (2.16), (2.19), and (2.65), we may write

$$G_{\Theta}^{d}[f](\boldsymbol{x}) = \sum_{\boldsymbol{j}\in X(\Theta)} \sum_{\substack{\boldsymbol{i}\in\Theta \\ \boldsymbol{1}\le\boldsymbol{j}\le\boldsymbol{m}(\boldsymbol{i})}} \mathrm{Re}\left[t_{\boldsymbol{j}}\, \hat{f}_{\boldsymbol{j}}^{\boldsymbol{i}}\, \varphi_{\boldsymbol{j}}(\boldsymbol{x})\right] = \sum_{\boldsymbol{j}\in X(\Theta)} \mathrm{Re}[w_{\boldsymbol{j}}\, \varphi_{\boldsymbol{j}}(\boldsymbol{x})] \tag{2.104}$$

Levels 1–3



Levels 4–6

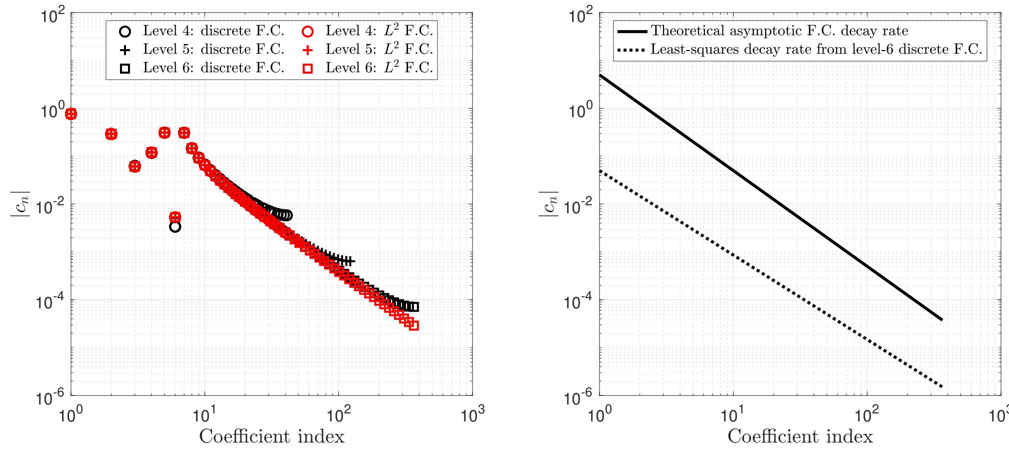


Figure 2.9 Left column: discrete and L^2 Fourier coefficients (F.C.) of (2.103) on 1D grids of various sizes. Right column: decay rates from (2.95) and (2.101).

Here, we compute the sparse discrete Fourier coefficients $w_{\boldsymbol{j}}$ at the front end of our interpolation algorithm, before any point-evaluations at $\boldsymbol{x}$:

$$w_{\boldsymbol{j}} = \sum_{\substack{\boldsymbol{i}\in\Theta \\ \boldsymbol{1}\le\boldsymbol{j}\le\boldsymbol{m}(\boldsymbol{i})}} t_{\boldsymbol{j}}\, \hat{f}^{\boldsymbol{i}}_{\boldsymbol{j}}\,, \qquad \boldsymbol{j}\in X(\Theta). \tag{2.105}$$

These $w_{\boldsymbol{j}}$ are linear combinations of the discrete Fourier coefficients $\hat{f}^{\boldsymbol{i}}_{\boldsymbol{j}}$ summed over the constituent tensors of the sparse grid. From (2.52) and (2.67), the coefficient $w_{\boldsymbol{j}}$ corresponds to the multidimensional mode

$$(|\sigma(j_1)|, |\sigma(j_2)|, \dots, |\sigma(j_d)|)$$

where $|\sigma(j_k)|$ is the frequency in direction k. Thus, the relevant least-squares problem becomes

$$\min_{\boldsymbol{\alpha},C} \frac{1}{2} \sum_{\boldsymbol{j}\in X(\Theta)} \left(C + \boldsymbol{\alpha}\cdot\log(\boldsymbol{1}+\tilde{\boldsymbol{\sigma}}(\boldsymbol{j})) + \log(|w_{\boldsymbol{j}}|)\right)^2 \tag{2.106}$$

where

$$\tilde{\boldsymbol{\sigma}}(\boldsymbol{j}) = (|\sigma(j_1)|, \dots, |\sigma(j_d)|) \tag{2.107}$$

and $\sigma(j)$ is defined in (2.53). The least-squares problem (2.106) holds for any general lower Θ, but in practice, we begin with a target function space $\mathbb{T}_\Lambda$ and use Θ_{opt} from (2.93).

Algorithm 1 Adaptive refinement algorithm

```
$n \leftarrow 0$
Start with $\Lambda = \Lambda^{\boldsymbol{1}}_{hyp}(L_0)$ with $L_0 \ge 2$; define $\Theta$ according to (2.93)
Compute the samples of $f$ and load the values into the grid
while num_samples < budget do
    Solve (2.106) for $\hat{\boldsymbol{\alpha}}$
    Find $L_{n+1}$ such that $\Lambda^{\hat{\boldsymbol{\alpha}}}(L_{n+1}) \not\subseteq \Lambda$; define $\Theta^{\hat{\boldsymbol{\alpha}}}(L_{n+1})$ by (2.93)
    $\Lambda \leftarrow \Lambda \cup \Lambda^{\hat{\boldsymbol{\alpha}}}(L_{n+1})$; $\Theta \leftarrow \Theta \cup \Theta^{\hat{\boldsymbol{\alpha}}}(L_{n+1})$; $n \leftarrow n+1$
    Compute the samples of $f$ at the new points
end while
```

We show pseudocode for our algorithm in Algorithm 1. Importantly, since the solution of the least-squares problem (2.106) is heavily dependent on $X(\Theta)$, then one should choose Λ_0 so that it contains enough points to compute an initial anisotropy estimate that is reliable (i.e., heuristically "close" to the true $\boldsymbol{\alpha}$). For example, if f contains only higher frequencies in some direction where only a small number of points is used, the estimate may give the appearance that the direction is not important and Algorithm 1 will add points in other dimensions, thus deviating significantly from the correct anisotropy of f.

Remark 1 (Ad-hoc stabilization)**.** *Given a black-box model, it is not feasible to determine a priori the appropriate size of $\Lambda(L_0)$ that would yield a stable initial estimate of the anisotropic coefficients. However, the weights are only used to guide the refinement process. Thus, if we encounter a negative weight $\alpha_k \leq 0$ for some direction k, we can simply replace that weight with the smallest positive one, which will force the refinement to put additional points in direction k, which in turn will improve the estimate in the following iterations. If all weights are negative, then we continue the refinement using isotropic weights $\boldsymbol{\alpha} = \mathbf{1}$. The correction strategy will allow us to work past negative weights, but it is still possible for a coarse grid to yield positive yet incorrect weight that would deteriorate the convergence. However, the theoretical estimates are only asymptotic and in our numerical examples we observe the opposite behavior, namely that the pre-asymptotic weights improve the initial error compared to the optimal analytic weights, e.g. in 2.11. Therefore, in our examples we use isotropic initial $\Lambda(L_0)$ with $L_0 = 3$ which is one more than the absolute minimum.*

2.4.6 Selected numerical examples

We include several examples in this section to illustrate the performance of Algorithm 1. We will apply our algorithm to purpose-built periodic polynomials of known anisotropy and then to the chemistry problem that motivated this work. These simulations use the open-source Tasmanian package developed at Oak Ridge National Laboratory [200], which implements Algorithm 1 for sparse trigonometric interpolation.

First, to obtain a theoretical convergence rate for our interpolation algorithm, let $f \in H^{\boldsymbol{n}}([0,1]^d)$. Using a theorem of Jackson [168], we can bound the infimum term in 2.98 by

$$\inf_{T \in \mathbb{T}_\Lambda} \|f - T\|_\infty \leq \frac{C(f)}{N^{M+1}} \tag{2.108}$$

where $C > 0$ is a constant depending on f, $M = \min_k n_k$, and $N = \#\Theta_m^{opt}$ is the number of nodes. From [40, 140], we have

$$\left\| G^d_{\Theta(L)} \right\| \leq C^d \left(\#\Theta(L)\right)^2 .$$

Heuristically approximating $\#\Theta^{opt}$ as $\log(N)$ in light of $m(l) = 3^l$ gives

$$\|f - G^d_\Theta[f]\|_\infty \leq O\left(\log^2(N)/N^{M+1}\right) \tag{2.109}$$

for N sufficiently large.

Remark 2 (Alternative Function Space)**.** *As noted in Sections 2.1 and 2.2, much early work on sparse grids sought to approximate function spaces of some total degree:*

$$\Lambda^{\boldsymbol{\alpha}}_{tot}(L) = \left\{ \boldsymbol{i} \in \mathbb{N}^d \ : \ \boldsymbol{i} \cdot \boldsymbol{\alpha} \leq L \right\} \tag{2.110}$$

In Fourier interpolation, the total-degree space is suitable for target functions f having a holomorphic extension in component k within a polyellipse of radius α_k around the real axis, for each $1 \leq k \leq d$. To see why, suppose f is a function satisfying the previous analyticity assumptions. From, e.g. [102], the Fourier coefficients obey the asymptotically sharp estimate

$$|c_{\boldsymbol{\nu}}(f)| \leq C(f)\, exp(-\boldsymbol{\alpha} \cdot \tilde{\boldsymbol{\sigma}}(\boldsymbol{\nu}))\,, \qquad \boldsymbol{\nu} \in \mathbb{N}^d\,, \tag{2.111}$$

where $\tilde{\boldsymbol{\sigma}}(\boldsymbol{\nu})$ is defined by (2.107)*. By taking the negative logarithm of the right-hand side of 2.111 and ignoring the constant, we obtain the total-degree space* (2.110)*. For functions of this type, the least-squares problem* (2.106) *becomes*

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^d,\, \bar{C} \in \mathbb{R}} \ \frac{1}{2} \sum_{\boldsymbol{j} \in \Theta_m} (\bar{C} + \boldsymbol{\alpha} \cdot \tilde{\boldsymbol{\sigma}}(\boldsymbol{j}) + \log(|w_{\boldsymbol{j}}|))^2\,. \tag{2.112}$$

Our numerical examples will include a modification of Algorithm 1 that uses the total-degree space (2.110) *and the least-squares problem* (2.112)*.*

2.4.6.1 Periodic polynomials

We manufacture some multidimensional target functions that are engineered to have a certain order of differentiability and periodicity. We define the univariate functions $g_i : [-1, 1] \to \mathbb{R}$ as

$$\begin{aligned}
g_1(x) &= x^3 - x\,, \\
g_2(x) &= \frac{x^4}{4} - \frac{x^2}{2}\,, \\
g_3(x) &= \frac{x^5}{20} - \frac{x^3}{6} + \frac{7x}{60}\,, \\
g_4(x) &= \frac{x^6}{120} - \frac{x^4}{24} + \frac{7x^2}{120}\,, \\
g_5(x) &= \frac{x^7}{840} - \frac{x^5}{120} + \frac{7x^3}{360} - \frac{31x}{2520}\,,
\end{aligned}$$

which we have derived by starting with $g_1(x)$ and integrating repeatedly and choosing the constant to preserve periodicity. By construction, $g_k \in H^k([-1, 1])$, where we translate $[-1, 1]$ to $[0, 1]$ using a linear transformation and note that the $k + 1$-th derivative is discontinuous across the periodic boundary. For $1 \leq i \leq 5$, we normalize in the sup-norm by taking $h_i =$

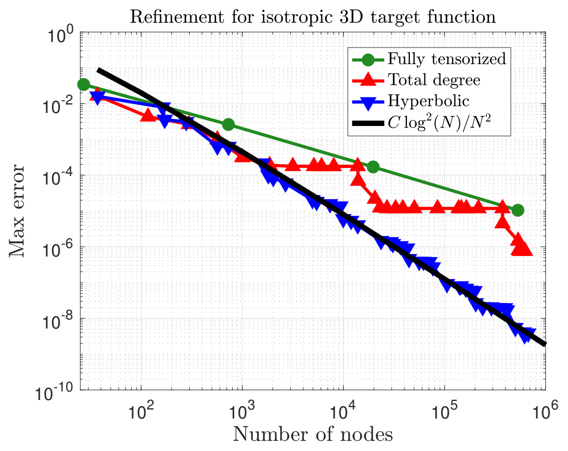


Figure 2.10 Isotropic refinement for trigonometric interpolation of $f_{(1,1,1)}(\boldsymbol{x}) = \prod_{k=1}^{3} h_1(x_k)$ with different choices of $\Lambda^{\boldsymbol{\alpha}}(L)$. Here, $\boldsymbol{\alpha} = \mathbf{1}$ and refinement occurs solely by incrementing L. As expected, the hyperbolic cross-section refinement converges at the expected rate and outperforms the total-degree and fully tensorized methods.

$g_i/\|g_i\|_{L^\infty([-1,1])}$. Thus, the multivariate target functions are

$$f_{\boldsymbol{i}}(\boldsymbol{x}) = \prod_{k=1}^{d} h_{i_k}(x_k), \qquad \mathbf{1} \le \boldsymbol{i} \le \mathbf{5}\,. \tag{2.113}$$

The domain of interpolation for (2.113) is $\Gamma = [-1,1]^d$. The Fourier coefficients obey the estimate (2.96), so a hyperbolic function space like (2.97) is appropriate, as Figure 2.10 demonstrates. We calculate the error by drawing 2000 validation points $\boldsymbol{x}_j \sim \mathcal{U}(\Gamma)$, where $\mathcal{U}(\Gamma)$ is the uniform distribution on Γ, with

$$\text{error} = \max_{1\le j\le 2000} |f(\boldsymbol{x}_j) - G^d_\Theta[f](\boldsymbol{x}_j)|\,.$$

In the isotropic example, the initial grids have approximately the same number of nodes, and we refine up to a maximum of 700000 nodes.

We now consider target functions with various numbers of inputs and anisotropy. The initial grid for each refinement strategy has approximately the same number of nodes, and we refine up to a maximum of 200000 nodes.

Next we consider an anisotropic example. In Figure 2.11, we compare different anisotropic grids, and we use the six-dimensional target function

$$\tilde{f}(\boldsymbol{x}) = h_1(x_1)h_5(x_4) + h_2(x_2)h_5(x_5) + h_3(x_3)h_5(x_6)\,. \tag{2.114}$$

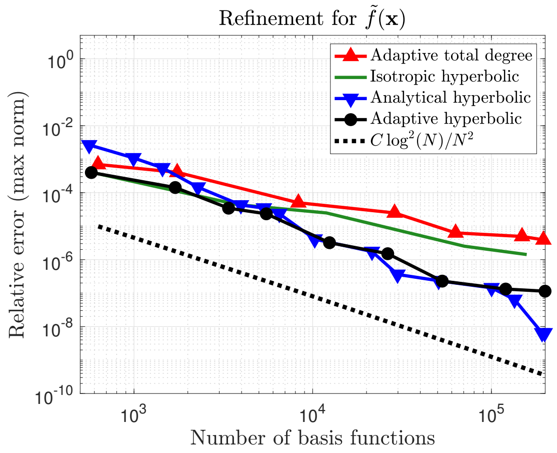


Figure 2.11 Convergence results for (2.114). The adaptive hyperbolic cross section methods matches the convergence rate of the analytic anisotropy, but without using any prior knowledge.

Table 2.4 Anisotropy ratios for two-dimensional product functions at the end of refinement. Column 2 uses Algorithm 1 and Column 3 uses Remark 2.

f_i	Final $\hat{\alpha}_1/\hat{\alpha}_2$ (hyperbolic)	Final $\hat{\alpha}_1/\hat{\alpha}_2$ (TD)	True α_1/α_2
$(1,2)$	0.72	0.73	0.75
$(1,3)$	0.61	0.61	0.60
$(1,4)$	0.49	0.49	0.50
$(1,5)$	0.45	0.45	0.43
$(2,3)$	0.84	0.84	0.80
$(2,4)$	0.68	0.68	0.67
$(2,5)$	0.62	0.62	0.57
$(3,4)$	0.81	0.81	0.83
$(3,5)$	0.74	0.74	0.71
$(4,5)$	0.91	0.91	0.86

Since $h_k \in H^k(\Gamma)$, then by (2.96) and (2.97), we know the anisotropy of $\tilde{f}$ beforehand:

$$\boldsymbol{\alpha} = (1,2,3,5,5,5) + \mathbf{2} = (3,4,5,7,7,7).$$

The line in Figure 2.11 labeled "Analytical hyperbolic" uses the known anisotropy $\boldsymbol{\alpha}$, while the adaptive strategies solve the relevant least-squares problem for $\hat{\boldsymbol{\alpha}}$ at each refinement iteration. In terms of convergence behavior, all strategies with a hyperbolic cross-section space outperform the total-degree space of Remark 2. Additionally, the adaptive algorithms based on solving the least-squares problem (2.106) converge at a similar rate as using the known target space $\Lambda^{\boldsymbol{\alpha}}(L)$ directly. Both the adaptive and analytical anisotropic strategies converge at approximately the rate given in (2.109). *This shows that Algorithm 1 is well suited to handle periodic models where the anisotropy is not known a priori.*

At the end of refinement, we obtain the following anisotropy estimates (normalized so that $\hat{\alpha}_1 = \alpha_1 = 3$):

$$\begin{aligned} \hat{\boldsymbol{\alpha}}_{hyp} &= (3.00,\ 3.53,\ 4.35,\ 5.58,\ 5.70,\ 5.73), \\ \hat{\boldsymbol{\alpha}}_{TD} &= (3.00,\ 3.63,\ 4.51,\ 6.11,\ 5.73,\ 5.40)\,. \end{aligned}$$

In Table 2.4, we show the anisotropy ratios at the end of adaptive refinement for two-dimensional product polynomials of the form (2.113). We compute the true anisotropy ratio for $f_{\boldsymbol{i}}$ by recalling $\alpha_k = i_k + 2$. Both Algorithm 1 and the modifications in Remark 2 are reasonably able to detect the relative anisotropy of the target function.

2.4.6.2 Particle in a two-dimensional box

We construct a two-dimensional particle in a box (PIB) system. This is a staple example in textbooks on quantum mechanics, e.g. [120]. Here, the anisotropy arises from different perturbations in the x and y directions. As discussed in [120], the Hamiltonian for the unperturbed one-dimensional PIB on the interval $[0,1]$ is

$$\hat{\mathcal{H}} = -\frac{1}{2}\frac{\mathrm{d}^2}{\mathrm{d}x^2} + V(x), \qquad V(x) = \begin{cases} 0, & x \in [0,1] \\ \infty, & \text{else} \end{cases} \tag{2.115}$$

where we have used atomic units and set the particle mass equal to the electron rest mass, $m_e = 1$. For $n = 1, 2, \ldots$, the normalized wavefunctions satisfy

$$\hat{\mathcal{H}}\,\psi_n = E_n\,\psi_n \qquad \Longrightarrow \qquad \psi_n(x) = \sqrt{2}\,\sin(n\pi x)\,,\ E_n = \frac{1}{2}n^2\pi^2\,.$$

Inspired by exercises in quantum mechanics textbooks [120], we use the potentials

$$f_1(x) = \begin{cases} 15, & x \in [0, 1/4] \cup [3/4, 1] \\ 0, & \text{else} \end{cases}, \qquad f_2(y) = 60\left(y - \frac{1}{2}\right)^2$$

in our two-dimensional perturbed PIB system and treat them as perturbations. Note that the maximum value of each perturbation is less than $E_2^{(0)}$, the energy of the unperturbed $n = 2$ energy level. The two-dimensional Hamiltonian is

$$\hat{\mathcal{H}} = -\frac{1}{2}\nabla^2 + V(x) + V(y) + f_1(x) + f_2(y) \tag{2.116}$$

where V is given in (2.115). The full two-dimensional wavefunction has the form

$$\Psi_{\boldsymbol{n}}(x, y) = \psi_{n_1}(x)\,\psi_{n_2}(y)$$

where n_k is the quantum number in dimension k.

We will demonstrate the performance of various refinement strategies on the wavefunction of (2.116) corresponding to $n_x = n_y = 2$. To evaluate the target wavefunction, we first decompose (2.116) into the x and y parts and apply first-order nondegenerate perturbation theory (see, e.g., p. 233 of [120]). The first-order correction to the wavefunction for the x component is

$$\psi_{2,x}^{(1)}(x) = \sum_{n \neq 2} \frac{\int_0^1 \psi_n^{(0)}(u)\, f_1(u)\, \psi_2^{(0)}(u)\, \mathrm{d}u}{E_2^{(0)} - E_n^{(0)}}\; \psi_n^{(0)}(x) \tag{2.117}$$

where $\psi_n^{(0)}$ and $E_n^{(0)}$ are the unperturbed wavefunctions and energies corresponding to (2.116). Similarly, for the y component, we get

$$\psi_{2,y}^{(1)}(y) = \sum_{n \neq 2} \frac{\int_0^1 \psi_n^{(0)}(u)\, f_2(u)\, \psi_2^{(0)}(u)\, \mathrm{d}u}{E_2^{(0)} - E_n^{(0)}}\; \psi_n^{(0)}(y)\,. \tag{2.118}$$

Thus, we take the two-dimensional target wavefunction as

$$\Psi_{2,2}(x, y) = \left(\psi_2^{(0)}(x) + \psi_{2,x}^{(1)}(x)\right)\left(\psi_2^{(0)}(y) + \psi_{2,y}^{(1)}(y)\right)\,. \tag{2.119}$$

We evaluate the integral coefficients in (2.117)-(2.118) with Maple and find that the only nonzero coefficients correspond to functions of the form $\sin(2\pi k x)$, yielding an *a priori* anisotropy estimate. The L^2-Fourier coefficients of $\Psi_{2,2}(x, y)$ decay like $O(1/k^3)$ in the x component and $O(1/k^5)$ in y, where k is the coefficient index. This both justifies the use of approximation space $\Lambda_{hyp}^{\boldsymbol{\alpha}}$ and gives the prior anisotropy $\boldsymbol{\alpha} = (3, 5)$. Computationally, we truncate the series

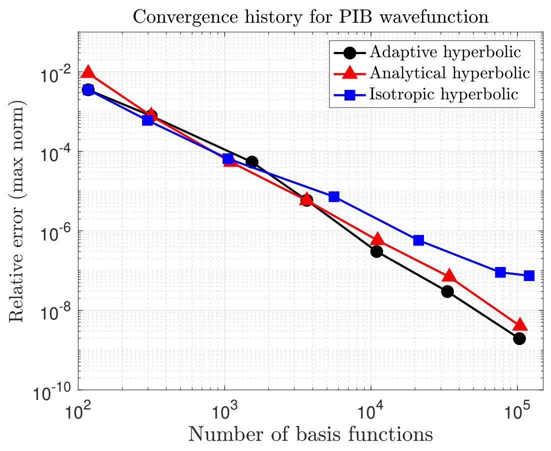


Figure 2.12 Convergence history of approximating (2.119) with various techniques.

in (2.117)-(2.118) at $N = 10^4$ terms and use (2.119) as our target function. In practice, though, one would not use Fourier interpolation on a known truncated Fourier series; instead, a more accurate solution technique would provide the target wavefunction, e.g. [45]. Perturbation theory, however, is straightforward enough to use for the end goal of demonstrating the convergence behavior of our adaptive refinement method.

For sparse interpolation, we use the hyperbolic index set Λ_{hyp} and refine according to three strategies: adaptive (Alg. 1), analytical anisotropy, and isotropic. We show the convergence behavior in (2.12). Similarly to Section 2.4.6.1, adaptive refinement performs as well as analytical anisotropic refinement, but without any prior knowledge. Asymptotically, the errors of the analytical anisotropic and adaptive strategies in Figure 2.12 decay at roughly the same rate and are an order of magnitude better than isotropic refinement. Furthermore, since we know the Fourier coefficients explicitly as a result of (2.117)-(2.118), we may construct the optimal lower approximation space directly from the explicit coefficients. In Figure 2.13 we show Λ_{hyp} at the final iteration of adaptive refinement along with the smallest lower set of size $\#(\Lambda_{hyp})$ containing the $N \leq \#(\Lambda_{hyp})$ largest Fourier coefficients. Figure 2.13 shows that adaptive refinement closely resembles the lower set containing the $N \leq \#(\Lambda_{hyp})$ largest Fourier coefficients, except for some rectangular gaps introduced by the growth rule $m(l) = 3^l$. We chose equal vertical and horizontal axes to make the anisotropy clear.

2.4.6.3 Potential energy surface of 2-butene

Now we consider the motivating application of this section: the adaptive approximation of a molecule's potential energy surface (PES) where the anisotropy $\boldsymbol{\alpha}$ is not known beforehand. The molecule of interest is 2-butene, whose molecular structure is shown in Figure 2.14. Using

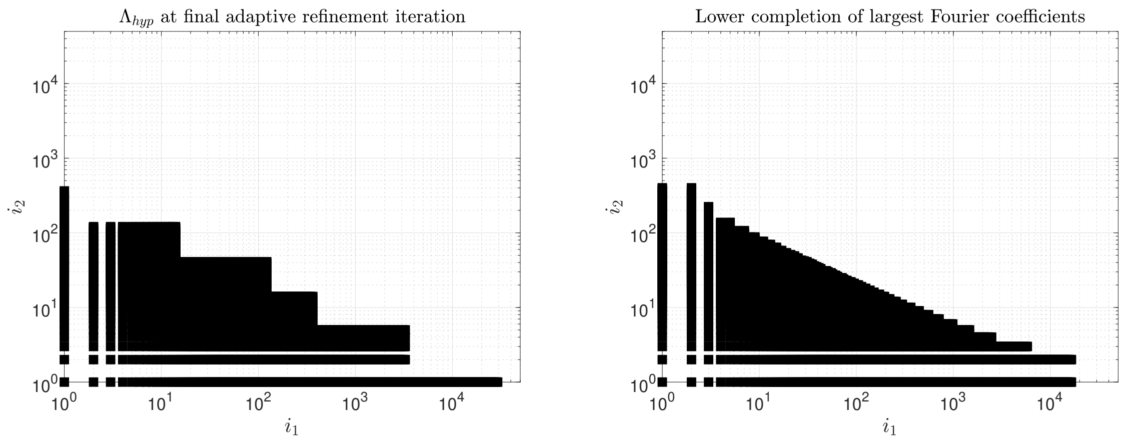


Figure 2.13 Λ_{hyp} at final iteration of adaptive refinement (left); lower completion of indices for the largest Fourier coefficients (right). Note the logarithmic scaling. Equal vertical and horizontal axes are chosen to display the anisotropy.

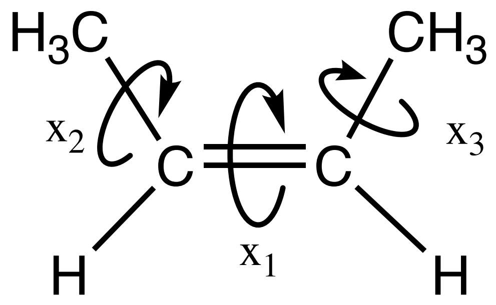


Figure 2.14 Molecular structure of 2-butene, labeled with rotations of interest.

the terminology of Chapter 1, we will approximate the relaxed PES $E_n(\boldsymbol{x})$, defined by (1.2), where $\boldsymbol{x}$ is the vector of design variables.

For rotational design variables (denoted $\boldsymbol{\theta}$), a polynomial interpolant does not guarantee periodicity of ∇E_n with respect to $\boldsymbol{\theta}$, which leads to nonphysical phenomena (e.g., nonconservation of energy). Therefore, a trigonometric interpolation basis is appropriate when $\boldsymbol{x}$ contains only bond angles and dihedral rotations. Bond lengths, in general, are not periodic over an interpolation domain, so approximation by trigonometric polynomials would lead to inaccuracies at the domain boundary [91].

As hinted earlier, solving the optimization (1.2) subject to the generalized eigenvalue problem (1.1) is an extremely expensive calculation. To trim down computational cost, it is common practice in quantum chemistry to use approximate Hamiltonians and wavefunctions [120]. In our case, we use density functional theory (with the B3LYP hybrid functional) to simplify the Hamiltonian [92, 108, 198], and we approximate the wavefunctions with the 6-311G* Pople basis

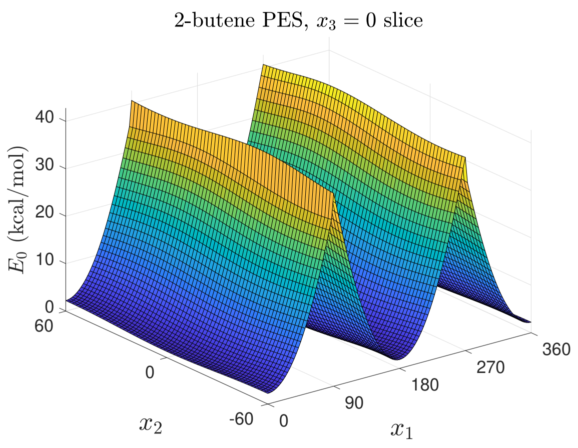


Figure 2.15 Slice of 2-butene PES for $x_3 = 0$.

set [112]. We describe basis sets and hybrid functionals in much more detail in Chapter 4. We use the Gaussian 16 software package [73] to handle the approximation of Hamiltonians and wavefunctions. By default, Gaussian 16 performs the optimization in (1.2) using a variant of the EDIIS algorithm tuned for molecular geometry optimizations [122].

Previous work constructed a sparse polynomial interpolant of $E_0(\boldsymbol{x})$ and $E_1(\boldsymbol{x})$ for 2-butene to study the transition from the *cis*- to *trans*- conformation via the first singlet excited state [146]. We use the same design variables from that study, shown in Figure 2.14. The design variable x_1 is more influential on the PES than x_2 and x_3, but the exact anisotropy is not known in advance.

The domain for our 2-butene ground-state ($n = 0$) PES is $\Gamma = [0, 360] \times [-60, 60] \times [-60, 60]$. The coordinates x_2 and x_3 correspond to dihedral rotations of CH_3, which have period 120°. We show a slice of the 2-butene PES in Figure 2.15. The ridges at $x_1 = 90$ and $x_1 = 270$ indicate a discontinuous first derivative, so we hypothesize that the hyperbolic function space (2.97) and Algorithm 1 are appropriate for this problem.

There are numerous sources of noise going into the evaluation of $E_0(\boldsymbol{x})$: density functional theory approximates the Hamiltonian, the 6-311G* basis set approximates the wavefunction Ψ, and the optimization (1.2) has internal stopping criteria. Therefore, we do not report the max error of the interpolant $G^d_\Theta[E_0](\boldsymbol{x})$, which could be heavily skewed by non-interpolatory error. Instead, we give the root-mean-square error (RMSE) over 2000 validation points drawn uniformly over Γ:

$$\text{RMSE} = \sqrt{\frac{\sum_{j=1}^{2000} (E_0(\boldsymbol{x}_j) - G^d_\Theta[E_0](\boldsymbol{x}_j))^2}{2000}}\,, \qquad \boldsymbol{x}_j \sim \mathcal{U}(\Gamma)\,. \tag{2.120}$$

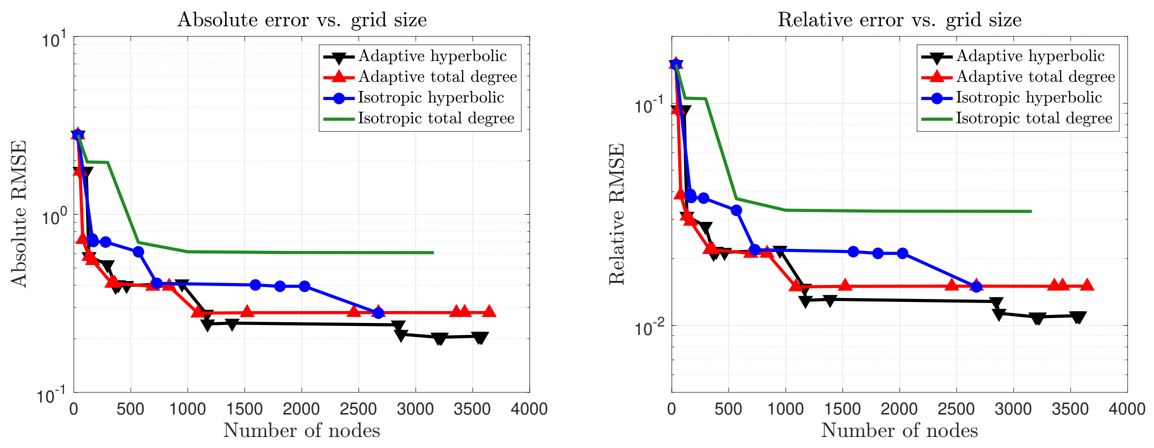


Figure 2.16 Absolute (left) and relative (right) error results for sparse interpolation of 2-butene ground-state PES.

In our sparse grid constructions, we use Θ_{opt} based on the hyperbolic function space Λ in (2.97) as well as the total-degree space (2.110). For each variety of Θ, we refine both adaptively (according to Algorithm 1 or Remark 2) and isotropically (taking $\boldsymbol{\alpha} = \mathbf{1}$ and incrementing L). In all cases, we initialize each grid with 37 nodes. Each function sample takes approximately 30 seconds to evaluate, and occasionally the optimization (1.2) may fail to converge to the correct (or any) local minimum. Due to limitations on available computing time, we refine up to a maximum of only 4000 nodes. If a refinement strategy terminates prior to 4000 nodes, that is because the next increment of L would result in the number of nodes exceeding 4000. We show the results in Figure 2.16. Following Pople in his 1998 Nobel lecture, we adopt 1 kcal/mol as the threshold of acceptable chemical accuracy for energies [172].

First, we note that the asymptotic absolute RMS errors in Figure 2.16 are consistent with Pople's definition of chemical accuracy for energies (i.e., less than 1 kcal/mol). Furthermore, the limiting relative error for adaptive hyperbolic refinement is approximately 1%. Second, even though the one-dimensional interpolation rule (2.55) grows exponentially, we can still add smaller batches of nodes at each iteration by using (2.93) and Algorithm 1, which mitigates the exponential growth of (2.55). Third, in both the adaptive and isotropic cases, the asymptotic error is lower for a hyperbolic cross-section than for a total-degree space.

2.5 Mixed basis

The question still remains as to how we combine sparse trigonometric interpolation with sparse polynomial interpolation. The interpolation coefficients in (2.11) are $c_j = f(\boldsymbol{x_j})$ for polynomial interpolation, but for trigonometric interpolation, c_j comes from a discrete Fourier transform, so we need to express Equation (2.16) differently. Helpfully, we may rewrite sparse Clenshaw–Curtis and trigonometric interpolants in adjoint form [200] as

$$G_\Theta[f](\boldsymbol{x}) = \sum_{\boldsymbol{j}\in X(\Theta)} \psi_{\boldsymbol{j}}(\boldsymbol{x})\, f(\boldsymbol{x}_{\boldsymbol{j}}), \tag{2.121}$$

where $X(\Theta)$ is the set of all sparse-grid indices (as opposed to the allowable tensors $\boldsymbol{i}$) defined in Equation (2.21), $\psi_{\boldsymbol{j}}(\boldsymbol{x})$ is the adjoint basis at an evaluation point $\boldsymbol{x}$, and $f(\boldsymbol{x}_{\boldsymbol{j}})$ is the function value at a specific node.

Now we partition the geometry into $\boldsymbol{x} = (\boldsymbol{y}, \boldsymbol{z})$, where $\boldsymbol{y}$ contains the periodic design variables and $\boldsymbol{z}$ has the nonperiodic ones. Furthermore, let us assume that the periodic and nonperiodic portions of f can be separated with multiplication and addition, but not function composition. In the context of PES approximation, this assumption is motivated by the exact solution of hydrogenlike atoms, which separates the wavefunction into a product of spherical harmonics and a radial part [120]. In this case, f has the structure

$$f(\boldsymbol{q}) = f_1(\boldsymbol{y}) \otimes f_2(\boldsymbol{z}),$$

where f_1 is the periodic portion of f, f_2 is the nonperiodic portion, and $\otimes$ denotes the possibility of both products and sums. Then we may use Equation (2.121) to apply Clenshaw–Curtis and trigonometric interpolation separately:

$$G_\Theta[f](\boldsymbol{x}) = \sum_{\boldsymbol{j}\in\Theta_m^{trig}} \sum_{\boldsymbol{k}\in\Theta_m^{poly}} \psi_{\boldsymbol{j}}^{trig}(\boldsymbol{y})\, \psi_{\boldsymbol{k}}^{poly}(\boldsymbol{z})\, f(\boldsymbol{y}_{\boldsymbol{j}}, \boldsymbol{z}_{\boldsymbol{k}}) \tag{2.122}$$

Equation (2.122) requires two grids: a sparse trigonometric grid for the periodic coordinates and a sparse polynomial-basis grid for the nonperiodic coordinates. Then, the overall grid is the tensor product of these two sparse grids. Moreover, one may think of the resulting grid as a sparse grid itself, with its own $\Theta = \Theta^{poly} \otimes \Theta^{trig}$.

CHAPTER

3 TASMANIAN

For implementation of sparse grids, we use the open-source Tasmanian package[1] (Toolkit for Adaptive Stochastic Modeling And Non-Intrusive ApproximatioN), developed by Stoyanov [200, 203] at Oak Ridge National Laboratory. Tasmanian is written in C++, with Python, MATLAB, and Fortran wrappers. The documentation is detailed [200], and the software contains a wide selection of sparse interpolation rules. In addition to the algorithms in Sections 2.3.2 and 2.4.2, Tasmanian allows the user to specify any of the admissible tensor spaces in Equations (2.22)–(2.25) of Section 2.2. Tasmanian also calls certain optimized or accelerated software tools if the user has installed them and has configured Tasmanian with the appropriate flags. These tools include OpenMP (parallelism), BLAS (optimized linear algebra), CUDA/cuBLAS various GPU acceleration tools (CUDA, HIP, and DPC++), and MAGMA (CPU+GPU hybrid architectures, [211]). The manual (available online at the Tasmanian website[1]) describes how to install Tasmanian and configure it for use with external acceleration tools [200].

Tasmanian has always included globally-defined sparse polynomial interpolants. Indeed, the current manual [200] describes more than 15 node choices for polynomial interpolation, one of which is the Clenshaw–Curtis rule of Section 2.3. However, prior to 2018, Tasmanian did not include Fourier interpolation. The author spent the summers of 2018 and 2019 at Oak Ridge National Laboratory adding trigonometric interpolation and the associated adaptive refinement method to Tasmanian in collaboration with Stoyanov [140]. The author's contributions are

[1] `https://github.com/ORNL/TASMANIAN/`; `https://tasmanian.ornl.gov/`

publicly available on the Tasmanian GitHub repository[2] as part of the production code. In Appendix B.2.2, we have included the version of the C++ Fourier basis code as of the end of the author's summer internship.

3.1 Installation and basic usage

To install Tasmanian and prepare it for general use, the reader first downloads the package from the Tasmanian website[1] or the Tasmanian GitHub repository.[2] The recommended choice is to download the latest stable release, but it is possible to clone the `master` branch from GitHub, which may contain highly experimental features. Once the package is on the user's local disk, there are several options for installation: `cmake`, the `install.sh` wrapper script, direct GNU `make`, `pip`, and `spack`. Here, for accessibility for the average user, we will discuss the `install.sh` wrapper script and `pip`. The `install.sh` script is merely a wrapper containing preset `cmake` options, but `cmake` is much more customizable and is recommended for high-performance computing applications.[3]

Before attempting to install Tasmanian, the user must first have a C/C++ compiler (such as `gcc` or `clang`) and `cmake`. Furthermore, if the user wishes to use the Python wrapper, then Tasmanian requires `numpy` and `c_types`. The Anaconda Python distribution includes both of these packages by default.

On a Unix-based machine with `cmake` installed, the user should open a terminal window and run the following commands for the basic install script:

```
cd <path-to-Tasmanian-download>
./install <Tasmanian-install-directory> <optional-MATLAB-work-directory>
↪  <optional-flags>
```

After building the source code, the installer will perform automatic tests to verify that the user's installation is functional. We strongly encourage the user to specify all directories as absolute paths. By default, Tasmanian will attempt to locate existing Python, OpenMP, and BLAS installations and link them to the Tasmanian source. If the user does not provide a MATLAB path, the installer will not install the MATLAB interface. For a complete list of optional flags, run

```
./install -help
```

[2] `https://github.com/ORNL/Tasmanian`

[3] Developers or advanced end-users may need to use `cmake` for the additional customizability. The online Tasmanian documentation [200] describes the details of installation with `cmake`.

from within the Tasmanian download folder. Some acceleration tools and interfaces are present in the software and not part of the default installation (e.g., CUDA, MAGMA, and Fortran), but may be enabled with the appropriate flag.

Alternatively, the user may install Tasmanian using `pip`. To enable optional installation flags (e.g., MATLAB), the CMake flags must be exported as environment variables first:

```
export Tasmanian_MATLAB_WORK_FOLDER=<path>
python3 -m pip install Tasmanian --user
```

Due to the ever-evolving nature of programming languages, the user is strongly encouraged to consult the online manual[1] for the current flags and calling sequences.

From the user's perspective, the main functionality is in the C++ `TasmanianSparseGrid()` class. The user will call this class and then invoke the subroutine `make***Grid()`. Currently, the available grid functions are `makeGlobalGrid`, `makeSequenceGrid`, `makeLocalPolynomialGrid`, `makeWaveletGrid`, and `makeFourierGrid`. At a minimum, each of these routines requires as input arguments the domain dimension d, the output dimension, the parameter L, and the tensor-selection strategy for Θ. The grids of interest to us are mainly `Global`, `Sequence`, and `Fourier`.[4] The first of these three performs polynomial interpolation with a Lagrange basis; the second, polynomial interpolation with a Newton basis;[5] and the third, trigonometric interpolation.

We show an example for the Python and MATLAB interfaces for `makeGlobalGrid` in Appendix B.2.1, complete with functional code listings (as of October 2018). In each example for Appendix B.2.1, we explicitly set the path in the code. For convenience, however, the user may permanently add the Tasmanian Python interface by adding the following line to his or her `~/.bashrc` file:

```
export PYTHONPATH =
↪  "${PYTHONPATH}:<Tasmanian-install-dir>/share/Tasmanian/python"
```

Similarly, for MATLAB, the user may run the following in the command window:

[4]The basis functions of a local polynomial grid are only *locally* nonzero on the interpolation domain, which can be useful for locally adaptive refinement if the interpolated function has rapid local changes. A wavelet grid represents the underlying function in terms of wavelets, which generalize the notion of a periodic transform. The interested reader is encouraged to consult the Tasmanian manual [200] for more details on wavelet and local polynomial grids.

[5]A Newton basis involves more work to compute the interpolation coefficients as a fixed up-front cost, but the payoff is that each evaluation of the interpolant is $O(N)$ rather than $O(N^2)$, as with Lagrange polynomials. Here, N is the number of nodes.

```
addpath('<Tasmanian-install-directory>/share/Tasmanian/matlab');
savepath;
```

The careful reader will notice from Appendix B.2.1 that the names of functions and the ordering of input arguments are not always the same between the Python and Matlab interfaces. The Python command names are identical to the names of the C++ routines, which the Tasmanian manual describes in detail [200]; the input arguments are ordered the same as well. Currently, however, the Matlab commands are less standardized, so we recommend checking the manual for the names of Matlab routines, which always begin with the prefix `tsg`. Furthermore, the ordering of input arguments are not always the same as for the C++ functions, so before running an unfamiliar Tasmanian function in the Matlab interface, we recommend using

```
help tsgFunctionName
```

which provides a detailed docstring.

3.2 Back-end program flow

Once the user invokes `make***Grid()` with input, the code performs overhead calculations, including the computation of Θ, the weights $t_{\boldsymbol{i}}$ for Equation (2.16), the active tensors and weights (i.e., the nonzero $t_{\boldsymbol{i}}$ and the corresponding subset of Θ), and the grid nodes $\{\boldsymbol{x}_{\boldsymbol{j}}^{\boldsymbol{i}}\}_{(\boldsymbol{i},\boldsymbol{j})\in\bar{X}(\Theta)}$. The user then retrieves the needed nodes using `getNeededPoints()` and loads the function values at the nodes with `loadNeededPoints()`. At this point, what happens is determined by the grid type. Global grids will merely store the function values; sequence grids will compute and store the Newton interpolation coefficients; and Fourier grids will compute and store the Fourier coefficients $\hat{f}_{\boldsymbol{j}}^{\boldsymbol{i}}$. These computations only need to be done at the front end or when the grid is adaptively refined by adding new nodes. Now, for nested grids, a particular basis function may appear more than once in the sum of Equation (2.16). We compute the "effective weights" $w_{\boldsymbol{j}}$ for the basis function $\phi_{\boldsymbol{j}}$ as

$$w_{\boldsymbol{j}} = \sum_{\boldsymbol{i}\in\Theta} \sum_{\boldsymbol{1}\le\boldsymbol{\nu}\le\boldsymbol{m}(\boldsymbol{i}),\ \boldsymbol{\nu}=\boldsymbol{j}} t_{\boldsymbol{i}}\, c_{\boldsymbol{\nu}}^{\boldsymbol{i}}\,, \tag{3.1}$$

which is a running sum of the product of each interpolation coefficient and $t_{\boldsymbol{i}}$ corresponding to the basis function $\phi_{\boldsymbol{j}}$. Here, we have assumed that the underlying function is real-valued; if the output dimension is larger than 1, then we compute (3.1) for each output dimension separately. In this way, we evaluate each basis function $\phi_{\boldsymbol{j}}$ exactly once.

After all of the overhead is completed, each interpolant evaluation is straightforward. The user uses one of the subroutines `evaluate`, `evaluateFast`, or `evaluateBatch`. For `evaluate`, user provides a single evaluation point $\boldsymbol{y} \in D \subset \mathbb{R}^d$, where D is the interpolation domain, and the code computes and returns

$$G_\Theta^d[f](\boldsymbol{y}) = \sum_{\boldsymbol{j} \in X(\Theta)} w_{\boldsymbol{j}} \phi_{\boldsymbol{j}}(\boldsymbol{y}) \,. \tag{3.2}$$

For `evaluateFast`, the user still specifies a single evaluation point $\boldsymbol{y}$, but rather than computing the sum in (3.2) directly, it will treat (3.2) as the dot-product of two vectors and use BLAS, cuBLAS, CUDA, or MAGMA to accelerate the computation. If none of these are installed, then `evaluateFast` will default to `evaluate`. For `evaluateBatch`, the user provides multiple evaluation points $\{y_p\}_{p=1}^M \subset D$. In this case, we index the weights and basis functions in (3.2) from 1 to N and get

$$\begin{bmatrix} G_\Theta^d[f](\boldsymbol{y}_1) \\ \vdots \\ G_\Theta^d[f](\boldsymbol{y}_M) \end{bmatrix} = \begin{bmatrix} \phi_1(\boldsymbol{y}_1) & \cdots & \phi_N(\boldsymbol{y}_1) \\ \vdots & \ddots & \vdots \\ \phi_1(\boldsymbol{y}_M) & \cdots & \phi_N(\boldsymbol{y}_M) \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_N \end{bmatrix} . \tag{3.3}$$

Upon calling `evaluateBatch`, Tasmanian will use acceleration tools to compute the matrix-vector product. For functions with output dimensions larger than 1, the weights $w_{\boldsymbol{j}}$ will be different in each output dimension, so (3.2) will be a matrix-vector product, and (3.3) will be a matrix-matrix product. In this context, systems with many nodes, many evaluation points, or many output dimensions make acceleration tools even more appealing.

3.3 Computational bottlenecks in trigonometric interpolation

The linear systems (3.2) and (3.3) are obvious computational bottlenecks, but the tools needed to make them more manageable are already in Tasmanian, and those systems appear for global grids as well. For Fourier interpolation specifically, the main bottlenecks are (a) computing the discrete Fourier coefficients $\hat{f}_{\boldsymbol{j}}^{\boldsymbol{i}}$ in Equation (2.67) at the front end, and (b) evaluating the basis functions $\varphi_{\boldsymbol{j}}(\boldsymbol{x}) = \prod_{k=1}^d \varphi_{j_k}(x_k)$ at an evaluation point $\boldsymbol{x} \in D \subset \mathbb{R}^d$. We will address the Fourier coefficients first.

As we noted in Section 2.4, computing $\hat{f}_{\boldsymbol{j}}^{\boldsymbol{i}}$ using Equation (2.67) is an $O(N^2)$ calculation—prohibitively expensive for a large number of nodes N. However, the choice of $m(l) = 3^l$ in our one-dimensional rule allows for a straightforward $O(N \log N)$ fast Fourier transform (FFT). In one dimension, the intuitive idea of an FFT is to break a length-3^l signal into three signals of length 3^{l-1} and perform the transform on these smaller sequences, with the length-1 signal as the base case. Mathematically, given $N = 3^l$ data points f_n $(0 \leq n \leq N-1)$, we can express the transformed values F_n as

$$F_n = \sum_{j=0}^{N-1} f_j \exp\left(-\frac{2\pi \mathrm{i}\, j\, n}{N}\right), \qquad n = 0, 1, \ldots, N-1\,. \tag{3.4}$$

We will now derive the recursion relation for the radix-3 fast Fourier transform. Introducing $p = 0, 1, 2$ and $q = 0, 1, \ldots, \frac{N}{3} - 1$, we get

$$F_{q+p\cdot(N/3)} = \sum_{j=0}^{N-1} f_j \exp\left(-\frac{2\pi \mathrm{i}\, j\,(q + p\cdot(N/3))}{N}\right)\,. \tag{3.5}$$

We now consider the terms in the sum. For $j = 3m$, we have

$$\begin{aligned} f_j \exp\left(-\frac{2\pi \mathrm{i}\, j\,(q + p\cdot(N/3))}{N}\right) &= f_{3m}\, \exp\left(-\frac{2\pi \mathrm{i}\,(3m)\,(q + p\cdot(N/3))}{N}\right) \\ &= f_{3m}\, \exp\left(-\frac{2\pi \mathrm{i}\, m\, q}{N/3}\right) \exp\left(2\pi \mathrm{i}\, m\, p\right) \\ &= f_{3m}\, \exp\left(-\frac{2\pi \mathrm{i}\, m\, q}{N/3}\right)\,. \end{aligned}$$

For $j = 3m + 1$, we have

$$\begin{aligned} f_j \exp\left(-\frac{2\pi \mathrm{i}\, j\,(q + p\cdot(N/3))}{N}\right) &= f_{3m+1}\, \exp\left(-\frac{2\pi \mathrm{i}\,(3m+1)\,(q + p\cdot(N/3))}{N}\right) \\ &= f_{3m+1}\, \exp\left(-\frac{2\pi \mathrm{i}\, m\, q}{N/3}\right) \exp\left(-\frac{2\pi \mathrm{i}\, q}{N}\right) \exp\left(-\frac{2\pi \mathrm{i}\, p}{3}\right)\,. \end{aligned}$$

Finally, for $j = 3m + 2$, we have

$$\begin{aligned} f_j \exp\left(-\frac{2\pi \mathrm{i}\, j\,(q + p\cdot(N/3))}{N}\right) &= f_{3m+2}\, \exp\left(-\frac{2\pi \mathrm{i}\,(3m+2)\,(q + p\cdot(N/3))}{N}\right) \\ &= f_{3m+2}\, \exp\left(-\frac{2\pi \mathrm{i}\, m\, q}{N/3}\right) \exp\left(-\frac{4\pi \mathrm{i}\, q}{N}\right) \exp\left(-\frac{4\pi \mathrm{i}\, p}{3}\right)\,. \end{aligned}$$

In view of these different cases, we define $f_{0,m} = f_{3m}$, $f_{1,m} = f_{3m+1}$, and $f_{2,m} = f_{3m+2}$ so that we may write Equation (3.5) as

$$\begin{aligned} F_{q+p\,\cdot\,(N/3)} = &\sum_{m=0}^{(N/3)-1} f_{0,m} \exp\left(-\frac{2\pi\mathrm{i}\,m\,q}{N/3}\right) \\ &+ \exp\left(-\frac{2\pi\mathrm{i}\,q}{N}\right) \exp\left(-\frac{2\pi\mathrm{i}\,p}{3}\right) \sum_{m=0}^{(N/3)-1} f_{1,m} \exp\left(-\frac{2\pi\mathrm{i}\,m\,q}{N/3}\right) \\ &+ \exp\left(-\frac{4\pi\mathrm{i}\,q}{N}\right) \exp\left(-\frac{4\pi\mathrm{i}\,p}{3}\right) \sum_{m=0}^{(N/3)-1} f_{2,m} \exp\left(-\frac{2\pi\mathrm{i}\,m\,q}{N/3}\right) \end{aligned} \tag{3.6}$$

With Equation (3.6), we see that we recursively compute the Fourier transform of data with length $N/3$, with length 3 as the base case. We provide the recursive pseudocode below, in Algorithm 2; however, the implementation in Tasmanian does not use recursive function calls. Note that the twiddle factors

$$\exp\left(-\frac{2\pi\mathrm{i}\,p}{3}\right)$$

are merely $\exp\left(\pm\frac{2\pi\mathrm{i}}{3}\right)$, but we have left them in the form of Equation (3.6) to make the correspondence clear. For $d > 1$, we perform a sequence of FFTs in each domain dimension as our multi-dimensional FFT. The final step is to notice that the exponents in φ_j are not j (as in Equation (3.4)), but rather $\sigma(j)$, so we use an indexing map to maintain consistency of indices.

Now we turn our attention to evaluating the complex exponentials $\varphi_{\boldsymbol{j}}(\boldsymbol{x})$ in Equation (2.66) for some evaluation point $\boldsymbol{x} \in D \subset \mathbb{R}^d$. In C++, the naïve way is to use `std::exp` (overloaded by the `complex` library). However, this approach is approximately 30 times more expensive than the addition operation. For context, `std::sin()` and `std::cos()` are roughly 10 times more expensive than addition.[6] Accordingly, we avoid direct calls of `std::exp()` by evaluating only the one-dimensional $\varphi_2(x_k) = \exp(2\pi\mathrm{i}\,x_k)$ and then iteratively conjugating and multiplying. We describe this algorithm using pseudocode in Algorithm 3. After computing the cache of one-dimensional functions $\varphi_{j_k}(x_k)$ for each dimension $1 \le k \le d$, we then multiply together the relevant one-dimensional functions to evaluate a general multi-dimensional complex exponential $\varphi_{\boldsymbol{j}}(\boldsymbol{x})$.

[6]These statistics come from running 100,000,000 of a given operation against a benchmark of simple variable assignment. The computing environment is a 2012 MacBook Pro running macOS Sierra wtih 8GB of RAM and a 2.9 GHz dual-core Intel Core i7 processor.

Algorithm 2 1D FFT with $N = 3^l$ points

```
function FFT_1D(data)
    $N$ = length(data)
    if $N == 1$ then
        return data
    end if
    for $j = 0 : (N/3 - 1)$ do
        data_mod0($j$) = data($3j$)
        data_mod1($j$) = data($3j + 1$)
        data_mod2($j$) = data($3j + 2$)
    end for
    out_mod0 = FFT_1D(data_mod0)
    out_mod1 = FFT_1D(data_mod1)
    out_mod2 = FFT_1D(data_mod2)
    for $j = 0 : (N/3 - 1)$ do
        out($j$) = out_mod0($j$) + out_mod1($j$) + out_mod2($j$)
        out($j + N/3$) = out_mod0($j$) + $\exp\left(-\frac{2\pi \mathrm{i} j}{N}\right) \exp\left(-\frac{2\pi \mathrm{i}}{3}\right)$ out_mod1($j$)
                         + $\exp\left(-\frac{4\pi \mathrm{i} j}{N}\right) \exp\left(-\frac{4\pi \mathrm{i}}{3}\right)$ out_mod2($j$)
        out($j + 2N/3$) = out_mod0($j$) + $\exp\left(-\frac{2\pi \mathrm{i} j}{N}\right) \exp\left(-\frac{4\pi \mathrm{i}}{3}\right)$ out_mod1($j$)
                         + $\exp\left(-\frac{4\pi \mathrm{i} j}{N}\right) \exp\left(-\frac{8\pi \mathrm{i}}{3}\right)$ out_mod2($j$)
    end for
    return out
end function
```

Algorithm 3 Evaluating complex exponential basis functions

```
function BUILD_CACHE($\boldsymbol{x}$)
    for $k = 1 : d$ do
        max_levels($k$) = $\max_{\boldsymbol{i} \in \Theta} i_k$
        cache($k$,1) = 1
        cache($k, 2$) = std::complex<$\cos(2\pi x_k), \sin(2\pi x_k)$>
        cache($k, 3$) = std::conj(cache($k$,2))
        for $j = 2 : ($getNumPoints(max_levels($k$))$ - 1)/2$ do
            cache($k$, $2j$) = cache($k, 2$) * cache($k, 2j - 2$)
            cache($k$, $2j + 1$) = cache($k, 3$) * cache($k, 2j - 1$)
        end for
    end for
    return cache
end function
```

CHAPTER

4 QUANTUM CHEMISTRY

The behavior of particles is entirely different at the quantum-mechanical level than at the classical level. When particles are so small as to be treated with quantum mechanics, measuring a system becomes complicated because the mere act of measuring is a disturbance. To find a physically measurable quantity α, we use the corresponding operator $\hat{A}$ and solve the eigenvalue problem

$$\hat{A}\,\psi = \alpha\,\psi\,. \tag{4.1}$$

We call the self-adjoint operator $\hat{A}$ an *observable*, the eigenvalue $\alpha \in \mathbb{R}$ a *measurement*, and the eigenfunction ψ the *wavefunction*. The probability density function of finding a particle of state ψ at a particular location $\boldsymbol{x}$ is $|\psi(\boldsymbol{x})|^2$. Though some observables, like position and momentum, have uncountably many eigenvalues, there are some important observables for which Equation (4.1) has only countably many eigenvalues. This important distinction between quantum and classical mechanics means that certain quantum measurements cannot take on a continuum of values. In fact, one of the most important operators in quantum chemistry is the molecular Hamiltonian $\hat{\mathcal{H}}$, which is the observable for energy and has only countably many eigenvalues.

In this chapter, we will review the background chemistry underlying our potential energy surfaces. We will begin with the general Schrödinger equation for $\hat{\mathcal{H}}$ and describe the example of the one-dimensional particle in a box (PIB). Then we will describe two techniques of approximating the energy of a system (the variational method and perturbation theory) and apply them to the PIB. After that, we will solve for the energies and wavefunctions of a hydrogenlike atom. Finally, we conclude with methods of approximating the wavefunctions and energies of a many-electron system.

4.1 Schrödinger equation

The time-independent Schrödinger equation for a system is

$$\hat{\mathcal{H}}\psi = E\psi \tag{4.2}$$

where

$$\hat{\mathcal{H}} = \hat{T} + \hat{V}\,.$$

Here, $\hat{T}$ is the kinetic energy term, and $\hat{V}$ is the potential energy term. $\hat{V} = V(\boldsymbol{x})$ will vary from problem to problem, but it is worthwhile to express $\hat{T}$ in terms of $\boldsymbol{x}$.

In one-dimensional classical mechanics,

$$T = \frac{1}{2}mv^2 = \frac{p^2}{2m}$$

where m is the mass of the particle, and p is the momentum. Analogously, in quantum mechanics, we have

$$\hat{T} = \frac{\hat{\boldsymbol{p}} \cdot \hat{\boldsymbol{p}}}{2m}$$

where

$$\hat{\boldsymbol{p}} = -\mathrm{i}\hbar\nabla$$

is the momentum operator, $\mathrm{i} = \sqrt{-1}$, and

$$\hbar = \frac{h}{2\pi} = \frac{6.626 \times 10^{-34}\ \mathrm{J \cdot s}}{2\pi}$$

is the reduced Planck constant. Therefore,

$$\hat{T} = -\frac{\hbar^2}{2m}\nabla^2,$$

so we may write $\hat{\mathcal{H}}$ in the more useful form

$$\hat{\mathcal{H}} = -\frac{\hbar^2}{2m}\nabla^2 + V(\boldsymbol{x})\,.$$

Since $|\psi(\boldsymbol{x})|^2$ is a probability distribution function, the wavefunction ψ must be normalized.

That is, in bra-ket notation,

$$\langle\psi|\psi\rangle = 1$$

where $\langle f|g\rangle$ denotes the L^2 inner product over the entire (possibly multi-dimensional) domain of ψ. We also define the matrix element of the operator $\hat{A}$:

$$\langle f|\hat{A}|g\rangle = \langle f|\hat{A}g\rangle\,.$$

4.2 Example systems

We will now present the Hamiltonians for commonly studied systems. The first two of these systems (particle in a box and hydrogenlike atom) are exactly solvable. The last is the Hamiltonian for a molecule, which must be solved approximately.

4.2.1 Exactly solvable problems

4.2.1.1 Particle in a box

We will now work through the particle-in-a-box (PIB) system to illustrate how to solve for the energies E and wavefunctions ψ. The PIB is often one of the first examples covered in quantum texts due to its simplicity (cf. Levine in [120]). We are doing this example to show the step-by-step process of solving the Schrödinger equation, and will use the PIB to illustrate approximation methods in quantum mechanics due to the PIB's simplicity.

Consider the particle confined to the potential well

$$V(x) = \begin{cases} 0, & 0 < x < L \\ \infty, & \text{otherwise} \end{cases}.$$

We will solve the Schrödinger equation for this system

$$-\frac{\hbar^2}{2m}\frac{d^2\psi}{dx^2} + V(x)\psi(x) = E\psi(x)\,.$$

Now since V is infinite outside $(0, L)$ and the particle's probability density function (pdf) for position is $|\psi(x)|^2$, then the particle will be confined to $(0, L)$ and zero outside. So the system becomes

$$\begin{cases} -\frac{\hbar^2}{2m}\psi'' = E\psi(x), & 0 < x < L \\ \psi(0) = \psi(L) = 0 \end{cases}.$$

From now on in this example, we will restrict ψ to $[0, L]$. The solutions of this system are of the form

$$\psi(x) = c_1 \cos(\omega x) + c_2 \sin(\omega x), \qquad \omega = \sqrt{\frac{2mE}{\hbar^2}}\,,$$

where c_1 and c_2 are constants. Enforcing boundary conditions gives $c_1 = 0$ and

$$\sin(\omega L) = 0,$$

which implies

$$\omega L = n\pi, \qquad n = 1, 2, \ldots,$$

which gives

$$E_n = \frac{\hbar^2 n^2 \pi^2}{2mL^2} = \frac{h^2 n^2}{8mL^2}\,. \tag{4.3}$$

The wavefunction corresponding to E_n is

$$\psi_n(x) = C_n \sin\left(\frac{n\pi x}{L}\right)\,.$$

To normalize, we enforce

$$\begin{aligned} 1 &= \langle \psi | \psi \rangle \\ &= C_n^2 \int_0^L \sin^2\left(\frac{n\pi x}{L}\right)\,dx \\ &= C_n^2 \int_0^L \left(\frac{1}{2} - \frac{1}{2}\cos\left(\frac{2n\pi x}{L}\right)\right)\,dx \\ &= C_n^2\,\frac{L}{2} \end{aligned}$$

so $C_n = \sqrt{2/L}$, and therefore

$$\psi_n(x) = \sqrt{\frac{2}{L}}\,\sin\left(\frac{n\pi x}{L}\right)\,.$$

Figure 4.1 shows the wavefunctions and moduli for the three lowest-lying states with $L = 1$. Note that there are certain regions of space where there is zero probability of finding the particle; furthermore, this location of zero probability changes with n.

4.2.1.2 Hydrogen-like atoms

In this section, we will solve the Schrödinger equation for hydrogen-like atoms, which have a nucleus (with proton number Z) orbited by a single electron. Our approach follows that of Levine in [120]. Solving this system will give the electronic energy as well as a probability distribution function for electronic position. Due to the rotational motion of the electron, spherical polar

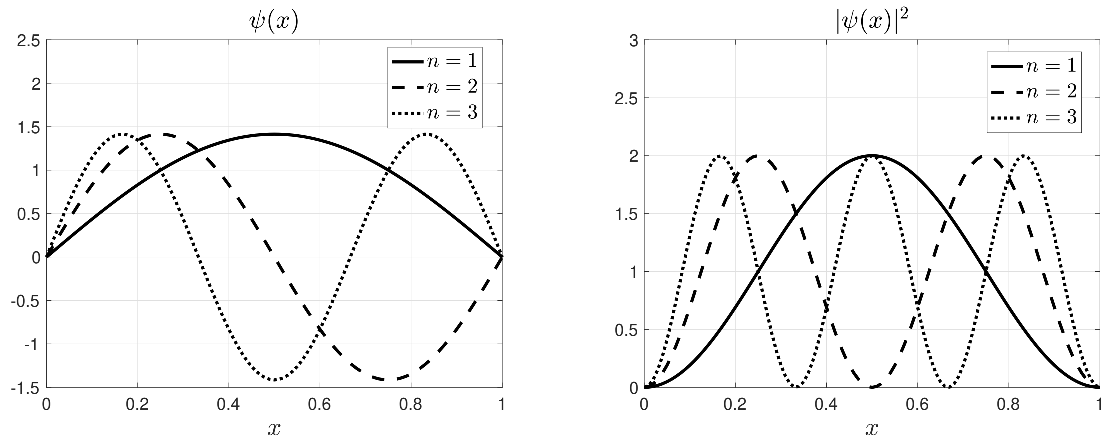


Figure 4.1 PIB solutions for $n = 1, 2, 3$.

coordinates is the natural choice to represent the system. The conversion between spherical polar coordinates and Cartesian coordinates is

$$\begin{cases} x = r \sin\theta \cos\phi \\ y = r \sin\theta \sin\phi \\ z = r \cos\theta \end{cases}$$

In other words, $r^2 = x^2 + y^2 + z^2$, θ is the angle that the vector $\boldsymbol{r}$ makes with the z-axis, and ϕ is the angle that the projection of $\boldsymbol{r}$ onto the xy-plane makes with the x-axis.

The Hamiltonian for a hydrogen-like atom is

$$\hat{\mathcal{H}} = -\frac{\hbar^2}{2\mu}\nabla^2 - \frac{Ze'^2}{4\pi\varepsilon_0 r} \tag{4.4}$$

where ∇^2 is the Laplacian, $e' = 1.60 \times 10^{-19}$ is the charge of an electron in coulombs, ε_0 is the permittivity of free space (a physical constant contained in Coulomb's law), and

$$\mu = \frac{m_e\, m_p}{m_e + m_p}$$

is the reduced mass of the system. Because the mass of a proton is quite large compared to the mass of an electron, it is standard for quantum-chemistry texts to assume $\mu \approx m_e$ [120].

Levine in [120] shows that in spherical polar coordinates,

$$\nabla^2 = \frac{\partial^2}{\partial r^2} + \frac{2}{r}\frac{\partial}{\partial r} + \frac{1}{r^2}\left(\frac{\partial^2}{\partial \theta^2} + \cot\theta \frac{\partial}{\partial \theta} + \frac{1}{\sin^2\theta}\frac{\partial^2}{\partial \phi^2}\right)$$

At this point, we introduce the angular-momentum operator $\hat{L}^2$, defined as

$$\hat{L}^2 = -\hbar^2 \left(\frac{\partial^2}{\partial \theta^2} + \cot\theta \frac{\partial}{\partial \theta} + \frac{1}{\sin^2\theta} \frac{\partial^2}{\partial \phi^2} \right)$$

Finding the eigenvalues and eigenfunctions of $\hat{L}^2$ is tedious and omitted here (see Levine in [120]), but it is a building block toward solving the hydrogenlike atom. The normalized eigenfunctions $Y_{l,m}(\theta, \phi)$ are given by

$$Y_{l,m}(\theta,\phi) = \sqrt{\frac{2l+1}{4\pi} \frac{(l-|m|)!}{(l+|m|)!}}\; P_l^{|m|}(\cos\theta)\, \exp(\mathrm{i}m\phi), \qquad \begin{cases} l &= 0, 1, 2, \ldots \\ m &= -l, -l+1, \ldots, l-1, l \end{cases}$$

where the associated Legendre polynomials $P_l^{|m|}$ are

$$P_l^{|m|}(w) = \frac{1}{2^l\; l!} (1-w^2)^{|m|/2} \frac{d^{l+|m|}}{dw^{l+|m|}} \left[(w^2-1)^l \right].$$

The eigenvalues are

$$\hat{L}^2\, Y_{l,m} = \hbar^2 l(l+1)\, Y_{l,m}\,.$$

Since the potential V is a function of r only and $\hat{L}^2$ gives us the eigenfunctions for the angular part of ∇^2, then we may take the wavefunction ψ as

$$\psi(r,\theta,\phi) = R(r)\, Y_{l,m}(\theta,\phi)$$

where R has yet to be determined. The eigenvalue equation

$$\hat{\mathcal{H}}\psi = E\psi$$

yields

$$-\frac{\hbar^2}{2m_e} \left(R''(r) + \frac{2R'(r)}{r} - \frac{l(l+1)}{r^2} R(r) \right) - \frac{Z\, e'^2}{4\pi\varepsilon_0 r} R(r) = E\, R(r)\,.$$

Equivalently, we have

$$R''(r) + \frac{2R'(r)}{r} + \left(\frac{2m_e\, E}{\hbar^2} + \frac{2Z}{a_0 r} - \frac{l(l+1)}{r^2} \right) R(r) = 0 \tag{4.5}$$

where $a_0 = 4\pi\varepsilon_0\hbar^2/(m_e e'^2)$ is a constant known as the *Bohr radius*.

Solving (4.5) is tedious, so we present the solution here and refer the reader to Levine [120] for the details. We have

$$R(r) = R_{nl}(r) = r^l \exp\left(-\frac{Zr}{a_0 n}\right) \sum_{j=0}^{n-l-1} b_j\, r^j \ .$$

From the upper limit of the sum, we have $l \le n-1$. The coefficients b_j are given by

$$b_{j+1} = \frac{2Z}{a_0 n}\, \frac{j-(n-l-1)}{(j+1)(j+2l+2)} b_j \ .$$

So we have

$$\psi_{nlm}(r,\theta,\phi) = R_{nl}(r)\, Y_{lm}(\theta,\phi) = r^l \exp\left(-\frac{Zr}{a_0 n}\right) \sum_{j=0}^{n-l-1} b_j\, r^j\ Y_{lm}(\theta,\phi) \ . \tag{4.6}$$

where the quantum numbers n, l, m satisfy

$$\begin{cases} n & = 1, 2, \dots \\ l & = 0, 1, \dots, n-1 \\ m & = -l, -l+1, \dots, l-1, l \end{cases} .$$

The principal quantum number n is called the *shell*, the angular momentum number l is the *subshell*, and the magnetic quantum number m_l determines the number of spatial orbitals as well as their orientation in space.

For completeness, we find b_0 by enforcing normalization conditions on the ground state. Since $Y_{l,m}$ are normalized, we integrate over r in spherical coordiantes to obtain

$$\begin{aligned} 1 &= \langle \psi_{100} | \psi_{100} \rangle \\ &= b_0^2 \int_0^\infty \exp(-2Zr/a_0) r^2\, dr \ \int_{-1}^{1} \int_0^{2\pi} Y_{lm}(\theta,\phi)\, d\phi\, d(\cos\theta) \\ &= b_0^2 \left(\frac{a_0}{2Z}\right)^3 \int_0^\infty e^{-u} u^2\, du \\ &= b_0^2 \left(\frac{1}{4}\right) \left(\frac{a_0}{Z}\right)^3 \end{aligned}$$

so $b_0 = 2(Z/a_0)^{3/2}$.

The energies associated with ψ_{nlm} depend only on n and are given by

$$E_n = -\frac{\hbar^2 Z^2}{2m\, a_0^2\, n^2} \ . \tag{4.7}$$

For each n, we have $l \in \{0, \dots, n-1\}$, and m can take on $2l+1$ possible values, so we have

$$\sum_{l=0}^{n-1} 2l + 1 = n^2$$

linearly independent wavefunctions corresponding to E_n. Energy levels with two or more linearly independent wavefunctions are called *degenerate*.

4.2.2 Molecules

Molecules are chemical systems composed of atoms that are bonded together. The arrangement of the nuclei in a molecule can be described in Cartesian coordinates, but it is more useful to describe the geometry in terms of internal coordinates [163]: bond lengths, bond angles, and dihedral angles. Bond lengths are the distances between two bonded atoms and are customarily measured in angstroms (Å).[1] A bond angle is the angle formed between a given atom and two atoms adjacent to it. A dihedral angle is the angle made by an atom with the plane specified by an additional three angles. Therefore, for a general molecule with N atoms, there will be $N-1$ bond lengths, $N-2$ bond angles, and $N-3$ dihedral angles, for a total of $3N-6$ internal coordinates. These coordinates are formatted into a table called a Z-matrix.

For molecules, the full Hamiltonian is

$$\hat{\mathcal{H}}_{full} = \hat{T}_{el} + \hat{T}_{nuc} + V_{el-el} + V_{el-nuc} + V_{nuc-nuc}$$

where

$$\begin{aligned}
\hat{T}_{el} &= -\sum_{i=1}^{N_{elec}} \frac{\hbar^2}{2m_e}\nabla_i^2\,, && \text{(electron kinetic energy)}\\
\hat{T}_{nuc} &= -\sum_{A=1}^{N_{atom}} \frac{\hbar^2}{2m_A}\nabla_A^2\,, && \text{(nuclear kinetic energy)}\\
V_{el-el} &= \sum_{i=1}^{N_{elec}}\sum_{j>i}^{N_{elec}} \frac{e'^2}{4\pi\varepsilon_0 r_{ij}^2}\,, && \text{(electron–electron repulsion)}\\
V_{el-nuc} &= \sum_{A=1}^{N_{atom}}\sum_{i=1}^{N_{elec}} \frac{Z_A e'^2}{4\pi\varepsilon_0 r_{Ai}^2}\,, && \text{(electron-nuclear attraction)}\\
V_{nuc-nuc} &= \sum_{A=1}^{N_{atom}}\sum_{B>A}^{N_{atom}} \frac{Z_A Z_B e'^2}{4\pi\varepsilon_0 r_{AB}^2}\,, && \text{(nuclear-nuclear repulsion)}
\end{aligned}$$

where r is the Euclidean distance, (i, j) are electronic indices, and (A, B) are nuclear indices.

[1] 1 Å$= 10^{-10}$ m.

To simplify the molecular Hamiltonian, quantum chemistry uses the well-established Born–Oppenheimer approximation [27], which assumes (i) that the molecular wavefunction is a product of electronic and nuclear wavefunctions

$$\psi = \psi^{elec}\psi^{nuc}$$

and (ii) that the nuclei are fixed. With this assumption, $\hat{T}_{nuc} = 0$, and $V_{nuc-nuc}$ is a fixed quantity of energy, $E_{nuc-nuc}$. It remains to solve

$$\hat{\mathcal{H}}\,\psi_{elec} = (\hat{T}_{el} + V_{el-el} + V_{el-nuc})\,\psi_{elec} = E_{elec}\,\psi_{elec} \tag{4.8}$$

and at the end take

$$E = E_{nuc-nuc} + E_{elec}\,.$$

4.3 Approximating solutions of the Schrödinger equation

For many systems, the Schrödinger equation is difficult, if not impossible, to solve in closed form. In this section, we will present techniques to approximate the wavefunctions and energies of a system.

4.3.1 Variational method

The variational method seeks to approximate the ground-state energy of a system using a trial function $\varphi(\boldsymbol{x})$. We define

$$\tilde{E} = \frac{\langle\varphi|\hat{\mathcal{H}}|\varphi\rangle}{\langle\varphi|\varphi\rangle} \tag{4.9}$$

where

$$\langle\varphi|\hat{\mathcal{H}}|\varphi\rangle = \int_D \varphi^*(\boldsymbol{x})\hat{\mathcal{H}}\varphi(\boldsymbol{x})\,d\boldsymbol{x}$$

and D is the wavefunction's whole domain. The power of the variational method is due to the following theorem.

Theorem 10. *(Variational Principle) Let $\hat{\mathcal{H}}$ be a Hamiltonian with eigenvalues $E_1 \leq E_2 \leq \cdots$. For any choice of trial functions φ satisfying the boundary conditions of the problem,*

$$\tilde{E} \geq E_1\,.$$

Proof. Let φ be an arbitrary trial function satisfying the boundary conditions of $\hat{\mathcal{H}}$. We expand

φ in terms of the orthonormal basis of eigenfunctions $\{\psi_k\}_{k\geq 1}$ of $\hat{\mathcal{H}}$:

$$\varphi(\boldsymbol{x}) = \sum_{k=1}^{\infty} c_k \psi_k(\boldsymbol{x})\,.$$

This gives

$$\begin{aligned}\langle\varphi|\hat{\mathcal{H}}|\varphi\rangle &= \sum_{j=1}^{\infty}\sum_{k=1}^{\infty} c_j^* E_k c_k \langle\psi_j|\psi_k\rangle \\ &= \sum_{k=1}^{\infty} E_k |c_k|^2 \\ &\geq E_1 \sum_{k=1}^{\infty} |c_k|^2\,.\end{aligned}$$

But we also have

$$\langle\varphi|\varphi\rangle = \sum_{j=1}^{\infty}\sum_{k=1}^{\infty} c_j^* c_k \langle\psi_j|\psi_k\rangle = \sum_{k=1}^{\infty} |c_k|^2\,,$$

which gives $\tilde{E} \geq E_1$. □

Since $\tilde{E} \geq E_1$, then the variational method gives an upper bound on the ground-state energy. Whether or not this bound is useful depends on how well the trial function φ approximates the true wavefunction ψ_1. We will illustrate the power of the variational method in the following example.

Example 5. *We will estimate the ground-state energy of the PIB from Sec. 4.2.1.1 using the variational method. From Figure 4.1, we note that the ground-state wavefunction appears parabolic, opening downward, with zeros at $x = 0, x = L$. So we take*

$$\varphi(x) = x(L-x)\,.$$

We compute

$$\begin{aligned}\langle\varphi|\hat{\mathcal{H}}|\varphi\rangle &= -\frac{\hbar^2}{2m}\int_0^L x(L-x)\frac{d^2}{dx^2}[x(L-x)]\,dx \\ &= \frac{\hbar^2}{m}\int_0^L x(L-x)\,dx \\ &= \frac{\hbar^2 L^3}{6m}\end{aligned}$$

and

$$\begin{aligned}\langle\varphi|\varphi\rangle &= \int_0^L x^2(L-x)^2\,dx \\ &= \int_0^L x^2(x^2-2Lx+L^2)\,dx \\ &= \frac{L^5}{5}-\frac{L^5}{2}+\frac{L^5}{3} \\ &= \frac{L^5}{30}\end{aligned}$$

so

$$\tilde{E} = \frac{30\hbar^2 L^3}{6mL^5} = \frac{5}{4\pi^2}\frac{h^2}{mL^2} \approx 0.12665\frac{h^2}{mL^2}\,.$$

Since $E_1 = \frac{h^2}{8mL^2}$ *for the PIB by Equation* (4.3)*, then this trial function gives an energy estimate that is accurate to two significant figures.*

4.3.2 Basis sets

One straightforward application of the variational method is to parameterize the trial function φ as a linear combination of known functions χ_p:

$$\varphi(\boldsymbol{x}; c_1,\ldots,c_M) = \sum_{p=1}^{M} c_p\chi_p(\boldsymbol{x})\,.$$

Now we regard $\tilde{E}$ as a function of the weights c_p. We can find the optimal weights c_i by setting

$$\frac{\partial\tilde{E}}{\partial c_p} = 0, \qquad p = 1,2,\ldots,M\,.$$

The set $\{\chi_p\}_{p=1}^M$ is known as a *basis set.* Basis sets are ubiquitous in computational chemistry, and there are many different choices available. In this section, we will discuss some of the available basis sets to approximate the orbital wavefunctions of atoms and molecules.

Since we are approximating the wavefunctions of hydrogenlike atoms, a natural choice of χ would be functions having the same shape as ψ_{nlm} in Equation (4.6):

$$\chi^{STO}(r,\theta,\phi) = C\,r^{n-1}e^{-\zeta r}\,Y_{lm}(\theta,\phi) \tag{4.10}$$

where $\zeta \in \mathbb{R}^+$ and C is a normalization constant. This function is called a *Slater-type orbital (STO)* [120]. In practice, a few different χ are used to form a basis set, each with a different ζ, and the choice of ζ is determined by fitting (4.10) to known orbital shapes. Importantly, the choice of ζ depends on both the atom and orbital under consideration. While STOs work well for many-electron atomic calculations [120], they are not well-suited for molecular calculations with multiple nuclei. This is because one must compute inner products $\langle \chi_p | \chi_q \rangle$, where the basis functions can be centered on different atoms.

Gaussian-type functions (GTFs) are much better suited to calculating integrals. In practice, these basis sets are made up of *contracted Gaussians*, which are linear combinations of so-called *primitive Gaussians*:

$$\chi_q^{prim}(x,y,z) = C\, x^i y^j z^k \exp\left(-\zeta_q \left(x^2+y^2+z^2\right)\right), \tag{4.11}$$

$$\chi^{contr}(x,y,z) = \sum_{q=1}^{N_{prim}} d_q \chi_q^{prim}(x,y,z), \tag{4.12}$$

where $i,j,k \in \mathbb{N}_0$, $\zeta_q \in \mathbb{R}^+$, and C is a normalization constant. The exponents i, j, and k are chosen based on shape of the orbital under consideration [120]. For example, the 2p_x orbital would have $i = 1$, $j = 0$, $k = 0$. The orbital exponent ζ_q and weights d_q are optimized against known orbital shapes for particular atoms. In molecules, each χ^{contr} is centered on a particular atom [120]. The inner product $\langle \chi_A^{contr} | \chi_B^{contr} \rangle$, where the contracted Gaussians are centered on nuclei A and B, is much easier to compute with GTFs than with STOs. Indeed, numerous *ab initio* chemistry packages use GTFs, such as Gaussian 16 [73], NWChem [217], and Orca [149, 150].

One commonly encountered flavor of GTF basis set is the so-called *Pople basis set*, arising out of the group of Nobel laureate John Pople. These basis sets are named in a systematic manner, which we will describe in Example 6. The following Pople basis sets are available in Gaussian 16, along with citations to the original papers describing the basis sets:

- 3-21G [23],
- 6-21G [23],
- 4-31G [58],
- 6-31G [89],
- 6-311G [112].

Additionally, Pople basis sets may include polarization and diffuse functions [120]. If the maximum orbital angular momentum number in the valence shell is l, a polarization function is a basis function corresponding to an $(l+1)$-type orbital.[2] Diffuse functions are basis functions with small orbital exponents ζ, which are useful to desribe systems where electrons may be quite

[2]For example, a polarization function for a hydrogen atom ($l = 0$) would be a p-orbital-type function ($l = 1$).

far from the nucleus (e.g., anions). The presence of polarization functions is denoted in the set by one asterisk (*) after the "G" for atoms with $Z \geq 3$ and by two asterisks (**) if every atom has polarization functions. The presence of diffuse functions follows similar rules but uses "+" before the "G".

Example 6. *Consider the Pople basis set 6-311+G*. The first piece of information is how many numbers there are before and after the hyphen. The amount of numbers before the hyphen tells how many contracted Gaussians describe each* **core** *orbital; the amount of numbers after the hyphen gives the number of contracted Gaussians for each* **valence** *orbital. Thus, for 6-311G, there is one basis function for each core orbital and three for each valence orbital. The values of the numbers numbers tell how many primitive Gaussians go into each contracted Gaussian function in the basis set. So the contracted Gaussian on each core orbital is composed of six primitives, and the three contracted Gaussians on each valence orbital are composed of three, one, and one primitives respectively. There are diffuse and polarization functions, but only on atoms with $Z \geq 3$.*

4.3.3 Slater determinants

For a system with N_{elec} electrons, it would be tempting to take

$$\Psi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{N_{elec}}) \overset{?}{=} \prod_{k=1}^{N_{elec}} \psi_k(\boldsymbol{x}_k)$$

where ψ_k is the 3D wavefunction for the orbital occupied by electron k. However, this choice is entirely incorrect as it ignores the effects of spin.[3] Electrons have half-integer spin, obeying

$$m_s = \pm\frac{1}{2}\,.$$

Customarily, α denotes spin-up ($m_s = \frac{1}{2}$) and β spin-down ($m_s = -\frac{1}{2}$). The Pauli exclusion principle [162] states that no two electrons in the same orbital can have the same spin, which means mathematically that wavefunctions of many-electron systems must be antisymmetric under particle exchange. That is,

$$\Psi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i, \ldots, \boldsymbol{x}_j, \ldots, \boldsymbol{x}_{N_{elec}}) = -\Psi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_j, \ldots, \boldsymbol{x}_i, \ldots, \boldsymbol{x}_{N_{elec}})$$

We express each spin-orbital ϕ_i as the product of a spatial part φ and a spin part σ:

$$\phi_i(\boldsymbol{x}) = \varphi_i(\boldsymbol{x})\sigma_i\,, \qquad k = 1, 2, \ldots, N_{elec}\,,$$

[3]Spin is also known as *intrinsic angular momentum* and is a feature of quantum mechanics.

where σ_i can either be α or β. Two electrons can have the same spatial orbital $\varphi_i(\boldsymbol{x})$ as long as they have opposite spins. To ensure antisymmetry, Slater first proposed in 1929 [191] to take

$$\Psi(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{N_{elec}}) = \frac{1}{\sqrt{N_{elec}!}} \begin{vmatrix} \phi_1(\boldsymbol{x}_1) & \phi_2(\boldsymbol{x}_1) & \cdots & \phi_{N_{elec}}(\boldsymbol{x}_1) \\ \phi_1(\boldsymbol{x}_2) & \phi_2(\boldsymbol{x}_2) & \cdots & \phi_{N_{elec}}(\boldsymbol{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(\boldsymbol{x}_{N_{elec}}) & \phi_2(\boldsymbol{x}_{N_{elec}}) & \ldots & \phi_{N_{elec}}(\boldsymbol{x}_{N_{elec}}) \end{vmatrix} . \tag{4.13}$$

The right-hand side of Equation (4.13) is known as a *Slater determinant.* By the properties of determinants, a Slater determinant is guaranteed to be antisymmetric under electron exchange. Many modern computational chemistry methodologies express the spatial part of each orbital in Equation (4.13) as

$$\varphi_i(\boldsymbol{x}) = \sum_{\mu=1}^{M} c_{\mu i}\, \chi_\mu(\boldsymbol{x})$$

and find the expansion coefficients $c_{\mu i}$ using the variational method (see Secs. 4.3.1 and 4.3.2).

4.3.4 Density functional theory (DFT)

In 1964, Hohenberg and Kohn proved two important theorems in quantum chemistry [92]. Their first theorem states that the electron density

$$\rho(\boldsymbol{x}) = N_{elec} \sum_{s_1} \cdots \sum_{s_{N_{elec}}} \int_{\mathbb{R}^3} \cdots \int_{\mathbb{R}^3} |\Psi(\boldsymbol{x}, s_1, \ldots, \boldsymbol{x}_{N_{elec}}, s_{N_{elec}})|^2 \, d\boldsymbol{x}_2 \cdots d\boldsymbol{x}_{N_{elec}},$$

where s_i are the spin states associated with spatial orbital i, uniquely determines the Hamiltonian and the properties of the system. That is, it is not necessary in principle to find Ψ (a function of $3N_{atom} - 6$ variables), but only ρ (a function of three variables). The second Hohenberg–Kohn theorem states that there exists an energy functional $E[\rho]$ such that

$$E[\rho] \geq E_0$$

where E_0 is the true ground-state energy, with equality achieved if and only if $\rho(\boldsymbol{x})$ is the true ground-state electron density.

In 1965, Kohn and Sham [108] expressed $E[\rho]$ as

$$E[\rho] = T_S[\rho] + J[\rho] + E_{XC}[\rho] + E_{nuc-nuc} \tag{4.14}$$

where $T_S[\rho]$ is the energy of a fictitious noninteracting system (to be described shortly), $J[\rho]$ is the energy of electron–electron repulsion, and $E_{XC}[\rho]$ is the exchange–correlation term. The exact form of $E_{XC}[\rho]$ is unknown, and there are many different choices of functionals within the Kohn–Sham DFT framework.

Kohn and Sham approximated the wavefunction of the noninteracting system as a Slater determinant of fictitious "Kohn–Sham" orbitals ϕ_i [108]. Each Kohn–Sham orbital has the form

$$\phi_i(\boldsymbol{x}, s_i) = \varphi_i(\boldsymbol{x})\sigma(s_i),$$

where φ_i is the spatial orbital and $\sigma(s_i)$ is the spin component. The spatial part of each Kohn–Sham orbital satisfies[4]

$$\Bigg(-\frac{1}{2}\nabla^2 + \underbrace{\int_{\mathbb{R}^3} \frac{\rho(\boldsymbol{y})}{\|\boldsymbol{x}-\boldsymbol{y}\|}\, d\boldsymbol{y} + V_{XC}(\boldsymbol{x}) - \sum_{A=1}^{N_{atom}} \frac{Z_A}{\|\boldsymbol{x}-\boldsymbol{x}_A\|}}_{V_{KS}(\boldsymbol{x})} \Bigg)\varphi_i(\boldsymbol{x}) = \varepsilon_i\, \varphi_i(\boldsymbol{x}) \tag{4.15}$$

for $i = 1, 2, \ldots, N_{elec}$, where

$$V_{XC}(\boldsymbol{x}) = \frac{\partial E_{XC}[\rho]}{\partial \rho}\,,$$

and the ε_i are the Kohn–Sham orbital energies. Furthermore, Kohn and Sham showed [108] that *true* electron density of the ground state is

$$\rho(\boldsymbol{x}) = \sum_{i=1}^{N_{elec}} |\varphi_i(\boldsymbol{x})|^2\,,$$

where, to reiterate, the spatial parts of two electron orbitals can be identical as long as the electrons have opposite spin. We can now write some of the terms in the energy functional $E[\rho]$:

$$T_S[\rho] = -\frac{1}{2}\sum_{i=1}^{N_{orb}}\sum_{s_i}\langle\varphi_i|\nabla^2|\varphi_i\rangle,$$
$$J[\rho] = \frac{1}{2}\int_{\mathbb{R}^3}\int_{\mathbb{R}^3} \frac{\rho(\boldsymbol{x})\rho(\boldsymbol{y})}{\|\boldsymbol{x}-\boldsymbol{y}\|}\, d\boldsymbol{x}\, d\boldsymbol{y}\,.$$

One may find the Kohn–Sham orbitals using the self-consistent field (SCF) procedure [120]:

1. Start with an initial electron density $\tilde{\rho}$.

2. Express φ_i in terms of basis functions (Sec 4.3.2):

$$\varphi_i(\boldsymbol{x}) \approx \tilde{\varphi}_i(\boldsymbol{x}) = \sum_{\mu=1}^{M} c_{\mu i}\, \chi_\mu(\boldsymbol{x}).$$

[4]For the rest of this section, we will express equations in atomic units, where $\hbar = 1/(4\pi\varepsilon_0) = e' = m_e = 1$.

3. For each i, use the variational method to find the optimal coefficients $c_{\mu i}$ that minimize

$$\frac{\langle\tilde{\varphi}_i|\hat{h}_{KS}|\tilde{\varphi}_i\rangle}{\langle\tilde{\varphi}_i|\tilde{\varphi}_i\rangle}, \qquad \hat{h}_{KS} = -\frac{1}{2}\nabla^2 + V_{KS}(\boldsymbol{x}),$$

   where V_{KS} is built with the initial iterate $\tilde{\rho}$.

4. Update

$$\rho(\boldsymbol{x}) \leftarrow \sum_{i=1}^{N_{orb}} \sum_{s_i} |\tilde{\varphi}_i^{opt}(\boldsymbol{x}, s_i)|^2 .$$

5. Repeat steps 1–4 with $\tilde{\rho} \leftarrow \rho$ until the change in ρ is small.

Once this process has finished, we recover the energy using Equation (4.14).

CHAPTER

5

TRIGONOMETRIC APPROXIMATION OF POTENTIAL ENERGY SURFACES

In this chapter, we apply the sparse grid and Tasmanian machinery developed in Chapters 2 and 3 to the study of chemical systems. In this chapter, we use a sparse trigonometric interpolation basis to approximate the reduced-dimensional potential energy surface of a tungsten molecule. This chapter was originally published as an article in the *Journal of Physical Chemistry B* (2019), where Liu and Morrow were co-lead authors [141].[1] In Section 6, we construct a mixed-basis surrogate model that preserves periodicity on the rotational coordinates while using polynomial interpolation on bond lengths. We use this surrogate model to study NVE molecular dynamics of azomethane and compare to all-polynomial interpolation, the previous state-of-the-art as described in [146].

The potential energy surface (PES) of an electronic state of a chemical system is a function that maps the molecular geometry to the electronic energy within the Born–Oppenheimer approximation [26, 27]. Local structures of a PES, such as minima and saddle points, provide the geometry and energy information for stable and transition-state structures of a system. In addition, global features of PESs, which can be investigated by theoretical analyses [77, 145]

[1]Reprinted (adapted) with permission from [Morrow, Z., Liu, C., Kelley, C. T. and Jakubikova, E. "Approximating Periodic Potential Energy Surfaces Using Sparse Trigonometric Interpolation." *J. Phys. Chem. B* **123**.45 [2019], pp. 9677–9684. DOI: `10.1021/acs.jpcb.9b08210`]. Copyright 2019, American Chemical Society. ACS Articles on Request: `http://pubs.acs.org/articlesonrequest/AOR-IEFU7cTCymhZ2ac6hAzF`

and via molecular-dynamics simulations [2, 28, 33, 125, 146, 227], are useful for understanding chemical reactivity. In order to construct a PES efficiently, different methods have been developed, such as modified Shepard interpolation [42–44], permutationally invariant potential energy surface by linear least squares fitting [2, 28, 29, 33, 39, 178], neural network approaches [99, 130, 135], Gaussian process [48, 216], and the finite-element method [20–22]. Most of these techniques focus on constructing a full-dimensional PES, which treats the potential energy of an N-atom system as a function of $3N-6$ internal coordinates. Because the computational cost for constructing a PES increases rapidly with N, these full-dimensional methods are restricted to small molecules only (i.e., $N \leq 10$).

Fortunately, constructing a PES with all internal degrees of freedom (dofs) is not always necessary for studying the chemical reactivity of large systems with tens to hundreds of atoms. In many cases, only a small number of dofs, i.e., reaction coordinates (RCs), are essential for describing the system's chemical reactivity [25]. As a result, reduced-dimensional PESs with a small number of RCs have been widely employed to study various processes in large systems, such as the folding of polypeptides [57, 109, 161] and the intersystem crossing of transition-metal complexes [24, 145, 194].

Previous work from our group [144, 146] implemented the Smolyak sparse-grid interpolation algorithm [100, 192] to build the reduced-dimensional PESs, where the interpolation basis functions are Lagrange polynomials with the Clenshaw–Curtis points [41, 200]. This approach was shown to be efficient for both PES constructions and single-point energy evaluations [144, 146]. In addition, we developed a new molecular dynamics (MD) simulation method for reduced-dimensional PESs [125]. The new MD method relies on interpolated potential energy and coordinate functions, and their derivatives (first-order for energy function and second-order for coordinate functions) to solve the classical equations of motion in the Hamiltonian formalism [52, 115]. As a result, to generate smooth MD trajectories, the interpolated energy function must have continuous first derivatives, and the coordinate functions must have continuous second derivatives. These smoothness conditions apply to the whole domain, including the crossing of the periodic boundary.

The requirement for smoothness can be easily achieved within the domain for interpolation functions with polynomial basis. In one dimension, a polynomial of degree N on a closed interval $[a, b]$ is N times non-trivially continuously differentiable in the interior (a, b). When RCs only contain non-periodic coordinates, such as bond lengths, bending angles and normal coordinates, the MD trajectories are constrained within the interpolation domain because of the high potential energy barrier at the boundary. In those cases, the polynomial basis is capable of providing the desired smoothness.

On the other hand, internal rotations often play important roles in monomolecular reactions, such as the photoisomerization of polymers and biomolecules [126, 127, 228], hydrocarbon pericyclic reactions [93], and spin crossover of transition metal complexes [6, 177]. In many cases, one or more torsion angles are the primary reaction coordinates for describing the reaction process, while the remaining internal coordinates will only change slightly during the reaction to assist the primary reaction coordinates [121, 208, 224]. Reduced-dimensional PESs for these systems have an additional requirement for continuity: periodicity at boundaries. Polynomial interpolation will preserve periodicity of the underlying function values but gives no guarantees on the periodicity of the derivative. This causes the gradient of the surrogate PES to be discontinuous when an internal rotation crosses the periodic boundary. The left and right derivatives exist on either side, but they do not match. When a coordinate crosses the periodic boundary during an MD simulation, both kinetic energy and generalized forces will change suddenly with the discontinuous gradient [125], leading to an incorrect MD result. Thus, an interpolation algorithm with other basis functions is necessary for modeling reactions with periodic coordinates.

In this chapter, we describe a new sparse-grid interpolation method with a trigonometric basis for use in PES approximation [84, 200]. Instead of constructing an interpolant with polynomial basis functions, we use sines and cosines, which guarantee periodicity of the surrogate PES gradient with respect to internal rotations. We employ the $[W(Cp)(CO)_3]_2$ molecule as a model to test our new interpolation algorithm (see Figure 5.1). The energy barrier for gauche–anti interconversion of the molecule is 15.2 kcal/mol,[2] based on nuclear-resonance measurements [1]. Two-dimensional PESs were constructed with polynomial and trigonometric basis functions, where the RCs correspond to the rotation of $[W(Cp)(CO)_3]$ monomer (x_1) and the rotation of a Cp ring (x_2). A comparison of the two PESs is presented to show the advantages of a trigonometric basis for periodic coordinates.

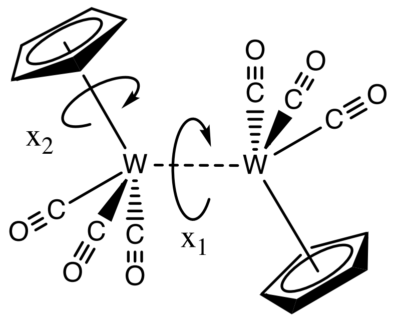


Figure 5.1 Model molecule used in this work, $[W(Cp)(CO)_3]_2$.

[2] 1 kcal = 4.184 kJ.

5.1 Computational details

The relaxed PES defined in Equation (1.2) requires an optimization over $\boldsymbol{\xi}$, as well as solving (approximately) the Schrödinger equation for each $\boldsymbol{q} = (\boldsymbol{x}, \boldsymbol{\xi})$ in the optimization iteration. Due to the computational expense involved, directly evaluating (1.2) in a dynamical simulation is impractical for systems with $N > 10$, necessitating a surrogate model $E_n^s(\boldsymbol{x})$. Moreover, when E_n and ∇E_n are periodic, the surrogate model E_n^s and ∇E_n^s must also be periodic. Sparse polynomial interpolation can approximate a PES for dynamical simulations [146]. As noted previously in the literature [140, 146, 206], sparse interpolation improves the ratio of approximation accuracy to the number of nodes, leading to a more efficient approximate PES with respect to the number of expensive ab initio calculations. When populating the nodes, each *ab initio* calculation is independent of the others, so the expensive part of the surrogate model is parallelizable. However, a polynomial interpolation basis can—and, in practice, does—fail to enforce periodicity of ∇E_n^s, leading to nonphysical dynamics. Therefore, we use the sparse trigonometric interpolation algorithm described in Section 2.4 to construct the surrogate potential energy surface.

5.1.1 Electronic structure calculations

All electronic structure calculations were carried out in the Gaussian 16 software package [73] with the B3LYP functional [15, 16, 117, 198]. Appendix A.1 contains specimen input files. The SDD pseudopotential and its associated basis set [103] were used for W, and the 3-21G basis set [23, 59–61, 79, 167] was used for H, C, and O in all calculations. The 3-21G basis set was chosen to reduce the computational cost of PES construction and validation. The optimized geometry produced by this level of theory agrees well with the crystal structure [1] of the $[W(Cp)(CO)_3]_2$ molecule (see Appendix A.1, Table A.1). Previous computational studies with 3-21G basis set and B3LYP functional on transition metal complexes and organic molecules also showed their ability to reproduce optimized structures, frequencies and PESs with the accuracy comparable to calculations with larger basis sets [69, 114, 219, 229]. Most importantly, the performance of the interpolation algorithm presented in this work is independent of the exact model chemistry utilized since the periodicity of the constructed PES and its gradient will not depend on the level of theory employed in the electronic structure calculations.

Frequency analysis was applied after each unconstrainted optimization to guarantee that a stationary point of the correct type was found. The geometry of the molecule is defined in Z-matrix format with four dummy atoms (see Figure 5.2 and Section A.1). Two dihedral angles, X1–W1–W2–X3 (x_1) and C1–X1–W1–W2 (x_2), were employed as the design variables for the construction of the PES. The domains for interpolation are $[0, 360)$ for x_1 and $[0, 72)$ for x_2.

The following symmetry was employed to further reduce the number of DFT calculations for the PES construction:

$$E(x_1, x_2) = E(360 - x_1, 72 - x_2)\,.$$

By exploiting this symmetry, we only need to run electronic structure calculations for nodes with $x_1 \in [0, 180]$ in order to populate the sparse-grid nodes.

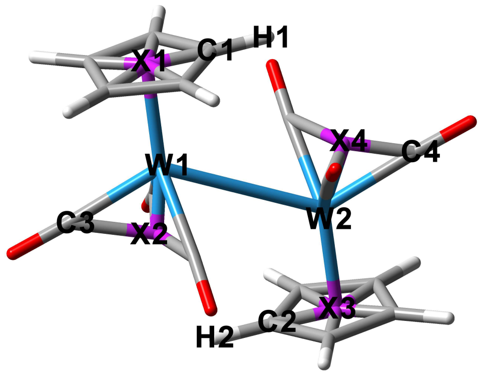


Figure 5.2 Global minimum structure for $[W(Cp)(CO)_3]_2$. Atoms X1–X4 are dummy atoms in the Z-matrix definition.

5.2 Results

We constructed two surrogate PESs for the ground state of the $[W(Cp)(CO)_3]_2$ molecule, shown in Figure 5.1 with the design variables x_1 and x_2 labeled. One PES employs the sparse polynomial interpolant used by Nance, Jakubikova, and Kelley [146]. The other utilizes the sparse trigonometric interpolant of $E_n(\boldsymbol{x})$, whose mathematical description is in Section 2.4. The second approach has not previously been deployed in surrogate PES modeling. We will test the following hypotheses: that a sparse trigonometric interpolant

(a) yields a more accurate approximation than a polynomial interpolation basis, and

(b) enforces periodicity of $\nabla E_n^s(\boldsymbol{x})$ to numerical accuracy.

Sparse grids for the trigonometric and polynomial interpolants are shown in Figure 5.3. Interpolation domains are $x_1 \in [0, 360]$ and $x_2 \in [0, 72]$ since x_2 corresponds to the rotation of a pentagonal group. The trigonometric grid has 135 points; the polynomial grid has 145 points. After evaluating the true PES $E_n(\boldsymbol{x})$ at each node shown in Figure 5.3, we invoked a simple call to TASMANIAN [200] to construct the surrogate potential energy surfaces shown in Figure 5.4. As long as the underlying PES is piecewise continuous, then the interpolation converges by (2.108). Apart from the nodes (shown as black dots), the surfaces in Figure 5.4 look similar to the eye along the x_1 direction. Furthermore, the shape of the PES and the different minimum-energy paths in Figure 5.4 suggest that the rotation of the Cp ring (x_2) is coupled with the rotation of the [W(Cp)(CO)$_3$] monomer (x_1).

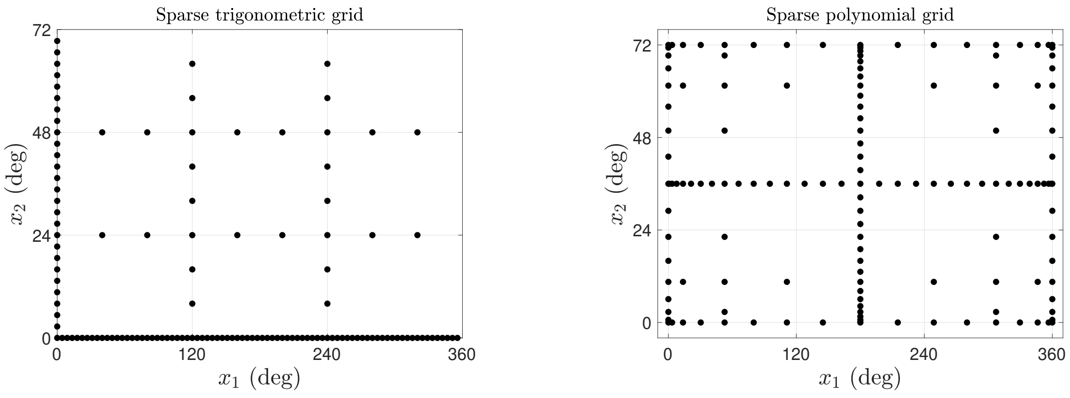


Figure 5.3 Left: Anisotropic sparse grid for trigonometric interpolant ($d = 2$, $L = 4$, $\boldsymbol{\alpha} = (5, 6)$). Note that we have more points along x_1 than x_2. Right: Sparse grid for polynomial interpolant using Clenshaw–Curtis nodes ($d = 2$, $L = 5$).

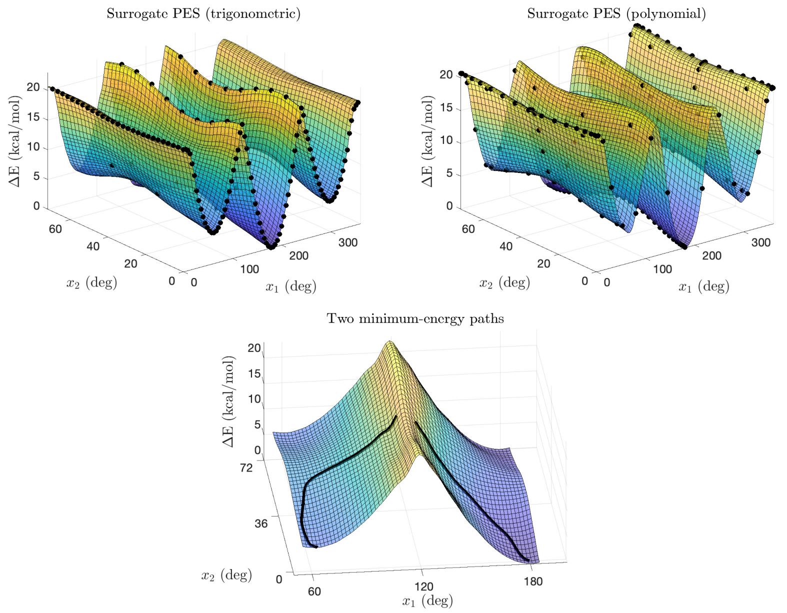


Figure 5.4 Surrogate ground state ($n = 0$) PES corresponding to the sparse grids in Figure 5.3. Interpolation nodes shown as black dots.

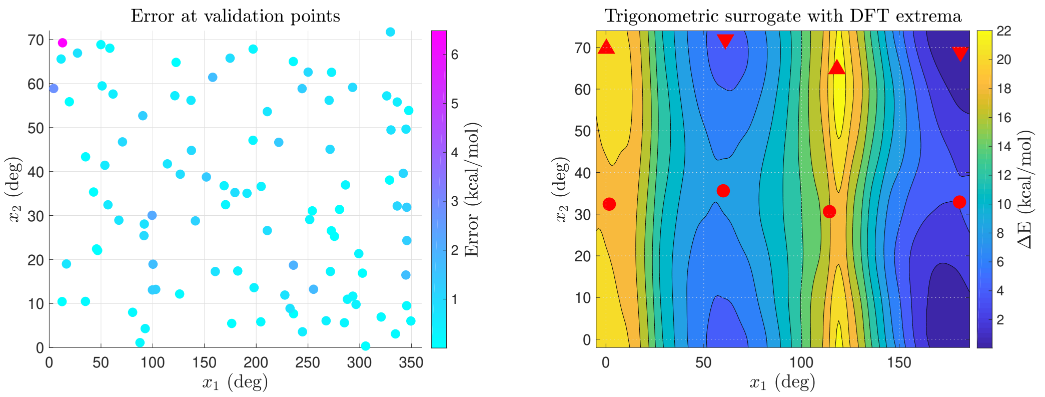


Figure 5.5 Left: 200 validation points. The points are colored based on the trigonometric surrogate PES error relative to the DFT-calculated energies. Right: Location of DFT extrema (minima are downward-pointing triangles, maxima are upward-pointing triangles, and saddle points are dots).

To quantify the error of our two surrogates versus the true PES, we randomly sampled 200 values of (x_1, x_2) from a uniform distribution on $[0, 360] \times [0, 72]$ using MATLAB's `rand` command, displayed in Figure 5.5. For each validation point, we performed a constrained optimization at the B3LYP/(SDD,3-21G) level of theory and compared the DFT-calculated energy to a single-point evaluation of the surrogate PES. We display the root-mean-square error (RMSE) and maximum absolute error (MAE) in Table 5.1. The 95% confidence interval for RMSE is calculated by treating the mean-squared error as a χ^2 random variable with 200 degrees of freedom [70].

Table 5.1 Error against true values of PES at 200 points drawn from uniform distribution on $[0, 360] \times [0, 72]$. Units are kcal/mol.

	Trigonometric	Polynomial
RMSE	0.70	1.26
RMSE 95% conf. int.	$(0.63, 0.77)$	$(1.15, 1.40)$
MAE	2.15	4.14

In addition, we compared the energy and geometry of minima, maxima and saddle points on the two surrogate PESs with the fully optimized DFT structures. We display the DFT extrema on top of a contour plot of the trigonometric surrogate in Figure 5.5. As described in the methodology section, DFT-optimized structures are characterized by the number of imaginary frequencies (i.e., zero for minima, one for saddle points, and two for maxima on a two-dimensional PES).

Details of the electronic structure calculations that produced these structures are described in Section A.1. The optimized dihedral angles and relative energies for those stationary points are summarized in Table 5.2. The error from the trigonometric interpolant is smaller than the error of polynomial interpolant.

Table 5.2 Dihedral angles and relative energies of stationary points from PESs and DFT optimizations. The energy in the table is relative to the minimum-energy conformation with x_1 close to 180 degrees. Units of x_1 and x_2 are degrees; units of E_{rel} are kcal/mol.

Type	DFT			Trigonometric			Polynomial		
	x_1	x_2	E_{rel}	x_1	x_2	E_{rel}	x_1	x_2	E_{rel}
max	0.2	69.7	20.7	2.9	64.9	20.7	4.1	66.3	20.8
saddle	1.7	32.4	19.9	-2.0	35.5	19.9	-1.9	39.0	19.9
min	61.1	71.9	5.1	63.0	70.1	5.2	66.6	72.7	4.6
saddle	60.0	35.6	7.5	56.5	38.1	8.1	62.3	35.7	7.7
max	118.1	64.8	23.4	119.3	60.6	23.1	121.7	82.2	20.8
saddle	114.3	30.6	21.8	118.8	25.6	19.6	122.2	49.2	18.7
min	181.3	68.9	0.0	180.0	71.8	0.0	177.6	68.4	0.0
saddle	180.8	32.9	3.3	180.1	36.1	4.1	180.4	33.1	3.3
RMSE				2.7	3.6	0.9	4.4	9.4	1.4

Next, we examined the mismatch of the surrogate PES gradient. We compute the maximum x_1-gradient mismatch as

$$\max_{x_2 \in [0,72]} |f_{x_1}(0, x_2) - f_{x_1}(360, x_2)|$$

where $f_{x_1}(0, x_2)$ is understood to be a derivative from the right, and $f_{x_1}(360, x_2)$ is a derivative from the left. The maximum x_2-gradient mismatch is computed analogously. These results are shown in Table 5.3. The numerical error of forward and backward differences is $\mathcal{O}(h)$, where h is the step size. Therefore, since the maximum mismatches in the trigonometric case are indeed $\mathcal{O}(h)$, they are numerically zero. In contrast, the mismatches in the polynomial case are far larger than $\mathcal{O}(h)$.

Table 5.3 Gradient mismatches for trigonometric and Clenshaw–Curtis bases. Right/left derivatives approximated by forward/backward differences with step size $h = 10^{-5}$.

	Trigonometric	Polynomial
Max x_1-gradient mismatch	9.02×10^{-6}	2.05×10^{-1}
Max x_2-gradient mismatch	2.99×10^{-5}	9.62×10^{-1}

5.3 Discussion

The visual results in Figure 5.4 look reasonable for the two choices of surrogate. There are peaks when $x_1 = 120$ and $x_1 = 240$, and a global minimum occurs at $x_1 = 180$ as expected from empirical studies [1]. The real significance is in Tables 5.1 and 5.2, which demonstrate that the error for the trigonometric interpolant is smaller (by ∼0.5 kcal/mol on average) than the error for the polynomial interpolant. The maximum absolute error is almost exactly 2 kcal/mol smaller for the trigonometric interpolant. Recall that the trigonometric sparse grid has 135 nodes and the polynomial sparse grid has 145 nodes. Sparse trigonometric interpolation results in a more accurate surrogate PES than sparse polynomial interpolation at no increase in the number of evaluations of $E_n(\boldsymbol{x})$ (as measured by the number of nodes).

Additionally, the numerical results in Table 5.3 indicate that $\nabla E_n^s(\boldsymbol{x})$ is periodic when we use the trigonometric interpolation basis. The discretization error for forward/backward differences is $\mathcal{O}(h)$, which is precisely what we observe for the trigonometric basis. For the Clenshaw–Curtis polynomial basis, we observe a real-life example of ∇E_n^s failing to be periodic. Importantly, this failure occurs even though ∇E_n is periodic (since x_1 and x_2 are rotations). Thus, for applications where it is an absolute necessity that the surrogate PES gradient be periodic, a polynomial interpolant should *not* be used. Furthermore, the discussion in the previous paragraph indicates that, due to improved accuracy at no extra cost, one should consider using a trigonometric interpolation basis even when periodicity of the gradient is not a rigid requirement.

For each interpolation basis, we observed a significantly larger error for the two structures with x_1 close to 120 degrees. The larger errors occur because optimized geometries in this region mix two possible conformations: the locked conformation and the unlocked conformation (see Figure 5.6). When x_1 is smaller than 120 degrees, one CO group of one [W(Cp)(CO)$_3$] monomer is pointing at the center of two CO groups of the other monomer in the lowest energy conformations, because such conformations minimize the steric effect for conformations with small x_1 values. For the same reason, at large x_1 values, the unlocked conformation is more favorable than the locked conformation. As a result, the optimized geometries differ on opposite sides of 120 degrees. The cusp of the true PES makes this local region poorly described by the

PES approximation with x_1 and x_2. Furthermore, due to the global definition of the interpolation basis functions, the quality of the entire surrogate is affected.

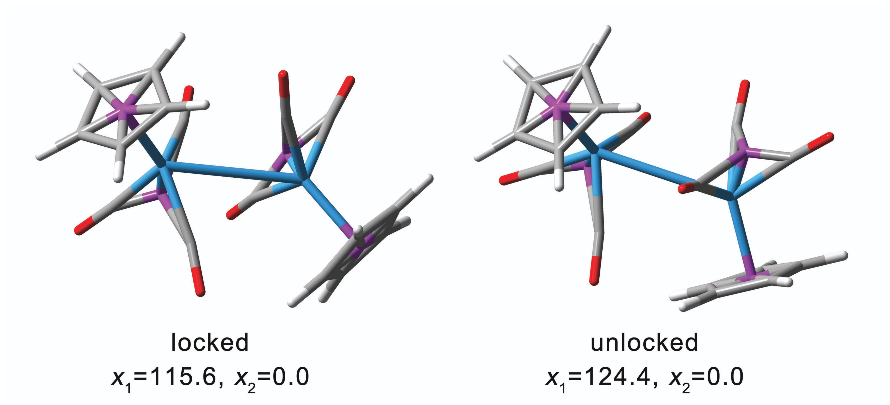


Figure 5.6 Locked and unlocked structures near $x_1 = 120$ and $x_2 = 0$.

As a caveat, the trigonometric interpolation algorithm we presented should only be used in problems where the PES is periodic in *all* components of $\boldsymbol{x}$ (i.e., all interesting geometry features are bond angles or dihedral angles). Approximating a non-periodic function with sines and cosines leads to poor accuracy at the edges of the domain, known as the Gibbs phenomenon [91]. The next chapter addresses what to do when the relaxed PES has a mixture of periodic and nonperiodic inputs.

CHAPTER

6 MOLECULAR DYNAMICS WITH MIXED-BASIS SURROGATES

In the previous chapter, we constructed surrogate PESs with sparse trigonometric interpolation [81, 84, 140] on a tungsten molecule where the design variables are all periodic [141]. The interpolation basis functions in this case are sines and cosines. This method enforces periodicity to numerical tolerances and also produces a slightly more accurate surrogate PES than sparse polynomial interpolation on the same system [141].

A limitation of existing interpolative PES approximation methods is that the same class of basis functions must be applied to each design variable [140, 144]. However, many chemical systems of interest involve both periodic coordinates (e.g., full-rotation torsion angles) and nonperiodic coordinates (e.g., bond lengths, bond angles). In these systems, polynomial interpolation on periodic coordinates will lead to nonphysical phenomena due to the lack of periodicity in the surrogate PES gradient. However, trigonometric interpolation on nonperiodic coordinates will lead to persistent inaccuracies at the domain boundary called *Gibbs effects* [91].

We chose azomethane for testing the mixed sparse polynomial and trigonometric interpolative PES approximation method. Azomethane is a relatively small molecule ($N = 10$ atoms), but it has interesting chemical properties to simulate. It has two stable conformations in the S_0 state, which are *trans* and *cis*, and it is known to decompose into N_2 and two methyl radicals by stepwise dissociation when it is excited to the S_1 state in gas phase (Figure 6.1) [5, 35]. Liu et al

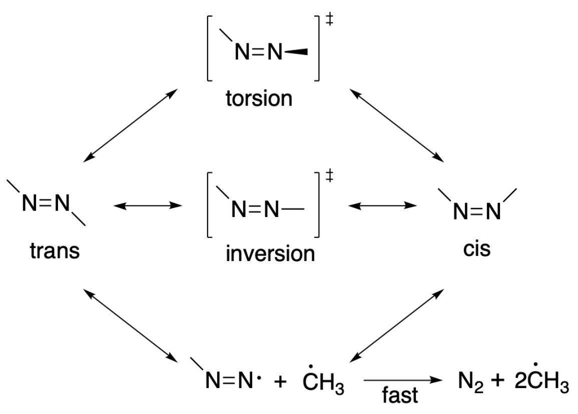


Figure 6.1 Isomerization and dissociation scheme of azomethane.

investigated the dissociation mechanism with state-average complete active space self-consistent-field (saCASSCF) and multireference configuration interaction with single and double excitation (MRCISD) methods to prove that dissociation of the C–N bond is sequential [128]. This was further studied by Sellner and coworkers using nonadiabatic *ab initio* surface-hopping dynamics with MCSCF-GVB-CAS and MRCISD methods [185]. The azomethane dynamics in solution are also studied with QM/MM to show that C–N dissociation is suppressed with the presence of polar or nonpolar solvents [183]. Cattaneo et al constructed a PES of azomethane at the CASSCF level and used trajectory surface hopping (TSH) and molecular dynamics surface hopping (MDSH) to show similar results [37, 38]. While we are focusing on the construction of and dynamics on the mixed-basis surrogate S_0 PES of azomethane in this chapter, the constructed mixed basis PES for azomethane will be used for a reduced-dimensional TSH photoexcitation and relaxation simulation in the future.

In this chapter, we present a mixed-basis interpolative method that uses the Smolyak sparse grid construction [192]. This method applies trigonometric interpolation to the periodic design variables and polynomial interpolation to all others. We construct a mixed-basis surrogate PES for azomethane and use it to drive a reduced-dimensional MD simulation of azomethane isomerization paths. This method requires fewer electronic structure evaluations than all-polynomial interpolation to obtain realistic energy barriers and locations of minima and transition states. Furthermore, we demonstrate explicitly that the mixed-basis surrogate appropriately conserves total energy over the entire time integration, while the all-polynomial surrogate does not.

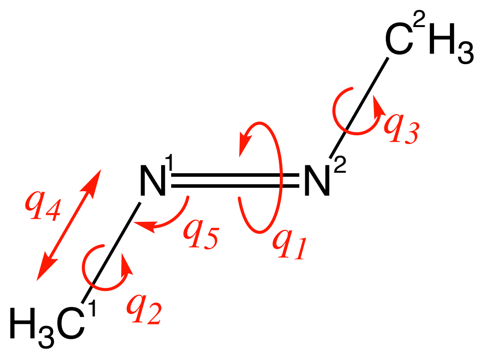


Figure 6.2 Structure of azomethane and design variables $\boldsymbol{q}$.

6.1 Computational details

In this section, we discuss how to construct a mixed-basis surrogate potential energy surface (PES), as well as describe the reduced-dimensional molecular dynamics (MD) framework. The full PES $\mathcal{E}(\boldsymbol{x})$ of an N-atom molecule is a function of the internal coordinates $\boldsymbol{x} \in \mathbb{R}^{3N-6}$. We first partition $\boldsymbol{x} = (\boldsymbol{q}, \boldsymbol{\xi})$, where $\boldsymbol{q} \in \mathbb{R}^d$ are the design variables and $\boldsymbol{\xi}$ are the remainder variables. Then we minimize over $\boldsymbol{\xi}$ to compute the relaxed PES:

$$E(\boldsymbol{q}) = \min_{\boldsymbol{\xi}} \mathcal{E}(\boldsymbol{q}, \boldsymbol{\xi})\,. \tag{6.1}$$

Recently there have been studies on automatically detecting the relevant design variables [86, 184], but one often selects the design variables based on *a priori* chemical knowledge or empirical studies of the system. Additionally, the full Cartesian geometry of the molecule must be continuous with respect to $\boldsymbol{q}$, which will be discussed in Section 6.1.1. If a certain remainder variable has discontinuities that cannot be smoothed out, then that variable may be added to the design variables to enforce continuity. Based on continuity requirements and previous studies of azomethane [36–38], we chose the design variables shown in Figure 6.2. Three of the controlled geometric parameters (q_1, q_4, q_5) were chosen to illustrate three pathways for conformational transition between *trans* and *cis*: rotation, inversion and methyl dissociation. The two methyl rotations (q_2, q_3) were controlled to prevent hydrogen atoms from unpredictably permuting their ordering during structure optimization. We describe $\boldsymbol{q}$ in much greater detail in Section 6.2.2.

Equation (6.1) requires an optimization over $\boldsymbol{\xi}$, in addition to an approximate solution of the Schrödinger equation for $\boldsymbol{q} = (\boldsymbol{x}, \boldsymbol{\xi})$ at each optimization step. Due to the computational cost, directly evaluating (6.1) repeatedly (e.g., in a dynamics simulation) is highly impractical for molecules with $N > 10$, rendering a surrogate model $V(\boldsymbol{q}) = E^s(\boldsymbol{q})$ necessary. Compared to least-squares, an advantage of interpolatory surrogates is that, in general, fewer evaluations

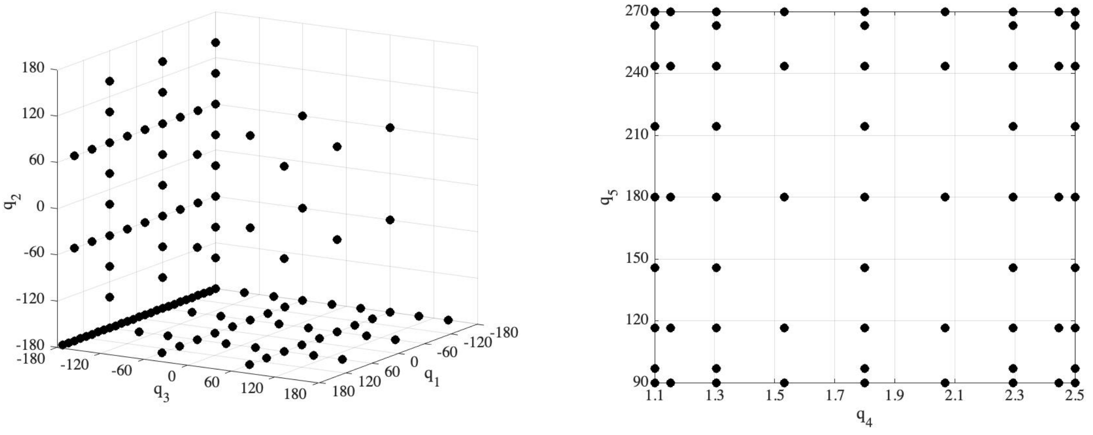


Figure 6.3 7215 mixed basis nodes composed of 111 trigonometric nodes (left) and 65 polynomial nodes (right).

of the true function are necessary to achieve a given accuracy threshold. Sparse interpolation is an attractive choice because of the improved ratio of approximation accuracy to the number of interpolation nodes, leading to a more accurate surrogate with respect to the number of expensive *ab initio* calculations. Each *ab initio* calculation is independent of the others, and therefore populating the nodes is naturally parallelizable.

Previously, sparse polynomial and trigonometric interpolation have been used to construct $E^s(\boldsymbol{q})$, where one chooses the interpolation basis at the front end and applies it to each component of $\boldsymbol{q}$. In this chapter, however, we aim to apply trigonometric interpolation to the periodic coordinates and polynomial interpolation to everything else. We use the mixed-basis algorithm in Section 2.5, with the nodes shown in Figure 6.3.

6.1.1 Reduced-dimensional molecular dynamics

We follow the relaxed reduced-dimensional molecular dynamics framework originally developed by Liu and coworkers [125], which we briefly summarize here. Specifically, we use the NVE ensemble and Langevin thermostat to demonstrate the flexibility of our method. One difference between Liu's work and this thesis is that we have implemented our MD code in Python since Tasmanian's Python interface is much faster than its MATLAB interface. The codes, including examples, are freely available on GitHub.[1]

In the Hamiltonian formalism [52, 115], the equations of motion for the design variables $\boldsymbol{q}$

[1] `https://github.com/zbmorrow/mixed_basis_rrmd`

and the generalized momenta $\boldsymbol{p}$ are

$$\begin{cases} \dot{\boldsymbol{q}} &= \dfrac{\partial \mathcal{H}}{\partial \boldsymbol{p}} \\ \dot{\boldsymbol{p}} &= -\dfrac{\partial \mathcal{H}}{\partial \boldsymbol{q}} \end{cases} . \tag{6.2}$$

The classical Hamiltonian is a sum of potential and kinetic energy terms, expressed as

$$\mathcal{H}(\boldsymbol{q}, \boldsymbol{p}) = V(\boldsymbol{q}) + K(\boldsymbol{q}, \boldsymbol{p}) \tag{6.3}$$

where $V(\boldsymbol{q})$ is the sparse interpolant of the relaxed PES and $K(\boldsymbol{q}, \boldsymbol{p})$ involves the momenta and the mass-metric tensor $\boldsymbol{G}$:

$$K(\boldsymbol{q}, \boldsymbol{p}) = \frac{1}{2} \boldsymbol{p}^T \boldsymbol{G}^{-1}(\boldsymbol{q}) \, \boldsymbol{p}, \tag{6.4}$$

$$G_{ij}(\boldsymbol{q}) = \sum_{k=1}^{3N} m_k \frac{\partial X_k(\boldsymbol{q})}{\partial q_i} \frac{\partial X_k(\boldsymbol{q})}{\partial q_j} \tag{6.5}$$

The function $\boldsymbol{X} : \mathbb{R}^d \to \mathbb{R}^{3N}$ maps the design variables to the full Cartesian molecular geometry, and m_k is the atomic mass corresponding to Cartesian component X_k. In order to be approximated with sparse interpolation, $\boldsymbol{X}(\boldsymbol{q})$ must be continuous.

To integrate the system forward in time, we must first choose $\boldsymbol{q}(0) = \boldsymbol{q}_0$ and compute $\boldsymbol{p}(0) = \boldsymbol{p}_0$. The momenta are computed by selecting a starting temperature T and drawing $3N$ Cartesian velocities from a Boltzmann distribution

$$\boldsymbol{v}_0 = \sqrt{k_B \, \boldsymbol{M}^{-1} \, T} \, \boldsymbol{R}_t \, , \qquad \boldsymbol{R}_t \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}). \tag{6.6}$$

Here, k_B is the Boltzmann constant in appropriate units, $\boldsymbol{M} = \mathrm{diag}(m_1, \ldots, m_{3N})$, and $\boldsymbol{R}_t$ is a $3N$-dimensional standard normal random variable realized at time t. We then project the initial velocities onto the reduced-dimensional space by setting

$$\boldsymbol{p}(0) = \boldsymbol{X}'(\boldsymbol{q})^T \, \boldsymbol{v}_0 \, , \qquad (\boldsymbol{X}'(\boldsymbol{q}))_{ij} = \frac{\partial X_i(\boldsymbol{q})}{\partial q_j} \, . \tag{6.7}$$

From Equations (6.3)–(6.5), we need to construct a surrogate PES $V(\boldsymbol{q})$ and Cartesian mapping function $\boldsymbol{X}(\boldsymbol{q})$, which we approximate using sparse interpolation.

6.1.1.1 NVE simulations

In the NVE ensemble, we wish to simulate the trajectory of a molecule while conserving total energy. We integrate the system forward in time with the Störmer–Verlet method, which conserves total energy [199, 218]. At step $t = t_n$, the system is propagated forward by the three-step process

$$\begin{cases} \boldsymbol{q}_{n+1/2} & = \boldsymbol{q}_n + \dfrac{\Delta t}{2} \dfrac{\partial \mathcal{H}(\boldsymbol{q}_{n+1/2}, \boldsymbol{p}_n)}{\partial \boldsymbol{p}} \\ \boldsymbol{p}_{n+1} & = \boldsymbol{p}_n - \dfrac{\Delta t}{2} \left[\dfrac{\partial \mathcal{H}(\boldsymbol{q}_{n+1/2}, \boldsymbol{p}_n)}{\partial \boldsymbol{q}} + \dfrac{\partial \mathcal{H}(\boldsymbol{q}_{n+1/2}, \boldsymbol{p}_{n+1})}{\partial \boldsymbol{q}} \right] \\ \boldsymbol{q}_{n+1} & = \boldsymbol{q}_{n+1/2} + \dfrac{\Delta t}{2} \dfrac{\partial \mathcal{H}(\boldsymbol{q}_{n+1/2}, \boldsymbol{p}_{n+1})}{\partial \boldsymbol{p}} \end{cases} . \tag{6.8}$$

The first step is an implicit Euler half-step in $\boldsymbol{q}$, the second is a full Crank–Nicolson step in $\boldsymbol{p}$, and the third is an explicit Euler half-step in $\boldsymbol{q}$ [76]. The derivatives are expressed as

$$\frac{\partial \mathcal{H}}{\partial \boldsymbol{q}} = \frac{\partial V}{\partial \boldsymbol{q}} + \frac{\partial K}{\partial \boldsymbol{q}} \tag{6.9}$$

$$\frac{\partial \mathcal{H}}{\partial \boldsymbol{p}} = \boldsymbol{G}^{-1}(\boldsymbol{q})\, \boldsymbol{p} \tag{6.10}$$

and then evaluated numerically. The first two steps of Störmer–Verlet involve solving a nonlinear system of equations. For the all-polynomial surrogates, we use a derivative-free optimizer in the SciPy package [173, 221] since polynomial gradients are not guaranteed to be continuous across the periodic boundary [141]. Such derivative-free methods are more computationally costly, but they are the only option for regions near the periodic boundary, where $V_{poly}(\boldsymbol{q})$ is not differentiable. An advantage of the mixed-basis method is that we may solve the nonlinear system with Newton's method [104, 105], which is much cheaper computationally, since $V_{mixed}(\boldsymbol{q})$ is differentiable across the periodic boundary.

6.1.1.2 Langevin thermostat

The Langevin thermostat is a very popular algorithm for NVT simulations, in which temperature is conserved. The equations of motion (6.2) become a system of stochastic differential equations with additional terms to incorporate the coupling between the system and a thermal bath [4]. In Cartesian coordinates $(\boldsymbol{Q}, \boldsymbol{P})$, the Langevin equations are

$$\begin{cases} \mathrm{d}\boldsymbol{Q} & = \dfrac{\partial \mathcal{H}}{\partial \boldsymbol{P}} \,\mathrm{d}t \\ \mathrm{d}\boldsymbol{P} & = -\dfrac{\partial \mathcal{H}}{\partial \boldsymbol{Q}} \,\mathrm{d}t - \gamma \boldsymbol{P} \,\mathrm{d}t + \boldsymbol{\sigma} \,\mathrm{d}\boldsymbol{w} \end{cases} \tag{6.11}$$

where $\boldsymbol{Q}$ and $\boldsymbol{P}$ are the position and momentum of the particle. The additional terms capture the viscosity of the bath ($-\gamma \boldsymbol{P}$) and random forces ($\boldsymbol{\sigma} \,\mathrm{d}\boldsymbol{w}$), without which (6.11) and (6.2) are equivalent. From the fluctuation–dissipation theorem, the coefficient σ_i can be written in terms of γ as

$$\sigma_i = \sqrt{2\gamma m_i k_B T} \tag{6.12}$$

where T is the target temperature [4, 119]. Like Liu and coworkers [125], we employ the BAOAB method of Leimkuhler and Matthews [118, 119] to integrate (6.11). In the reaction coordinates $(\boldsymbol{q}, \boldsymbol{p})$, the BAOAB method is

$$\text{(BA)} \begin{cases} \boldsymbol{p}_{n+1/2} & = \boldsymbol{p}_n - \dfrac{\Delta t}{2} \dfrac{\partial \mathcal{H}(\boldsymbol{q}_n, \boldsymbol{p}_{n+1/2})}{\partial \boldsymbol{q}} \\ \boldsymbol{q}_{n+1/2} & = \boldsymbol{q}_n + \dfrac{\Delta t}{2} \dfrac{\partial \mathcal{H}(\boldsymbol{q}_n, \boldsymbol{p}_{n+1/2})}{\partial \boldsymbol{p}} \end{cases}, \tag{6.13}$$

$$\text{(O)} \left\{ \boldsymbol{p}'_{n+1/2} \quad = \mathrm{e}^{-\gamma \Delta t} \boldsymbol{p}_{n+1/2} + \sqrt{1 - \mathrm{e}^{-2\gamma \Delta t}}\, \boldsymbol{X}'(\boldsymbol{q}_{n+1/2})^T \sqrt{k_B \boldsymbol{M} T}\, \boldsymbol{R}_t \right. , \tag{6.14}$$

$$\text{(AB)} \begin{cases} \boldsymbol{q}_{n+1} & = \boldsymbol{q}_{n+1/2} + \dfrac{\Delta t}{2} \dfrac{\partial \mathcal{H}(\boldsymbol{q}_{n+1}, \boldsymbol{p}'_{n+1/2})}{\partial \boldsymbol{p}} \\ \boldsymbol{p}_{n+1} & = \boldsymbol{p}'_{n+1/2} - \dfrac{\Delta t}{2} \dfrac{\partial \mathcal{H}(\boldsymbol{q}_{n+1}, \boldsymbol{p}'_{n+1/2})}{\partial \boldsymbol{q}} \end{cases}. \tag{6.15}$$

6.2 Results and discussion

We used our mixed-basis formulation to construct surrogates for the S_0 PES and the Cartesian mapping function $\boldsymbol{X}(\boldsymbol{q})$. In an NVE framework, we compared the performance of the mixed-basis surrogate to both Born–Oppenheimer molecular dynamics (BOMD) [90, 215] and the previous state of the art, which uses an all-polynomial interpolation basis [125]. We then use a Langevin thermostat to demonstrate that our mixed-basis surrogate produces accurate temperature distributions and that azomethane isomerization is not likely to occur on the S_0 surface alone, even at high temperatures.

6.2.1 Electronic structure of azomethane

All electronic structure calculations utilized the Gaussian 16 software package [73] with the B3LYP hybrid functional [15, 16, 117, 198] and the 6-311G* basis set [112]. The computing environment was Henry2, a high-performance computing cluster at North Carolina State University. Each Gaussian instance used 16 cores on an Intel Xeon processor and was allocated 32GB RAM. Further details on the electronic structure calculations are in Section 6.2.2 and Appendix A.2.

One of the chemical properties of azomethane that is important to note is that its lowest-energy electronic state changes with the conformational change. As illustrated in Figure 6.4,

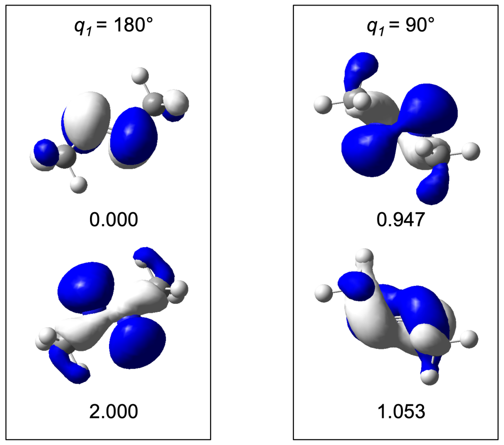


Figure 6.4 Natural orbitals of azomethane at $q_1 = 180°$ and $90°$ with occupation numbers.

natural orbitals of azomethane, which are nitrogen lone pairs and π^* of nitrogen 2p orbitals, at its ground state become degenerate when q_1 is $90°$. Therefore when the PES is calculated with closed-shell restricted wavefunctions, we observe a sharp peak near $q_1 = 90°$. In order to obtain a smooth PES, the stability tests [13] of DFT wavefunctions were performed after each calculation.

6.2.2 Constructing the surrogate PES

Each geometry is supplied to Gaussian in Cartesian coordinates, but the optimization uses redundant internal coordinates. Each Gaussian job proceeds as follows: (1) stability check [13] on wavefunction, (2) optimization [122] over remainder variables, (3) stability check, (4) optimization and frequency analysis, and (5) stability check. The safeguards are to ensure that the relaxation over $\boldsymbol{\xi}$ found a true minimum and that the lowest-energy wavefunction is selected. In this way, we can build surrogates for $E_{S_0}(\boldsymbol{q})$ and $\boldsymbol{X}_{S_0}(\boldsymbol{q})$. We provide sample Gaussian input files in Appendix A.2.

The design variables, shown in Figure 6.2, are defined as follows:

1. $q_1 \in [-180, 180]$ is the C^1–N^1=N^2–C^2 dihedral angle;
2. $q_2 \in [-180, 180]$ is equal to the N^2=N^1–C^1–H dihedral angle plus $(q_1 + 180)$, which we will explain shortly;
3. $q_3 \in [-180, 180]$ is the N^1=N^2–C^2–H$'$ dihedral angle;
4. $q_4 \in [1.1, 2.5]$ is the N^1–C^1 bond length; and
5. $q_5 \in [90, 270]$ is the N^2=N^1–C^1 bond angle.

Angles are measured in degrees, and bond lengths in Å. The variables q_1, q_2, and q_3 are periodic, while q_4 and q_5 are nonperiodic. We allow q_5 to be linear or greater than 180° in order to capture methyl inversion, in accordance with previous work [36, 38]. The lower bound on q_4 and both bounds on q_5 were chosen to be the widest possible bounds without the molecule dissociating during the optimization process. We need a large enough domain on either side of the equilibrium q_5 value to capture the well. Furthermore, the PES is mostly flat in the q_4 direction for $q_4 > 2.5$ [38], so we truncated the domain to ensure that $\boldsymbol{X}(\boldsymbol{q})$ is continuous; if q_4 drifts beyond 2.5 Å during the course of an MD simulation, we consider the molecule dissociated and terminate the simulation. The design variable q_2 was originally not part of our set of design variables but was added to maintain continuity of $\boldsymbol{X}(\boldsymbol{q})$.

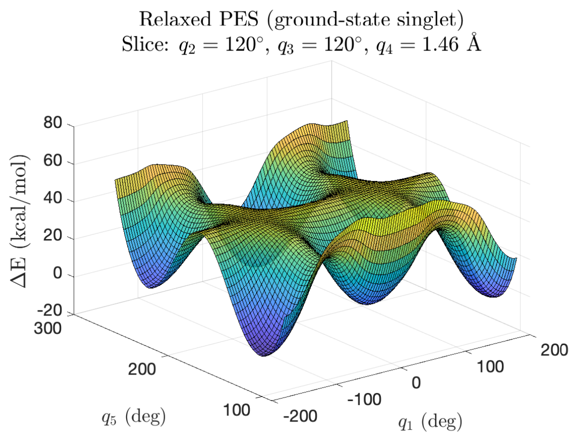
Since the energy of a molecule is invariant with respect to translations or rotations of the entire system, we must reconstruct the Cartesian geometries at the nodes in a consistent manner in order to build $\boldsymbol{X}(\boldsymbol{q})$. Liu, Jakubikova, and Kelley used Kabsch alignment to minimize the root-mean-squared deviation between each node and the global minimum. However, their application was NH_3 inversion, in which all atoms except nitrogen are moving confluently; in our system, several atoms (e.g. N^1, N^2, C^2) are mostly stationary. To obtain consistent Cartesian geometries, we apply translations and rotations of the entire molecule to place N^1 at the origin, N^2 on the positive x-axis, and C^2 in Quadrant I of the xy-plane. We then load these geometries into Tasmanian to construct $\boldsymbol{X}(\boldsymbol{q})$.

Since the molecule can be linear or $q_5 > 180$ and since the Cartesian mapping function $\boldsymbol{X}(\boldsymbol{q})$ must be smooth, we need to take special care to encode the geometry properly in our input files. For a given sparse grid node $\boldsymbol{q}^i$, we apply the following transformations before converting from internal coordinates to Cartesian:

1. If $q_5^i > 180$, then we set $q_1^i \leftarrow 180 + q_1^i$, $q_2^i \leftarrow 180 + q_2^i$, and $q_5^i \leftarrow 360 - q_5^i$.

2. After Step 1, if $q_5^i < 180$, we recover the $N^2{=}N^1{-}C^1{-}H$ dihedral by setting $q_2^i \leftarrow q_2^i - (q_1^i + 180)$. This step is necessary to avoid multivalued geometries in the limit $q_5 \to 180$.

After finishing one batch sparse grid nodes, we examine the surrogate PESs by plotting various 2-D slices and optimizing the geometry on the surface to a local minimum or transition state. We also animated randomly generated one-dimensional trajectories by restricting four of the design variables. We refine the sparse grid until (1) the calculated energies, relative to the *trans-* minimum, are within ∼5% of their Gaussian-optimized value and (2) all animations are smoothly varying. We show two representative slices near the global minimum in Figure 6.5. Table A.3 in Appendix A.2 shows the energies and geometries at minima and saddle points for the highest level of refinement, as well as a selection of animations of $\boldsymbol{X}(\boldsymbol{q})$.



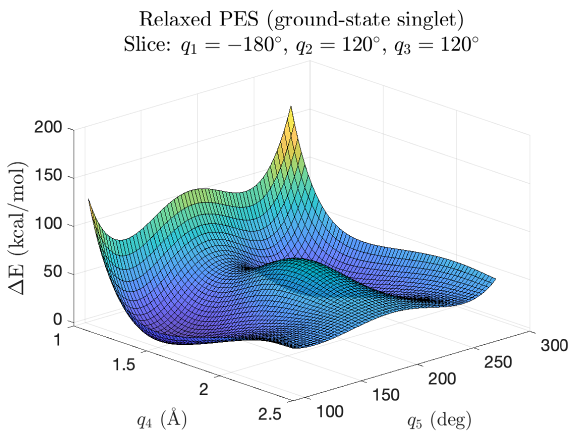


Figure 6.5 Mixed-basis surrogate PESs for the singlet ground state.

Table 6.1 Minimizers for the *trans*- conformation.

	# nodes	argmin (q_1, q_2, q_3, q_4, q_5)
Gaussian	—	$[180.00, 122.20, 122.20, 1.46, 113.00]$
Mixed basis	7215	$[-179.71, 121.91, 121.84, 1.46, 116.06]$
All polynomial	17233	$[-179.63, 137.91, -155.80, 1.47, 113.16]$

6.2.3 NVE: Mixed vs. all-polynomial basis

We begin by subjecting the surrogates to a relatively easy test: an NVE simulation at the *trans*-minimum on the S_0 surface [125]. We will compare BOMD, a mixed-basis surrogate, and an all-polynomial surrogate. Table 6.1 shows the initial geometry used for each flavor of surrogate. Ten initial velocities $\boldsymbol{v}(0) \in \mathbb{R}^{3N}$ are drawn from a Boltzmann distribution at 298.15 Kelvin. These are immediately used to run BOMD. For reduced-dimensional MD, we project the initial velocities onto $\boldsymbol{p}(0)$ using the surrogate $\boldsymbol{X}(\boldsymbol{q})$ corresponding to the basis in use. We integrate up to 2.5 ps with step size $\Delta t = 0.1$ fs. The computing environment for the reduced-dimensional MD simulations was XSEDE Bridges-2 [158, 212], while BOMD ran on Henry2, a high-performance computing cluster at North Carolina State University.

When measuring energy conservation in an MD simulation, one first stores kinetic and total energy at each time step. Then, the general principle is that one wants

$$\frac{\sigma(\{K_i + V_i\})}{\sigma(\{K_i\})} \leq \mathcal{O}(10^{-4}) \tag{6.16}$$

where σ denotes the standard deviation of the observations [4]. We computed these ratios for each run and method and show the results in Table 6.2. The sample means for BOMD and mixed-basis MD are within the usual bound, while that of the all-polynomial surrogate is orders

Table 6.2 Statistics of energy conservation ratios for each method, along with performance.

	BOMD	Mixed basis	All-polynomial
Mean	7.5×10^{-4}	1.8×10^{-5}	1.8×10^{-1}
Stdev	4.0×10^{-4}	1.2×10^{-5}	1.0×10^{-1}
Min	3.4×10^{-4}	6.2×10^{-6}	6.5×10^{-2}
Max	1.6×10^{-3}	4.5×10^{-5}	4.1×10^{-1}
Avg wall time (hr)	41.11	0.67	7.04
Cores	16	1	5

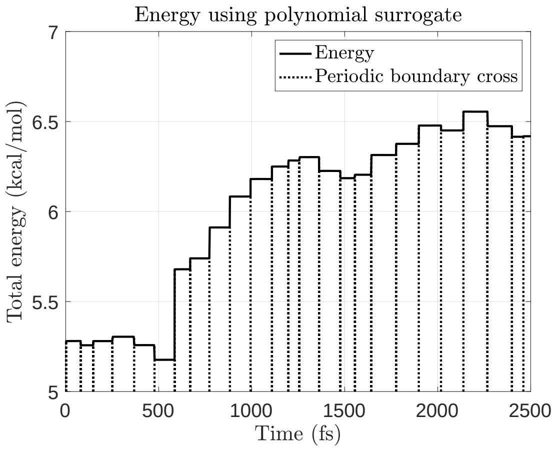


Figure 6.6 Solid line: total energy. Dotted lines: q_1, q_2, or q_3 is within 0.03° of ±180°.

of magnitude too large. Furthermore, the runtime is much lower when using a surrogate model, particularly for the mixed basis, which has the fastest turnaround time and lowest core usage. The core-hour usage of mixed-basis NVE MD is an order of magnitude lower than that of all-polynomial MD and two orders of magnitude lower than BOMD. In Figure 6.6, we show the iteration history of total energy during one of polynomial-basis simulations. When each jump occurs, at least one periodic design variables is near ±180°.

In Figure 6.7 we show histograms for selected design variables, remainder variables, and potential energy. Table 6.3 shows the mean, standard deviation, and range of each histogram. We show the all-polynomial results for completeness, but we re-emphasize that the all-polynomial surrogate does not conserve total energy. The degeneracy noted in Section 6.2.1 does not appear in these BOMD results since q_1 never drifts outside $[167.4, 192.0]$. For the N=N distance, the distribution of the mixed-basis trajectory is tight around its mean value since $r(\text{N=N})$ was optimized out when populating the sparse grid. For the design variables, certain distributions appear to have roughly the correct shape but are shifted right or left from BOMD. This is due to the surrogate PESs having slightly different equilibrium values for the *trans-* minimum.

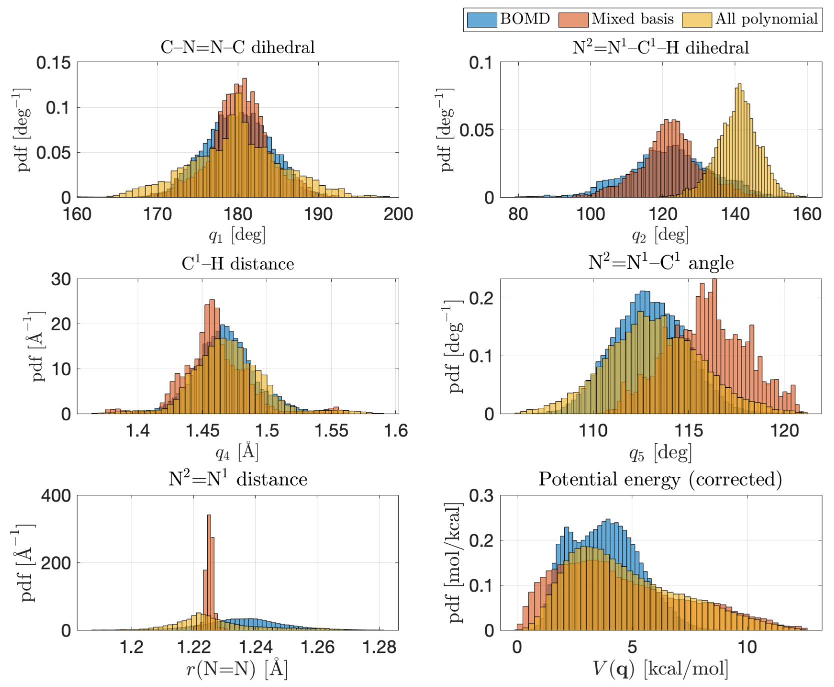


Figure 6.7 Histograms of selected design variables, remainder variables, and potential energy.

Since both reduced-dimensional surrogates use less than the full $3N-6$ internal coordinates, while BOMD uses all of them, we estimate the contribution of the remainder variables $\boldsymbol{\xi}$ in a manner analogous to Liu and coworkers [125]. At time step i, we add the contributions from $\boldsymbol{\xi}$ by setting

$$V_i^{corrected} = V_i^{surrogate} + \eta_i$$

where η_i is a uniform random variable on the interval $[0, \mathcal{H}_{BOMD} - \mathcal{H}_{surrogate}]$. Ideally, we would isolate the effects of the BOMD framework from the usage of additional design variables (i.e. by using a full-dimensional surrogate PES), but constructing the full 24-dimensional PES is computationally infeasible. As a result, the corrected distributions for $V(\boldsymbol{q})$ in Figure 6.7 do not match BOMD exactly.

6.2.4 Langevin thermostat

In this section, we focus only on the mixed-basis surrogate since it has been demonstrated that a polynomial surrogate will not properly capture energy when crossing a periodic boundary. We wish to study the effect of temperature on azomethane geometry. We start at the *trans*-minimum on the S_0 surface and integrate to 40 ps with a time step of $\Delta t = 0.05$ fs at various

Table 6.3 Statistics from NVE MD trajectories.

		BOMD	Mixed basis	All-polynomial
q_1 (deg)	mean	180.0	180.3	179.9
	stdev	4.0	3.7	5.8
	range	$[167.4, 192.0]$	$[168.0, 192.3]$	$[159.3, 198.6]$
q_2 (deg)	mean	122.2	122.1	140.8
	stdev	11.0	8.5	5.7
	range	$[77.6, 178.0]$	$[95.3, 147.7]$	$[119.5, 159.1]$
q_4 (Å)	mean	1.47	1.46	1.47
	stdev	0.02	0.03	0.03
	range	$[1.39, 1.56]$	$[1.37, 1.56]$	$[1.37, 1.59]$
q_5 (deg)	mean	112.9	115.9	113.2
	stdev	1.9	2.1	2.5
	range	$[106.9, 119.6]$	$[110.4, 121.0]$	$[106.1, 121.2]$
r(N=N) (Å)	mean	1.24	1.23	1.23
	stdev	0.013	0.001	0.013
	range	$[1.19, 1.28]$	$[1.22, 1.23]$	$[1.19, 1.28]$
$V(\boldsymbol{q})$ (kcal/mol)	mean	3.8	4.6	4.8
	stdev	1.5	2.8	2.4
	range	$[0.00, 8.9]$	$[0.02, 12.56]$	$[0.03, 12.53]$

target temperatures T.[2] The time step needs to be suitably small for Newton's method to converge. We used a relatively modest value for the friction coefficient, $\gamma = 0.01$ fs^{-1} [36, 125]. Initial velocities are drawn from a Boltzmann distribution at temperature T and then projected onto $\boldsymbol{p}(0)$.

We show the results in Figure 6.8. The average runtime of each thermostat simulation was 22.9 hours on XSEDE Bridges-2 with a single core [158, 212]. Even 3000 Kelvin is not hot enough to overcome the torsion transition structure ($q_1 \approx 90$). Furthermore, the dissociation energy of the C^1–N^1 bond is lower than the energy barrier of the torsion transition state [37]. As a result, isomerization is not energetically favorable via S_0 and temperature alone. In Figure 6.8(b), we have plotted the sample means and standard deviations of ensemble temperature versus their theoretical expectation values. Instantaneous temperature $\mathcal{T}(\boldsymbol{q}, \boldsymbol{p})$ is related to kinetic energy $K(\boldsymbol{q}, \boldsymbol{p})$ by

$$\mathcal{T}(\boldsymbol{q}, \boldsymbol{p}) = \frac{2K(\boldsymbol{q}, \boldsymbol{p})}{d\, k_B}$$

where d is the number of design variables. From the equipartition theorem [143], maximum

[2]We determined to need $\Delta t = 0.05$ fs for $T = 3000$ K, and we used it in all runs for consistency.

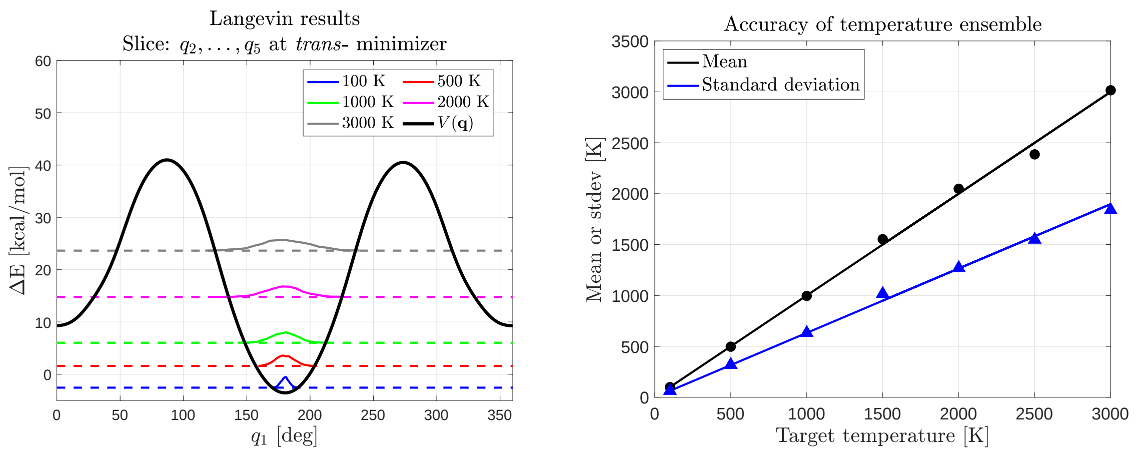


Figure 6.8 Left: Geometry distribution (solid) and average total energy (dashed lines) at different temperatures. Right: Predicted (lines) and observed (dots/triangles) statistical values for temperature.

likelihood estimators for the mean and standard deviation of $\mathcal{T}$ are given by

$$\mu(\mathcal{T}) = T$$
$$\sigma(\mathcal{T}) = \sqrt{2/d}\, T$$

where T is the target temperature of the thermostat. The theoretical and observed statistics agree closely with each other.

6.3 Conclusions

We have presented a mixed-basis interpolation algorithm that uses trigonometric interpolation on periodic design variables and polynomial interpolation on nonperiodic design variables. Unlike previous methods, it is not limited to molecules with purely periodic or purely nonperiodic reaction coordinates, and therefore is an improvement over the prior state of the art (same-basis interpolation) and widely generalizable to different systems. The results demonstrate that our method conserves total energy within accepted tolerances, is computationally efficient, and accurately reproduces the temperature distribution of a thermostat in a reduced-dimensional MD framework. The Python codes for mixed-basis interpolative PES approximation and reduced-dimensional MD are freely available on GitHub.[1]

This chapter has investigated only the lowest-lying singlet state of azomethane, but it is known that light-induced $S_0 \rightarrow S_1$ excitation is a likely isomerization and decomposition pathway for azomethane [36, 38, 185]. An extension of our surrogate PES approximation will be to implement a reduced-dimensional version of Tully's fewest-switches surface hopping algorithm.

CHAPTER

7 REDUCED-DIMENSIONAL SURFACE HOPPING

Molecular dynamics simulations often classically evolve the nuclear geometry on adiabatic potential energy surfaces (PESs), punctuated by random hops between energy levels in regions of strong coupling, in an algorithm known as *surface hopping*. However, the computational expense of integrating the geometry on a full-dimensional PES and computing the required couplings can quickly become prohibitive as the number of atoms increases. In this chapter, we describe a method for surface hopping that uses only important reaction coordinates, performs all expensive evaluations of the true PESs and couplings only once before simulating dynamics (offline), and then queries the stored values during the surface hopping simulation (online). Our Python codes are freely available on GitHub.[1] Using photodissociation of azomethane as a test case, this method is able to reproduce experimental results that have thus far eluded *ab initio* surface hopping studies.

Many important phenomena in chemistry are the result of nonadiabatic, femtosecond-level quantum transitions between electronic states [222]. Light-induced electronic transitions, for example, have attracted considerable recent study [169]. Applications include the mechanism of vision [78, 170], photophysics of DNA [159], photocatalysis [46, 49, 62], photovoltaics [31, 72, 129, 145, 187], and spectroscopy [9, 164, 197]. Due to the computational expense of simulating

[1] `https://github.com/zbmorrow/rrFSSH`

nonadiabatic processes with a fully quantum mechanical treatment, modern techniques favor a mixed quantum–classical dynamics (MQCD) approach [101, 222]. A popular and straightforward MQCD framework is the Ehrenfest (mean-field) method, where the nuclei evolve classically on a single potential energy surface (PES) that is the weighted average of the different quantum states [66, 175]. However, the mean-field approach is generally only valid in regions with weak coupling or similar nuclear behavior between quantum states [222].

Trajectory-based approaches seek to overcome the limitations of the pure mean-field approximation by using multiple PESs and computing PES couplings along a classical trajectory [169]. Indeed, these methods have become immensely popular over the last two and a half decades [169, 222]. Two of the most popular trajectory-based methods are multiple spawning [19, 136, 137, 220] and surface hopping [10, 85, 174, 176, 214]. Multiple spawning begins with Gaussian wave packets centered around the classical trajectory on a given PES and stochastically spawns new wave packets on a different PES when the classical trajectory approaches a conical intersection [136]. In this way, dynamics proceed on multiple PESs simultaneously. In surface hopping, however, the trajectory marches along a given PES and intermittently hops between surfaces in regions of strong coupling. Each surface hopping simulation runs on exactly one PES at any given time. The framework of multiple spawning treats quantum effects more rigorously from first principles [169], while surface hopping is attractive because it is straightforward to implement and analyze. The two methods typically produce similar results at similar costs if time integration is done efficiently [169]. Due to its ubiquity and simplicity, surface hopping is the MQCD approach we utilize here.

Computing the nonadiabatic coupling (NAC) vector required for surface hopping is an area of much recent activity, enabled by advances in computing power [74, 124, 186]. However, the computational expense of time-dependent density functional theory (TD-DFT) is still large enough that it is impractical to query the *ab initio* excitation energies and nonadiabatic couplings at each time point of every trajectory in the surface-hopping swarm.

Furthermore, the full $(3N-6)$-dimensional PES $\mathcal{E}_i(\boldsymbol{x})$ of an N-atom molecule in electronic state i typically includes only a handful of relevant reaction coordinates for a particular reaction. One can partition the geometry into the relevant coordinates (called *design variables*, $\boldsymbol{q} \in \mathbb{R}^d$) and everything else (called *remainder variables*, $\boldsymbol{\xi} \in \mathbb{R}^{3N-6-d}$). Then the so-called "relaxed PES" comes from optimizing over the remainder variables to produce a surface that is only a function of the design variables, as given by (1.2).

Due to the expense of evaluating the full or relaxed PES with an electronic structure program, construction of surrogate PESs is an active area of research. These surrogates allow the costly optimizations to be contained as an up-front cost in the offline phase, incurred only when first constructing the surrogate. Techniques of surrogate PES construction include permutationally invariant polynomials [2, 28, 29, 33], neural networks [99, 123, 135], interpolative moving least-squares [71, 83, 133, 134], modified Shepard interpolation [42–44, 209], Gaussian processes [48, 216], and the finite-element method [22]. Previous work in our group has approximated PESs with sparse interpolation. To build the surrogate, one simply needs to evaluate the true PES in the offline phase at a set of known design variable values $\{\boldsymbol{q}^j\}$. The Smolyak sparse interpolation algorithm ensures that the number of nodes $\boldsymbol{q}^j$ grows polynomially—rather than exponentially—in d, the number of design variables [84, 156, 192, 206]. In addition, a relaxed reduced-dimensional molecular dynamics (rr-MD) method, also from our group, allows for online MD simulations of the design variables only [125].

Construction of surrogate PESs with sparse interpolation in our group has used polynomial [145–147], trigonometric [140, 141], and mixed polynomial–trigonometric (Section 2.5) basis functions. The original development of trigonometric surrogate PESs occurred because energy would not be conserved in NVE rr-MD simulations with a polynomial surrogate PES if a component of $\boldsymbol{q}$ were periodic and crossed the periodic boundary, as we demonstrated in Chapter 6.

This chapter presents an implementation of Tully's fewest-switches surface hopping (FSSH) algorithm [214] in a reduced-dimensional framework (rr-FSSH) where all expensive electronic structure calculations are performed in the offline phase. We organize the remainder of this chapter as follows. In Section 7.1, we describe the computational details of our method: reduced-dimensional molecular dynamics, reduced-dimensional surface hopping, sparse grids, and the approximation of nonadiabatic couplings. In Section 7.2, we test our method on the photodissociation of azomethane in vacuo and compare against known experimental and *ab initio* results [30, 36–38, 53, 55, 138, 154, 183, 185].

7.1 Computational methods

In this section, we describe our online and offline computational methods. In the online phase, we run a swarm of molecular dynamics trajectories that rely on the surrogate PESs and NACs. In the offline phase, we use electronic structure programs to evaluate the required high-fidelity ground-state energies, excitation energies, and coupling vectors. The classical dynamics are governed by the reduced-dimensional molecular dynamics algorithms described in Section 6.1.1, along with mixed-basis sparse interpolation from Section 2.5. The rest of this section describes an implementation of surface hopping in a reduced-dimensional setting.

7.1.1 Reduced-dimensional FSSH

We now present the reduced-dimensional adaptation of Tully's immensely popular fewest-switches surface hopping method [214]. We let $\hat{H}_0(\boldsymbol{z}; \boldsymbol{X})$ denote the electronic Hamiltonian, where $\boldsymbol{z}$ is the electronic coordinate and $\boldsymbol{X}$ is the Cartesian geometry of the molecule. We opt to use the adiabatic eigenfunctions $\Phi_j(\boldsymbol{z};\boldsymbol{X})$ of $\hat{H}_0$ as the expansion basis. With this choice, we define the matrix elements

$$V_{ij}^{cart}(\boldsymbol{x}) = \langle \Phi_i(\boldsymbol{z};\boldsymbol{X}) | \hat{H}_0(\boldsymbol{z};\boldsymbol{X}) | \Phi_j(\boldsymbol{z};\boldsymbol{X}) \rangle = \mathcal{E}_i(\boldsymbol{X})\delta_{ij} \tag{7.1}$$

where $\mathcal{E}_i(\boldsymbol{X})$ is the full PES of state i as a function of Cartesian geometry, δ_{ij} is the Kronecker delta, and the brackets denote integration over $\boldsymbol{z}$. In Tully's original formulation, the nonadiabatic coupling vector in Cartesian coordinates is

$$\boldsymbol{d}_{ij}^{cart}(\boldsymbol{X}) = \langle \Phi_i(\boldsymbol{z};\boldsymbol{X}) | \nabla_{\boldsymbol{X}} \Phi_j(\boldsymbol{z};\boldsymbol{X}) \rangle \, . \tag{7.2}$$

However, we now want the couplings in terms of the design variables $\boldsymbol{q}$. Since $\boldsymbol{X} = \boldsymbol{X}(\boldsymbol{q})$ reconstructs the Cartesian geometry after optimizing the remainder variables, we get

$$V_{ij}(\boldsymbol{q}) = E_i(\boldsymbol{q})\delta_{ij} \tag{7.3}$$

The chain rule yields

$$\boldsymbol{d}_{ij}(\boldsymbol{q}) = \langle \Phi_i(\boldsymbol{z};\boldsymbol{X}(\boldsymbol{q})) | \nabla_{\boldsymbol{q}} \Phi_j(\boldsymbol{z};\boldsymbol{X}(\boldsymbol{q})) \rangle = \boldsymbol{X}'(\boldsymbol{q})^T \, \boldsymbol{d}_{ij}^{cart}(\boldsymbol{X}(\boldsymbol{q})) \, . \tag{7.4}$$

We now express the wavefunction of the electronic state of our system at time t in terms of the electronic basis functions

$$\Psi(\boldsymbol{z},\boldsymbol{q},t) = \sum_j c_j(t)\Phi_j(\boldsymbol{z};\boldsymbol{q}) \tag{7.5}$$

where we have expressed dependence directly in terms of $\boldsymbol{q}$ for notational ease. Combining (7.5) with the time-dependent Schrödinger equation, we may derive

$$\mathrm{i}\hbar\dot{c_k} = \sum_j \left(E_k(\boldsymbol{q})\delta_{kj} - \mathrm{i}\hbar\dot{\boldsymbol{q}} \cdot \boldsymbol{d}_{kj}(\boldsymbol{q}) \right) c_j \, . \tag{7.6}$$

With adiabatic electronic wavefunctions, the probability of transition from state i to state j during a time interval $[t, t+\Delta t]$ is [176, 214]

$$P(i \to j) = \max\left\{ 2\,\mathrm{Re}\left(\frac{c_j}{c_i}\,\dot{\boldsymbol{q}} \cdot \boldsymbol{d}_{ij}(\boldsymbol{q}) \right) \Delta t, \ 0 \right\} . \tag{7.7}$$

We integrate (7.6) with the Crank–Nicolson method [47, 76], which conserves the ℓ^2 norm of the solution, so that all state occupations sum to 1. The time step required to integrate (7.6) accurately is much smaller than that used for the classical dynamics (6.2), so we linearly interpolate all relevant quantities during intermediate steps [85, 214].

The transition probability in (7.7) is evaluated at each classical time step. If a switch from state i to j occurs at $t = t_n$ and $E_j(\boldsymbol{q}_n) \neq E_i(\boldsymbol{q}_n)$, then we must adjust the momentum $\boldsymbol{p}_n$ to conserve total energy [214]. Similarly to Hammes–Schiffer and Tully [85], we set

$$\boldsymbol{p}_n^{corr} = \boldsymbol{p}_n - \alpha\, \boldsymbol{d}_{ij}(\boldsymbol{q}_n) \tag{7.8}$$

and solve for α in light of (6.3)-(6.5), yielding the equation

$$\frac{1}{2}(\boldsymbol{d}^T \boldsymbol{G}^{-1}\boldsymbol{d})\alpha^2 - (\boldsymbol{p}^T \boldsymbol{G}^{-1}\boldsymbol{d})\alpha + (E_j - E_i) = 0\,. \tag{7.9}$$

Above, for notational simplicity, $\boldsymbol{d} = \boldsymbol{d}_{ij}$ and all quantities are evaluated at $t = t_n$ or $\boldsymbol{q} = \boldsymbol{q}_n$. If $E_j > E_i$ and momentum is insufficient to overcome the energy gap, i.e.

$$(\boldsymbol{p}^T \boldsymbol{G}^{-1}\boldsymbol{d})^2 - 2(\boldsymbol{d}^T \boldsymbol{G}^{-1}\boldsymbol{d})(E_j - E_i) < 0, \tag{7.10}$$

then we have a frustrated hop. In this case, we reflect momentum in the direction of $\boldsymbol{d}$ by setting

$$\alpha = 2\,\frac{\boldsymbol{p}^T \boldsymbol{G}^{-1}\boldsymbol{d}}{\boldsymbol{d}^T \boldsymbol{G}^{-1}\boldsymbol{d}}\,. \tag{7.11}$$

If, however, the inequality in (7.10) is reversed and two solutions of (7.9) exist, we have

$$\alpha = \frac{(\boldsymbol{p}^T \boldsymbol{G}^{-1}\boldsymbol{d}) \pm \sqrt{(\boldsymbol{p}^T \boldsymbol{G}^{-1}\boldsymbol{d})^2 - 2(\boldsymbol{d}^T \boldsymbol{G}^{-1}\boldsymbol{d})(E_j - E_i)}}{\boldsymbol{d}^T \boldsymbol{G}^{-1}\boldsymbol{d}} \tag{7.12}$$

and we take the "+" solution if $\boldsymbol{p}^T \boldsymbol{G}^{-1}\boldsymbol{d} < 0$ and the "−" solution otherwise [85]. We then proceed with the dynamics in (6.2) using $V(\boldsymbol{q}) = E_j(\boldsymbol{q})$, $\boldsymbol{q} = \boldsymbol{q}_n$, and $\boldsymbol{p} = \boldsymbol{p}_n^{corr}$.

7.1.2 Approximation of nonadiabatic couplings

The computation of $\boldsymbol{d}_{ij}^{cart}$ in (7.2) is relatively costly, so we would like to avoid doing it at every time step in the online phase. The Landau–Zener formula [116, 225, 230] is a popular approach to computing the transition probability (7.7) because it avoids NACs and is derived from first principles. However, the Landau–Zener formula is only applicable where the energy difference $E_i(\boldsymbol{q}) - E_j(\boldsymbol{q})$ is at a local minimum along the trajectory $\boldsymbol{q} = \boldsymbol{q}(t)$ [18, 207]. Another possibility is to define custom diabatic wavefunctions that allow for easier evaluation of the NACs [37], but such methods tend to be molecule-specific. We desire a general, molecule-blind method of querying the transition probability at all time steps and not only those corresponding to a local minimum of the energy difference along the trajectory.

Many components of the coupling vector will peak sharply in a small region and be zero over large portions of the design variable domain, behavior which is not amenable to interpolation with globally defined basis functions. As a result, we compute $\boldsymbol{d}_{ij}^{cart}$ in the offline phase at a list of design variable values known mathematically to possess a space-filling property known as *low discrepancy* [56, 151]. At every time point in the online phase, we approximate the NAC along the simulation trajectory using the known coupling at the nearby space-filling points.

We obtain the space-filling points from the Sobol' sequence [193], which is implemented in MATLAB and part of a class of sequences known as quasi-random. Such sequences are deterministic but fill a domain in much the same way that uniformly distributed random variables would. However, the number of quasi-random points needed to fill a domain is much lower than that of random variables. Quasi-random sequences are the core of quasi-Monte Carlo integration, which has significantly faster convergence than fully stochastic Monte Carlo techniques [56, 151] and is an active area of research [3, 75, 88, 142].

Using MATLAB, we obtain a set of Sobol' points $\{\boldsymbol{q}_m^{sobol}\} \subset [0,1]^d$ such that no point is more than 0.02 away from its nearest neighbor. We then transform these points, in place, by converting each component of $\boldsymbol{q}_m^{sobol}$ from $[0,1]$ to the corresponding physical domain (see Table 7.1). We evaluate the Cartesian NACs (7.2) at the geometries $\boldsymbol{X}(\boldsymbol{q}_m^{sobol})$, $m = 1, \ldots, N_{sobol}$.

At time $t = t_n$ in the online phase, we approximate $\boldsymbol{d}_{ij}^{cart}(\boldsymbol{q}_n)$ as the weighted average of all $\boldsymbol{d}_{ij}^{cart}(\boldsymbol{q}_m^{sobol})$ within some user-adjustable radius R of $\boldsymbol{q}_n$. (For notational simplicity, we express dependence of $\boldsymbol{d}_{ij}^{cart}$ only on $\boldsymbol{q}$.) We define the set

$$S_n(R) = \{m \in \mathbb{N},\ 1 \leq m \leq N_{sobol}\ :\ \|\boldsymbol{D}^{-1}(\boldsymbol{q}_m^{sobol} - \boldsymbol{q}_n)\|_2 \leq R\}\,, \tag{7.13}$$

where $\boldsymbol{D}$ is a diagonal matrix with D_{ii} being the length of the physical domain of component q_i. We include $\boldsymbol{D}$ to account for different length scales across components. If $S_n(R)$ is empty, then we use the nearest neighbor:

$$\boldsymbol{d}_{ij}^{cart}(\boldsymbol{q}_n) \approx \boldsymbol{d}_{ij}^{cart}\left(\underset{1 \leq m \leq N_{sobol}}{\arg\min}\ \|\boldsymbol{D}^{-1}(\boldsymbol{q}_m^{sobol} - \boldsymbol{q}_n)\|_2\right)\,. \tag{7.14}$$

Otherwise, we compute the weights

$$\begin{aligned} w_m &= \frac{b_m}{\sum_{k \in S_n(R)} b_k}, \qquad m \in S_n(R), \\ b_m &= \exp\left(-\frac{\|\boldsymbol{D}^{-1}(\boldsymbol{q}_m^{sobol} - \boldsymbol{q}_n)\|_2}{\min_{k \in S_n(R)} \|\boldsymbol{D}^{-1}(\boldsymbol{q}_k^{sobol} - \boldsymbol{q}_n)\|_2}\right), \qquad m \in S_n(R), \end{aligned}$$

and take

$$\boldsymbol{d}_{ij}^{cart}(\boldsymbol{q}_n) \approx \sum_{m \in S_n(R)} w_m \, \boldsymbol{d}_{ij}^{cart}(\boldsymbol{q}_m^{sobol}) \,. \tag{7.15}$$

Lastly, we use Equation (7.4) and continue with the trajectory.

While this approach will be less accurate than exactly evaluating $\boldsymbol{d}_{ij}$ at each classical time step, it will also be less costly since we perform only N_{sobol} evaluations of the NACs. In our examples, we have $N_{sobol} = 10000$, and with possibly thousands of members of a trajectory swarm, the computational savings quickly become apparent. Furthermore, the maximum nearest-neighbor distance of the Sobol points at most 0.02, so we have a robust set of data with which to compute (7.15).

7.2 Results and discussion

In this section, we describe the simulation setup and present *ab initio* results for our test reaction: the photodissociation of gaseous azomethane in a vacuum.

7.2.1 Electronic structure calculations

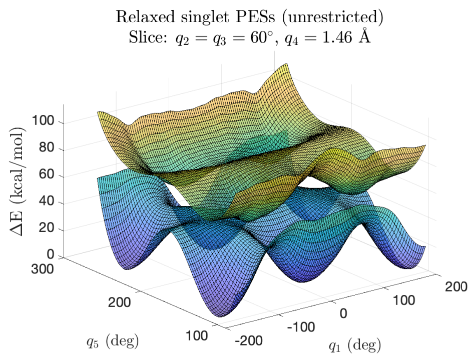
We perform ground-state density-functional theory (DFT) optimizations of azomethane at the nodes of the mixed-basis grid in Gaussian 16 [73] at the B3LYP/6-311G* level of theory [15, 16, 112, 117, 198]. After each geometry optimization, we perform stability analysis [13] to determine whether a closed- or open-shell wavefunction yields lower energy. If an instability is found, the geometry is reoptimized. We use Orca v4.2.1 [149, 150] to compute $S_0 \to S_1$ excitation energies (TD-DFT) at the optimized ground-state geometries, as well as Hellmann–Feynman nonadiabatic couplings at the Sobol' points (CIS). We use the stability-tested wavefunction to compute the NACs. We have included example input files for Gaussian and Orca in Appendix A.2. We refine the sparse grid until the energy differences of the minima and transition states on the S_0 surface are within $\sim$5% of their Gaussian-optimized value (Tables A.2 and A.3) and $\boldsymbol{X}(\boldsymbol{q})$ is smoothly varying. The final mixed-basis grid has 7215 nodes.[†]

We have previously described the design variables $\boldsymbol{q}$ (Figure 6.2) in great detail in Chapter 6, but we briefly reiterate them here. Table 7.1 summarizes the $d = 5$ design variables we use for this study, chosen to capture rotation, inversion, and dissociation transition states and strongly motivated by Cattaneo and Persico's 2001 study [38]. A preliminary 1998 study by Cattaneo and Persico [36] simplified the molecular geometry by treating the two methyl groups as point masses, which unfortunately eliminated several high- and low-frequency normal modes [38]. We opted to include the two methyl dihedrals not only to capture two of the low-frequency modes, like Cattaneo and Persico's 2001 study, but also to ensure smoothness of the interpolated $\boldsymbol{X}(\boldsymbol{q})$.

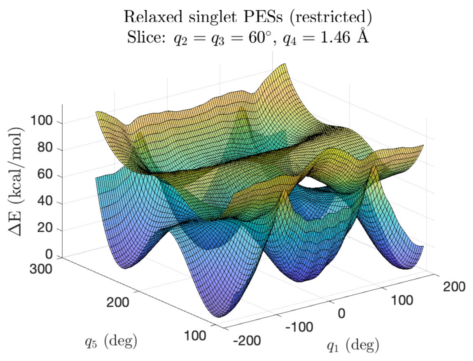
Table 7.1 List of design variables from Figure 6.2.

Label	Structural coordinate	Domain	Periodic?
q_1	C^2–N^2–N^1–C^1 torsion	$[-180, 180]$ deg	Y
q_2	N^2–N^1–C^1–H torsion	$[-180, 180]$ deg	Y
q_3	N^1–N^2–C^2–H' torsion	$[-180, 180]$ deg	Y
q_4	N^1–C^1 distance	$[1.1, 2.5]$ Å	N
q_5	N^2–N^1–C^1 angle	$[90, 270]$ deg	N

As described in Chapter 6, since q_5 can be linear or larger than 180°, we must do internal bookkeeping to encode the variables appropriately in an electronic structure program. Specifically, if $q_5 > 180$, then we add q_1 to 180° and subtract q_5 from 360. Since the linear structure is within the q_5 domain, we also must take care to avoid multivalued geometries as $q_5 \to 180$. We accomplish this by offsetting q_2 by $(q_1 + 180)$ whenever $q_5 \neq 180$.



(a)



(b)

Figure 7.1 (a) Mixed-basis S_0 and S_1 PESs (kcal/mol) with stability-checked wavefunctions. Energies are relative to the optimized *trans*- structure. (b) PESs using restricted wavefunctions.

In Figure 7.1a, we show a slice of the mixed-basis S_0 and S_1 PESs constructed with stability checks. Unlike previous work [36, 185], the unrestricted PES does not have a crossing seam, leading to small nonadiabatic coupling between the S_0 and S_1 surfaces. Indeed, we will demonstrate this phenomenon in the following section with a surface hopping swarm; see Figures 7.2, 7.3c, and 7.3e and associated discussion. To capture the crossing seam, we construct the S_0 PES by performing single-point energy calculations with restricted (closed-shell) wavefunctions at the geometries that were optimized with stability checks. We do this because directly performing

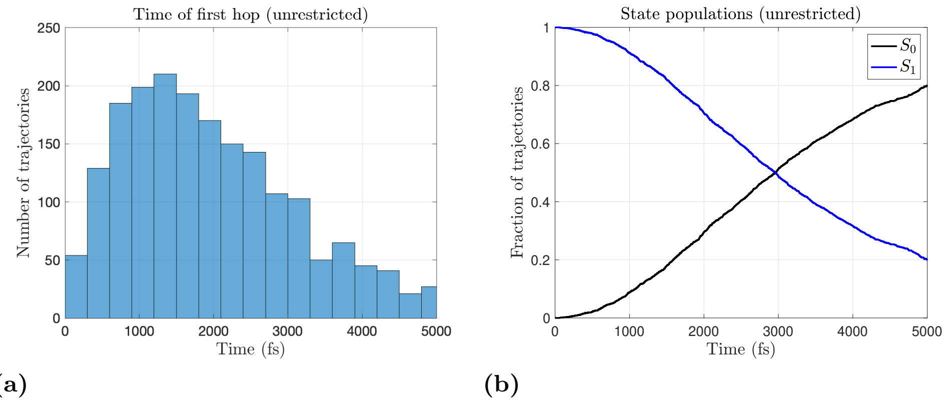


Figure 7.2 (a) First hopping times using unrestricted PESs and NACs. (b) State populations for the same.

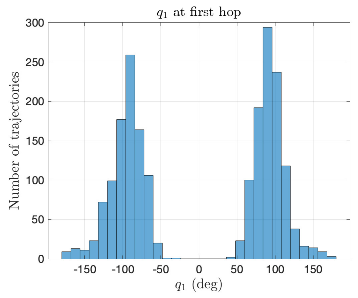
constrained optimizations with restricted wavefunctions yielded undesirable structures, in which hydrogen atoms migrate between the C and N atoms. We show the resulting restricted S_0 PES in Figure 7.1b. For this PES, we compute the couplings using restricted wavefunctions only. We did not observe meaningful changes in the S_1 PES between the two cases. Therefore, in both cases we employed the S_1 PES constructed from the TD-DFT calculations using the restricted S_0 PES as a reference.

7.2.2 Surface hopping swarm

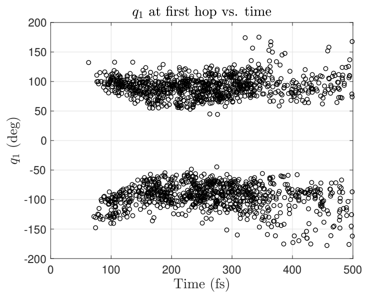
We sample 2000 initial geometries and momenta by running a reduced-dimensional Langevin thermostat (Section 6.1.1) on the S_0 PES at 298.15 K ($\gamma = 0.01$ fs^{-1}), starting at the *trans*-conformation. The experimental boiling point of azomethane at 1 atm is 273.45 K [63], which ensures that we are in the gaseous regime. We integrate the Langevin equations for a burn-in period of 10 ps and then sample every 20 fs to construct an ensemble of $\boldsymbol{q}_0$ and $\boldsymbol{p}_0$, similarly to Cattaneo and Persico [36, 38].

For each ensemble member, we excite vertically to the S_1 surface but keep $\boldsymbol{q}_0$ and $\boldsymbol{p}_0$ unchanged. We integrate the classical dynamics (6.2) with a time step of $\Delta t_c = 0.25$ fs and the quantum amplitudes (7.6) with $\Delta t_q = 0.01$ fs, up to 5 ps. The transition probabilities and momenta adjustments (including for frustrated hops) are given in Section 7.1.1. We approximate the NACs according to Section 7.1.2 using 10000 space-filling Sobol' points[†] and the cutoff radius $R = 0.05$. The computing environment is XSEDE Bridges-2 [158, 212], where each trajectory receives 2 GB RAM and one core of an AMD EPYC 7742 CPU. All codes are freely available on GitHub.[1]

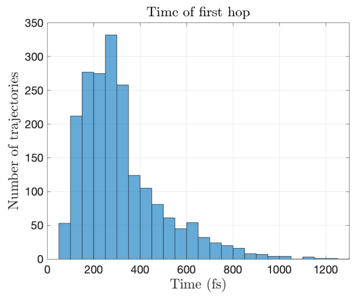
We show the results of surface hopping with the unrestricted PES and couplings in Figure 7.2.



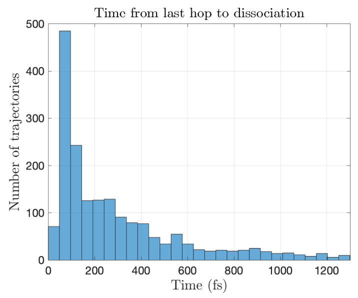
(a)



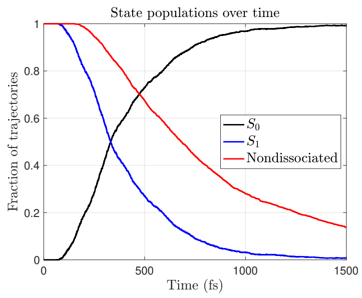
(b)



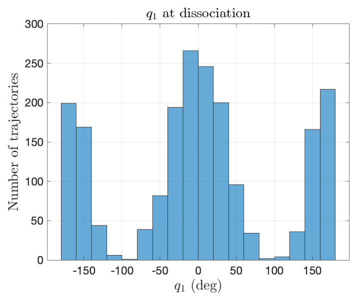
(c)



(d)



(e)



(f)

Figure 7.3 Results using the restricted PESs and NACs. (a) Values of q_1 at first hop. (b) Values of q_1 vs time of first hop. (c) Time of first $S_1 \to S_0$ transition. (d) Lag between final $S_1 \to S_0$ transition and dissociation. (e) Populations of the S_0/S_1 states and undissociated molecules. (f) Values of q_1 at dissociation.

The median hopping time is 1760 fs, which is far larger than previous theoretical predictions of 100–500 fs [36–38, 138, 183, 185]. Indeed, the S_1 population persists even after 5000 fs (Figure 7.2b). Therefore, we now use the restricted S_0 PES described in the previous section. We compare the maximum elements of the NAC vectors between the two cases, and we find that S_0/S_1 couplings are 20 times stronger for the restricted S_0 surface than for the unrestricted S_0 surface. We explain this observation by noting that the unrestricted S_0 PES exhibits an avoided crossing near $q_1 \approx \pm 90$, while the restricted S_0 PES shows a crossing seam in the same region.

We show the results for the restricted PES in Figure 7.3. All trajectories dissociated within 5 ps (defined as $q_4 > 2.5$ Å), and all dissociations occur on the S_0 surface. From Figs. 7.3a and 7.3b, we can see that the molecule is near the S_0/S_1 crossing seam ($q_1 \approx \pm 90°$) when the first hop occurs. The slight curvature in the swarms of Figure 7.3b reflects the geometry of the molecule cycling through the minimum on the S_1 surface. The median time of first hop is 276 fs (Figure 7.3c), and the median lag time between final hop and dissociation is 224 fs (Figure 7.3d), implying that dissociation happens relatively quickly after internal conversion (IC). We fit the

S_1 and nondissociated populations of Figure 7.3e to the exponential decay function

$$y = \exp\left(-(t - t_1)/t_2\right) \tag{7.16}$$

where t_1 and t_2 are the latency and decay times, respectively. The overall lifetime is $\tau = t_1 + t_2$. For the S_1 population, we obtain $t_1 = 78$ fs and $t_2 = 342$ fs, yielding a lifetime of $\tau = 420$ fs. For the lifetime of the nondissociated molecule, we find $t_1 = 188$ fs and $t_2 = 675$ fs, yielding a lifetime of $\tau = 863$ fs. We note that the latency time of the nondissociated population decay is smaller than the lifetime of the S_1 population, again showing an overlap between the time scales of IC and dissociation.

Figure 7.3f shows the values of q_1 at dissociation. Upon dissociation, we find very few trajectories in the vicinity of the crossing seam; most have relaxed into either *cis*- or *trans*- structures. If, like Cattaneo and Persico [38], we define *trans*- as $q_1 \in [-180, -150] \cup [150, 180)$ and *cis*- as $q_1 \in [-30, 30]$ (for $q_5 < 180$), then 31% of trajectories ended in *trans* and 37% in *cis*. This is in agreement with the vibrational frequency of C–N–N–C torsion (290.90 cm^{-1}), which yields a period of 115 fs. Since dissociation occurs around 224 fs after the last hop (in the median), then there is ample time to relax to the *cis*- or *trans*- conformations.

We also test the robustness of our method with respect parameter choices by altering various values while holding all others constant: $\Delta t_c = 0.05$ fs, $\Delta t_q = 0.001$ fs; $N_{sobol} = 5000$; $N_{sobol} = 1000$; $R = 0.20$; $R = 10^{-4}$ (forced nearest-neighbor); and $R = 10^{-4}$, $N_{sobol} = 1000$. Here, R is the radius in Equation (7.13). We display the results in Table 7.2. Our results also appear to be consistent across parameter choices, with the exception of excessively large radius R. Refining the time step does not significantly alter the results. Similarly, forcing the nearest-neighbor NAC approximation ($R = 10^{-4}$) does not cause a large change in the outputs. The nearest-neighbor NAC approximation appears to comparable in accuracy to the weighted average—without needing to compute exponential weights. Furthermore, if the evaluation of the NACs in the offline phase is quite costly, these results indicate that one may use a smaller number of Sobol' points ($N_{sobol} = 5000$, $d = 5$) to obtain lifetimes in agreement with previous *ab initio* studies. However, if N_{sobol} is too small, the accuracy tends to degrade somewhat ($N_{sobol} = 1000$ in our examples).

7.2.3 Discussion

The photochemical and thermal properties of azoalkanes have a long history of study [67, 179, 180, 185], particularly azomethane, as the simplest member of this class of compounds. This wealth of prior work makes azomethane a desirable test system. Recent literature is in very good agreement on the lifetime of azomethane in the S_1 state: 70–100 fs (experimental) [55] and 100–500 fs (theoretical) [36–38, 138, 183, 185]. The spread within theoretical results is largely due to

Table 7.2 Comparison of results for various parameter modifications.

	S_1 time constants			
Alteration	t_1 (fs)	t_2 (fs)	τ (fs)	Med. hop–dissoc. gap (fs)
None (baseline)	78	342	420	224
$\Delta t_c = 0.05$, $\Delta t_q = 0.001$	108	285	393	232
$N_{sobol} = 5000$	119	324	443	232
$N_{sobol} = 1000$	140	407	547	241
$R = 0.20$	190	790	980	230
$R = 10^{-4}$	109	290	399	236
$N_{sobol} = 1000$, $R = 10^{-4}$	142	404	546	236

a variety of model chemistries and simulation parameters. Our computed least-squares lifetime of $\tau = 420$ fs, obtained with reduced-dimensional dynamics and a relatively simple underlying model chemistry, is in excellent agreement with the previous theoretical results. Also, a series of studies within the last three decades has established that the photodissociation of azomethane happens stepwise in the S_0 state [36, 38, 53, 55, 154]:

$$\mathrm{CH_3NNCH_3} \rightarrow \mathrm{CH_3NN}\cdot + \mathrm{CH_3}\cdot \tag{7.17}$$

$$\mathrm{CH_3NN}\cdot + \mathrm{CH_3}\cdot \rightarrow \mathrm{N_2} + 2\mathrm{CH_3}\cdot \tag{7.18}$$

with the second dissociation very quickly (femtosecond-level) following the first. Because this reaction mechanism is well-established, we include only one C–N bond length in our design variables, focusing on the first step of the mechanism.

However, as noted by Sellner and coworkers [185], the time scale of dissociation following $S_0 \leftarrow S_1$ de-excitation is the subject of interesting, unresolved debate. Dissociation occurs either on a picosecond time-scale after relaxation to *trans-* or *cis-* conformations (statistical model), or on a femtosecond time-scale shortly after de-excitation (impulsive model). Lee's group observed experimental results favoring the statistical model [30, 154], while the experimental findings of Zewail's group favor the impulsive model [53, 55]. As noted by Zewail's group in *Science*, many fast femtosecond-level reactions do in fact occur, though they violate Rice–Ramsperger–Kassel–Marcus (RRKM) theory and the underlying assumption of statistical redistribution of vibrational energy [54].

In spite of the split experimental findings, all recent *ab initio* simulations of azomethane photodissociation favor the statistical model. Cattaneo and Persico [38] observed only 20% of dissociations within 1 ps of initial $S_0 \rightarrow S_1$ excitation, with 10% occurring before 400 fs and almost none before 250 fs. The same study required 100 ps before 75% of trajectories dissociated. Simulations from Lischka's group [183, 185] using the Newton-X package [10] observed a small number of dissociations ($\sim$5%) prior to 500 fs when additional vibrational quanta are added to the torsional rotation. Without additional torsional bias, no dissociations occurred prior to 500 fs [183, 185].

Unlike previous *ab initio* studies, our results favor a dissociation time faster than the ~ 1 ps prediction of the statistical model. Experimentally, Zewail's group found a 70–100 fs rise time of the $CH_3NN\cdot$ fragment, clearly favoring the impulsive model. In our simulations, 50% of trajectories dissociated within 700 fs of initial $S_0 \rightarrow S_1$ excitation and 72% within 1 ps, so subpicosecond dissociation is dominant. Moreover, our median time between the last hop and dissociation is on the order of 200 fs; there are intermediate hops back to S_1 before the trajectory finally settles on S_0. Though our results do not resolve the statistical–impulsive debate, they provide evidence that the impulsive findings of Zewail's group merit further study and could be justifiable on *ab initio* grounds.

CHAPTER

8 CONCLUSION

This dissertation has presented novel methodological developments within sparse grids as well as novel applications to molecular dynamics (MD) simulations and potential energy surface (PES) approximation. In Chapter 2, we began by discussing prior work on sparse grids for both polynomial and trigonometric basis choices [11, 84, 155, 156, 206]. We worked upwards from one-dimensional rules to sparse grids and presented exactness results and error estimates in both cases. We then presented a dimensionally adaptive refinement algorithm for sparse trigonometric interpolation that estimates the smoothness in each dimension based on the decay rates of the interpolation coefficients [139]. We concluded by presenting mixed-basis interpolation method that uses a tensor product of polynomial and trigonometric sparse grids, which can be applied to functions of mixed periodicity.

Chapter 3 provided an overview of the Tasmanian sparse grid package [200], where the author implemented sparse trigonometric interpolation and the adaptive refinement method. In Chapter 4, we gave an introduction to quantum and computational chemistry, beginning with exactly solvable problems and then discussing methods of approximating solutions to the many-body Schrödinger equation. We concluded Chapter 4 with a discussion of density functional theory (DFT), an immensely popular method for electronic structure calculations.

The remaining chapters described applications of sparse grids to different chemical systems. In Chapter 5, we applied sparse trigonometric interpolation to a tungsten molecule and demonstrate that the trigonometric basis accurately captures the periodicity of the PES gradient, while the polynomial basis does not. We showed additionally that the trigonometric interpolant is more accurate than the polynomial interpolant.

In Chapter 6, we uses mixed polynomial–trigonometric interpolation on azomethane to demonstrate the validity and practicality of mixed-basis interpolation. We constructed a mixed-basis surrogate PES and used it in reduced-dimensional MD simulations [125]. We compared this surrogate to an all-polynomial surrogate, the previous state of the art. We found that the mixed-basis surrogate leads to energy conservation, whereas the all-polynomial surrogate clearly fails to do so. In addition, the mixed-basis PES required fewer electronic structure calculations to reach a given level of accuracy compared to the all-polynomial PES. In Chapter 7, we turned our attention to nonadiabatic dynamics, which are important for light-induced phenomena following photoexcitation. We presented a modification of Tully's fewest-switches surface hopping (FSSH) algorithm [85, 214] in a reduced-dimensional setting. We then combined interpolation-enabled MD with an offline–online method of estimating expensive nonadiabatic coupling vectors.

The adaptive refinement algorithm presented in Chapter 2 is able to improve the convergence rate of sparse trigonometric interpolation over isotropic refinement, and it learns the anisotropy of the target function on the fly, without prior knowledge. The mixed-basis interpolant in Chapter 6 demonstrates a way to preserve periodicity where needed and not impose periodicity where it does not exist. As such, mixed-basis interpolation is applicable to a wider class of systems than prior work, which either needed all-periodic design variables (trigonometric interpolation in Chapter 5) or no periodic variables (if energy is to be conserved with polynomial interpolation). Mixed-basis interpolation conserves the energy in a microcanonical ensemble within chemically accepted metrics, whereas all-polynomial interpolation does not. Additionally, the simulations with mixed-basis surrogates are much less computationally expensive. The surface hopping algorithm presented in Chapter 7 is applicable in reduced-dimensional settings that are less expensive than using the full-dimensional PES. Furthermore, the method of approximating the nonadiabatic coupling vector with Sobol' points provides a means of controlling the computational effort of a surface hopping simulation. We applied this approach to photodissociation of azomethane and found that our method was able to reproduce experimental results that have thus far eluded *ab initio* studies. In the future, this framework (combining sparse grids, reduced-dimensional MD, and offline computation of nonadiabatic couplings) can be used to study nonadiabatic dynamics in more complex molecules, such as those in solar cells [72, 129, 145].

REFERENCES

[1] Adams, R. D., Collins, D. M. and Cotton, F. A. "Molecular Structures and Barriers to Internal Rotation in Bis(η^5-cyclopentadienyl)hexacarbonylditungsten and Its Molybdenum Analog." *Inorg. Chem.* **13**.5 (1974), pp. 1086–1090. DOI: `10.1021/ic50135a015`.

[2] Albaugh, A., Boateng, H. A., Bradshaw, R. T., Demerdash, O. N., Dziedzic, J., Mao, Y., Margul, D. T., Swails, J., Zeng, Q., Case, D. A., Eastman, P., Wang, L.-P., Essex, J. W., Head-Gordon, M., Pande, V. S., Ponder, J. W., Shao, Y., Skylaris, C.-K., Todorov, I. T., Tuckerman, M. E. and Head-Gordon, T. "Advanced Potential Energy Surfaces for Molecular Simulation." *J. Phys. Chem. B* **120**.37 (2016), pp. 9811–9832. DOI: `10.1021/acs.jpcb.6b06414`.

[3] Alexandrov, V., Davila, D., Esquivel-Flores, O., Karaivanova, A., Gurov, T. and Atanassov, E. "On Monte Carlo and Quasi-Monte Carlo for Matrix Computations." *Large-Scale Scientific Computing*. Ed. by Lirkov, I. and Margenov, S. Springer International Publishing, 2018, pp. 249–257.

[4] Allen, M. P. and Tildesley, D. J. *Computer Simulations of Liquids*. 2nd. Oxford University Press, 2017.

[5] Andrews, B. K., Burton, K. A. and Weisman, R. B. "Dynamics of the Two-Step Photodissociation of Azomethane." *J. Chem. Phys.* **96**.2 (1992), pp. 1111–1120. DOI: `10.1063/1.462197`.

[6] Ashley, D. C. and Jakubikova, E. "Ray-Dutt and Bailar Twists in Fe(II)-Tris(2,2'-bipyridine): Spin States, Sterics, and Fe–N Bond Strengths." *Inorg. Chem.* **57**.9 (2018), pp. 5585–5596. DOI: `10.1021/acs.inorgchem.8b00560`.

[7] Atkinson, K. and Han, W. *Theoretical Numerical Analysis: A Functional Analysis Framework*. Springer, 2001.

[8] Babenko, K. "Approximation by Trigonometric Polynomials in a Certain Class of Periodic Functions of Several Variables." *Dokl. Akad. Nauk SSSR* **132** (1960), pp. 982–985.

[9] Barbatti, M., Aquino, A. J. A. and Lischka, H. "The UV Absorption of Nucleobases: Semi-Classical Ab Initio Spectra Simulations." *Phys. Chem. Chem. Phys.* **12**.19 (2010), pp. 4959–4967. DOI: `10.1039/B924956G`.

[10] Barbatti, M., Ruckenbauer, M., Plasser, F., Pittner, J., Granucci, G., Persico, M. and Lischka, H. "Newton-X: A Surface-Hopping Program for Nonadiabatic Molecular Dynamics." *WIREs Comput. Mol. Sci.* **4**.1 (2014), pp. 26–33. DOI: `10.1002/wcms.1158`.

[11] Barthelmann, V., Novak, E. and Ritter, K. “High Dimensional Polynomial Interpolation on Sparse Grids.” *Adv. Comput. Math.* **12**.4 (2000), pp. 273–288. DOI: 10.1023/A:1018977404843.

[12] Baszenski, G. and Delvos, F.-J. “Multivariate Approximation Theory IV.” Ed. by C. K. Chui W. Schempp, K. Z. Birkhaüser Basel, 1989. Chap. A discrete Fourier transform scheme for Boolean sums of trigonometric operators, pp. 15–24.

[13] Bauernschmitt, R. and Ahlrichs, R. “Stability Analysis for Solutions of the Closed Shell Kohn–Sham Equation.” *J. Chem. Phys.* **104**.22 (1996), pp. 9047–9052. DOI: 10.1063/1.471637.

[14] Beck, J., Nobile, F., Tamellini, L. and Tempone, R. “Convergence of Quasi-Optimal Stochastic Galerkin Methods for a Class of PDEs with Random Coefficients.” *Comput. Math. Appl.* **67**.4 (2014), pp. 732–751. DOI: 10.1016/j.camwa.2013.03.004.

[15] Becke, A. D. “A New Mixing of Hartree-Fock and Local Density-Functional Theories.” *J. Chem. Phys.* **98**.2 (1993), pp. 1372–1377. DOI: doi:http://dx.doi.org/10.1063/1.464304.

[16] Becke, A. D. “Density-Functional Thermochemistry. III. The Role of Exact Exchange.” *J. Chem. Phys.* **98**.7 (1993), pp. 5648–5652. DOI: doi:http://dx.doi.org/10.1063/1.464913.

[17] Bellman, R. *Adaptive Control Processes: A Guided Tour.* Princeton University Press, 1961.

[18] Belyaev, A. K., Lasser, C. and Trigila, G. “Landau–Zener Type Surface Hopping Algorithms.” *J. Chem. Phys.* **140**.22 (2014), p. 224108. DOI: 10.1063/1.4882073.

[19] Ben-Nun, M. and Martínez, T. J. “Nonadiabatic Molecular Dynamics: Validation of the Multiple Spawning Method for a Multidimensional Problem.” *J. Chem. Phys.* **108**.17 (1998), pp. 7244–7257. DOI: 10.1063/1.476142.

[20] Berweger, C. D., Gunsteren, W. F. van and Müller-Plathe, F. “Molecular Dynamics Simulation with an Ab Initio Potential Energy Function and Finite Element Interpolation: The Photoisomerization of cis-Stilbene in Solution.” *J. Chem. Phys.* **108**.21 (1998), pp. 8773–8781. DOI: 10.1063/1.475397.

[21] Berweger, C. D., Gunsteren, W. F. van and Müller-Plathe, F. “The Photoisomerization of cis-Stilbene Does Not Follow the Minimum Energy Path.” *Angew. Chem. Int. Edit.* **38**.17 (1999), pp. 2609–2611. DOI: 10.1002/(SICI)1521-3773(19990903)38:17<2609::AID-ANIE2609>3.0.CO;2-C.

[22] Berweger, C. D., Gunsteren, W. F. van and Müller–Plathe, F. “Finite Element Interpolation for Combined Classical/Quantum Mechanical Molecular Dynamics Simulations.” *J.*

Comput. Chem. **18**.12 (1998), pp. 1484–1495. DOI: 10.1002/(SICI)1096-987X(199709)18:12<1484::AID-JCC6>3.0.CO;2-F.

[23] Binkley, J. S., Pople, J. A. and Hehre, W. J. "Self-Consistent Molecular Orbital Methods. 21. Small Split-Valence Basis Sets for First-Row Elements." *J. Am. Chem. Soc.* **102**.3 (1980), pp. 939–947. DOI: 10.1021/ja00523a008.

[24] Boilleau, C., Suaud, N. and Guihéry, N. "Ab Initio Study of the Influence of Structural Parameters on the Potential Energy Surfaces of Spin-Crossover Fe(II) Model Compounds." *J. Chem. Phys.* **137**.22 (2012), p. 224304. DOI: 10.1063/1.4768870.

[25] Bolhuis, P. G., Chandler, D., Dellago, C. and Geissler, P. L. "Transition Path Sampling: Throwing Ropes over Rough Mountain Passes, in the Dark." *Annu. Rev. Phys. Chem.* **53**.1 (2002), pp. 291–318. DOI: 10.1146/annurev.physchem.53.082301.113146.

[26] Born, M. and Fock, V. "Beweis des Adiabatensatzes." *Z. Phys.* **51**.3 (1928), pp. 165–180. DOI: 10.1007/BF01343193.

[27] Born, M. and Oppenheimer, R. "Zur Quantentheorie der Molekeln." *Ann. Phys. – Berlin* **389**.20 (1927), pp. 457–484. DOI: 10.1002/andp.19273892002.

[28] Bowman, J. M., Czakó, G. and Fu, B. "High-Dimensional Ab Initio Potential Energy Surfaces for Reaction Dynamics Calculations." *Phys. Chem. Chem. Phys.* **13**.18 (2011), pp. 8094–8111. DOI: 10.1039/C0CP02722G.

[29] Braams, B. J. and Bowman, J. M. "Permutationally Invariant Potential Energy Surfaces in High Dimensionality." *Int. Rev. Phys. Chem.* **28**.4 (2009), pp. 577–606. DOI: 10.1080/01442350903234923.

[30] Bracker, A. S., North, S. W., Suits, A. G. and Lee, Y. T. "The Near Ultraviolet Dissociation Dynamics of Azomethane: Correlated V-T Energy Disposal and Product Appearance Times." *J. Chem. Phys.* **109**.17 (1998), pp. 7238–7245. DOI: 10.1063/1.477402.

[31] Brédas, J.-L., Norton, J. E., Cornil, J. and Coropceanu, V. "Molecular Understanding of Organic Solar Cells: The Challenges." *Acc. Chem. Res.* **42**.11 (2009), pp. 1691–1699. DOI: 10.1021/ar900099h.

[32] Brillhart, J. and Selfridge, J. L. "Some Factorizations of $2^n \pm 1$ and Related Results." *Math. Comput.* **21** (1967), pp. 87–96.

[33] Brown, A., Braams, B. J., Christoffel, K., Jin, Z. and Bowman, J. M. "Classical and Quasiclassical Spectral Analysis of CH_5^+ Using an Ab Initio Potential Energy Surface." *J. Chem. Phys.* **119**.17 (2003), pp. 8790–8793. DOI: 10.1063/1.1622379.

[34] Bungartz, H.-J. and Griebel, M. "Sparse Grids." English. *Acta Numer.* **13** (2004), pp. 147–269.

[35] Burton, K. A. and Weisman, R. B. "Stepwise Photodissociation of Vapor-Phase Azomethane." *J. Am. Chem. Soc.* **112**.5 (1990), pp. 1804–1807. DOI: `10.1021/ja00161a024`.

[36] Cattaneo, P. and Persico, M. "Semiclassical Treatment of the Photofragmentation of Azomethane." *Chem. Phys. Lett.* **289** (1998), pp. 160–166. DOI: `https://doi.org/10.1016/S0009-2614(98)00402-3`.

[37] Cattaneo, P. and Persico, M. "Diabatic and Adiabatic Potential-Energy Surfaces for Azomethane Photochemistry." *Theor. Chem. Acc.* **103** (2000), pp. 390–398.

[38] Cattaneo, P. and Persico, M. "Semiclassical Simulations of Azomethane Photochemistry in the Gas Phase and in Solution." *J. Am. Chem. Soc.* **123**.31 (2001), pp. 7638–7645. DOI: `10.1021/ja0102843`.

[39] Chen, Q. and Bowman, J. M. "Quantum and Classical IR Spectra of $(HCOOH)_2$, $(DCOOH)_2$ and $(DCOOD)_2$ Using Ab Initio Potential Energy and Dipole Moment Surfaces." *Faraday Discuss.* (2018), pp. 33–49. DOI: `10.1039/C8FD00077H`.

[40] Chkifa, A., Cohen, A. and Schwab, C. "High-Dimensional Adaptive Sparse Polynomial Interpolation and Applications to Parametric PDEs." *Found. Comput. Math.* **14**.4 (2014), pp. 601–633. DOI: `10.1007/s10208-013-9154-z`.

[41] Clenshaw, C. W. and Curtis, A. R. "A Method for Numerical Integration on an Automatic Computer." *Numer. Math.* **2**.1 (1960), pp. 197–205. DOI: `10.1007/BF01386223`.

[42] Collins, M. A. "Molecular Potential-Energy Surfaces for Chemical Reaction Dynamics." *Theor. Chem. Acc.* **108**.6 (2002), pp. 313–324. DOI: `10.1007/s00214-002-0383-5`.

[43] Collins, M. A. "Molecular Potential Energy Surfaces by Interpolation." *Lec. Notes Comput. Sc.* (2003), pp. 159–167.

[44] Collins, M. A. and Parsons, D. F. "Implications of Rotation–Inversion–Permutation Invariance for Analytic Molecular Potential Energy Surfaces." *J. Chem. Phys.* **99**.9 (1993), pp. 6756–6772. DOI: `10.1063/1.465819`.

[45] Consortini, A. and Frieden, B. "Quantum-Mechanical Solution for the Simple Harmonic Oscillator in a Box." *Nuov. Cim. B* **35** (1976), 153–164. DOI: `10.1007/BF02724052`.

[46] Craig, C. F., Duncan, W. R. and Prezhdo, O. V. "Trajectory Surface Hopping in the Time-Dependent Kohn-Sham Approach for Electron-Nuclear Dynamics." *Phys. Rev. Lett.* **95** (16 2005), p. 163001. DOI: `10.1103/PhysRevLett.95.163001`.

[47] Crank, J. and Nicolson, P. "A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type." *Math. Proc. Camb. Philos. Soc.* **43**.1 (1947), 50–67. DOI: `10.1017/S0305004100023197`.

[48] Cui, J. and Krems, R. V. “Gaussian Process Model for Collision Dynamics of Complex Molecules.” *Phys. Rev. Lett.* **115**.7 (2015), p. 073202. DOI: 10.1103/PhysRevLett.115.073202.

[49] Daniel, C., Full, J., Gonzalez, L., Lupulescu, C., Manz, J., Merli, A., Vajda, S. and Wöste, L. “Deciphering the Reaction Dynamics Underlying Optimal Control Laser Fields.” *Science* **299**.5606 (2003), pp. 536–539.

[50] Dawes, R. and Quintas-Sánchez, E. “The Construction of Ab Initio-Based Potential Energy Surfaces.” *Rev. Comp. Ch.* John Wiley & Sons, Ltd, 2018. Chap. 5, pp. 199–263. DOI: 10.1002/9781119518068.ch5.

[51] Delvos, F.-J. and Schempp, W. *Boolean Methods in Interpolation and Approximation*. Pitman Research Notes in Mathematics Series. Longman Scientific and Technical, Harlow, 1989.

[52] Deriglazov, A. *Classical Mechanics*. Springer-Verlag, Berlin, 2016.

[53] Diau, E. W.-G., Abou-Zied, O. K., Scala, A. A. and Zewail, A. H. “Femtosecond Dynamics of Transition States and the Concept of Concertedness: Nitrogen Extrusion of Azomethane Reactions.” *J. Am. Chem. Soc.* **120**.13 (1998), pp. 3245–3246. DOI: 10.1021/ja9743553.

[54] Diau, E. W.-G., Herek, J. L., Kim, Z. H. and Zewail, A. H. “Femsosecond Activation of Reactions and the Concept of Nonergodic Molecules.” *Science* **279**.5352 (1998), pp. 847–851.

[55] Diau, E. W.-G. and Zewail, A. H. “Femtochemistry of trans-Azomethane: A Combined Experimental and Theoretical Study.” *ChemPhysChem* **4**.5 (2003), pp. 445–456. DOI: 10.1002/cphc.200200579.

[56] Dick, J., Kuo, F. Y. and Sloan, I. H. “High-Dimensional Integration: The Quasi-Monte Carlo Way.” *Acta Numer.* **22** (2013), pp. 133–288.

[57] Dill, K. A., Phillips, A. T. and Rosen, J. B. “Protein Structure Prediction and Potential Energy Landscape Analysis Using Continuous Global Minimization.” *P. Comput. Mol. Biol.* (1997), pp. 109–117.

[58] Ditchfield, R., Hehre, W. J. and Pople, J. A. “Self-Consistent Molecular-Orbital Methods. IX. An Extended Gaussian-Type Basis for Molecular-Orbital Studies of Organic Molecules.” *J. Chem. Phys.* **54**.2 (1971), pp. 724–728. DOI: 10.1063/1.1674902.

[59] Dobbs, K. D. and Hehre, W. J. “Molecular Orbital Theory of the Properties of Inorganic and Organometallic Compounds. 4. Extended Basis Sets for Third-Row and Fourth-Row, Main-Group Elements.” *J. Comput. Chem.* **7**.3 (1986), pp. 359–378. DOI: 10.1002/jcc.540070313.

[60] Dobbs, K. D. and Hehre, W. J. “Molecular Orbital Theory of the Properties of Inorganic and Organometallic Compounds. 5. Extended Basis Sets for First-Row Transition Metals.” *J. Comput. Chem.* **8**.6 (1987), pp. 861–879. DOI: 10.1002/jcc.540080614.

[61] Dobbs, K. D. and Hehre, W. J. “Molecular Orbital Theory of the Properties of Inorganic and Organometallic Compounds. 6. Extended Basis Sets for Second-Row Transition Metals.” *J. Comput. Chem.* **8**.6 (1987), pp. 880–893. DOI: 10.1002/jcc.540080615.

[62] Duncan, W. R., Stier, W. M. and Prezhdo, O. V. “Ab Initio Nonadiabatic Molecular Dynamics of the Ultrafast Electron Injection across the Alizarin-TiO_2 Interface.” *J. Am. Chem. Soc.* **127**.21 (2005), pp. 7941–7951. DOI: 10.1021/ja042156v.

[63] Dykyj, J., Svoboda, J., Wilhoit, R. C., Frenkel, M. and Hall, K. R. “Organic Compounds, C_0 to C_{84}.” *Vapor Pressure and Antoine Constants for Nitrogen Containing Organic Compounds.* Ed. by Hall, K. R. Vol. 20C. 2001. Chap. 2, p. 28. DOI: 10.1007/10688591_3.

[64] Dzjadyk, V. K. and Ivanov, V. V. “On Asymptotics and Estimates for the Uniform Norms of the Lagrange Interpolation Polynomials Corresponding to the Chebyshev Nodal Points.” *Anal. Math.* **9**.2 (1983), pp. 85–97. DOI: 10.1007/BF01982005.

[65] Ehlich, H. and Zeller, K. “Auswertung der Normen von Interpolationsoperatoren.” *Math. Ann.* **164** (1966), pp. 105–112.

[66] Ehrenfest, P. “Bemerkung über die Angenäherte Gültigkeit der Klassischen Mechanik Innerhalb der Quantenmechanik.” *Z. Phys.* **45** (1927), pp. 455–457. DOI: 10.1007/BF01329203.

[67] Engel, P. S. “Mechanism of the Thermal and Photochemical Decomposition of Azoalkanes.” *Chem. Rev.* **80**.2 (1980), pp. 99–150. DOI: 10.1021/cr60324a001.

[68] Espinosa-Garcia, J., Monge-Palacios, M. and Corchado, J. C. “Constructing Potential Energy Surfaces for Polyatomic Systems: Recent Progress and New Problems.” *Adv. Phys. Chem.* **2012** (2012), p. 164752. DOI: 10.1155/2012/164752.

[69] Estiú, G., Rama, J., Pereira, A., Cachau, R. E. and Ventura, O. N. “A Theoretical Study of Excited State Proton Transfer in 3-hydroxychromone and Related Molecules.” *J. Mol. Struc. Theochem* **487**.3 (1999), pp. 221–230. DOI: https://doi.org/10.1016/S0166-1280(98)00602-2.

[70] Faber, N. M. “Estimating the Uncertainty in Estimates of Root Mean Square Error of Prediction: Application to Determining the Size of an Adequate Test Set in Multivariate Calibration.” *Chemometr. Intell. Lab.* **49**.1 (1999), pp. 79–89. DOI: 10.1016/S0169-7439(99)00027-1.

[71] Farwig, R. “Multivariate Interpolation of Arbitrarily Spaced Data by Moving Least Squares Methods.” *J. Comput. Appl. Math.* **16**.1 (1986), pp. 79–93. DOI: 10.1016/0377-0427(86)90175-5.

[72] Fazzi, D., Barbatti, M. and Thiel, W. “Modeling Ultrafast Exciton Deactivation in Oligothiophenes via Nonadiabatic Dynamics.” *Phys. Chem. Chem. Phys.* **17**.12 (2015), pp. 7787–7799. DOI: 10.1039/C5CP00019J.

[73] Frisch, M. J., Trucks, G. W., Schlegel, H. B., Scuseria, G. E., Robb, M. A., Cheeseman, J. R., Scalmani, G., Barone, V., Petersson, G. A., Nakatsuji, H., Li, X., Caricato, M., Marenich, A. V., Bloino, J., Janesko, B. G., Gomperts, R., Mennucci, B., Hratchian, H. P., Ortiz, J. V., Izmaylov, A. F., Sonnenberg, J. L., Williams-Young, D., Ding, F., Lipparini, F., Egidi, F., Goings, J., Peng, B., Petrone, A., Henderson, T., Ranasinghe, D., Zakrzewski, V. G., Gao, J., Rega, N., Zheng, G., Liang, W., Hada, M., Ehara, M., Toyota, K., Fukuda, R., Hasegawa, J., Ishida, M., Nakajima, T., Honda, Y., Kitao, O., Nakai, H., Vreven, T., Throssell, K., Montgomery Jr., J. A., Peralta, J. E., Ogliaro, F., Bearpark, M. J., Heyd, J. J., Brothers, E. N., Kudin, K. N., Staroverov, V. N., Keith, T. A., Kobayashi, R., Normand, J., Raghavachari, K., Rendell, A. P., Burant, J. C., Iyengar, S. S., Tomasi, J., Cossi, M., Millam, J. M., Klene, M., Adamo, C., Cammi, R., Ochterski, J. W., Martin, R. L., Morokuma, K., Farkas, O., Foresman, J. B. and Fox, D. J. *Gaussian 16 Revision A.03.* 2016.

[74] Galván, I. F., Delcey, M. G., Pedersen, T. B., Aquilante, F. and Lindh, R. “Analytical State-Average Complete-Active-Space Self-Consistent Field Nonadiabatic Coupling Vectors: Implementation with Density-Fitted Two-Electron Integrals and Application to Conical Intersections.” *J. Chem. Theory Comput.* **12**.8 (2016), pp. 3636–3653. DOI: 10.1021/acs.jctc.6b00384.

[75] Gantner, R. N., Herrmann, L. and Schwab, C. “Quasi–Monte Carlo Integration for Affine-Parametric, Elliptic PDEs: Local Supports and Product Weights.” *SIAM J. Numer. Anal.* **56**.1 (2018), pp. 111–135. DOI: 10.1137/16M1082597.

[76] Gautschi, W. *Numerical Analysis.* Birkhäuser Basel, 2011.

[77] Glasstone, S., Eyring, H. and Laidler, K. J. *The Theory of Rate Processes.* McGraw-Hill, New York, 1941.

[78] González-Luque, R., Olaso-González, G., Merchán, M., Coto, P. B., Serrano-Andrés, L. and Garavelli, M. “On the Role of the Triplet State in the cis/trans Isomerization of Rhodopsin: A CASPT2//CASSCF Study of a Model Chromophore.” *Int. J. Quantum Chem.* **111**.13 (2011), pp. 3431–3437. DOI: 10.1002/qua.23079.

[79] Gordon, M. S., Binkley, J. S., Pople, J. A., Pietro, W. J. and Hehre, W. J. “Self-Consistent Molecular-Orbital Methods. 22. Small Split-Valence Basis Sets for Second-Row Elements.” *J. Am. Chem. Soc.* **104**.10 (1982), pp. 2797–2803. DOI: 10.1021/ja00374a017.

[80] Grafakos, L. *Classical Fourier Analysis*. Springer, 2014.

[81] Griebel, M. and Hamaekers, J. "Fast Discrete Fourier Transform on Generalized Sparse Grids." *Sparse Grids and Applications – Munich 2012*. Ed. by Garcke, J. and Pflüger, D. Lecture Notes in Computational Science and Engineering. Springer International Publishing Switzerland, 2014. Chap. 4, pp. 75–107.

[82] Gunzburger, M. D., Webster, C. G. and Zhang, G. "Stochastic finite element methods for partial differential equations with random input data." *Acta Numer.* **23** (2014), pp. 521–650. DOI: 10.1017/S0962492914000075.

[83] Guo, Y., Tokmakov, I., Thompson, D. L., Wagner, A. F. and Minkoff, M. "Interpolating Moving Least-Squares Methods for Fitting Potential Energy Surfaces: Improving Efficiency via Local Approximants." *J. Chem. Phys.* **127**.21 (2007), p. 214106. DOI: 10.1063/1.2805084.

[84] Hallatschek, K. "Fouriertransformation auf dünnen Gittern mit hierarchischen Basen." *Numer. Math.* **63**.1 (1992), pp. 83–97.

[85] Hammes-Schiffer, S. and Tully, J. C. "Proton Transfer in Solution: Molecular Dynamics with Quantum Transitions." *J. Chem. Phys.* **101**.6 (1994), pp. 4657–4667. DOI: 10.1063/1.467455.

[86] Hare, S. R., Bratholm, L. A., Glowacki, D. R. and Carpenter, B. K. "Low Dimensional Representations along Intrinsic Reaction Coordinates and Molecular Dynamics Trajectories Using Interatomic Distance Matrices." *Chem. Sci.* **10** (43 2019), pp. 9954–9968. DOI: 10.1039/C9SC02742D.

[87] Hart, J., Alexanderian, A. and Gremaud, P. "Efficient Computation of Sobol' Indices for Stochastic Models." *SIAM J. Sci. Comput.* **39**.4 (2017), A1514–A1530. DOI: 10.1137/16M106193X.

[88] He, Z. "On the Error Rate of Conditional Quasi–Monte Carlo for Discontinuous Functions." *SIAM J. Numer. Anal.* **57**.2 (2019), pp. 854–874. DOI: 10.1137/18M118270X.

[89] Hehre, W. J., Ditchfield, R. and Pople, J. A. "Self–Consistent Molecular Orbital Methods. XII. Further Extensions of Gaussian–Type Basis Sets for Use in Molecular Orbital Studies of Organic Molecules." *J. Chem. Phys.* **56**.5 (1972), pp. 2257–2261. DOI: 10.1063/1.1677527.

[90] Helgaker, T., Uggerud, E. and Jensen, H. J. A. "Integration of the Classical Equations of Motion on Ab Initio Molecular Potential Energy Surfaces Using Gradients and Hessians: Application to Translational Energy Release upon Fragmentation." *Chem. Phys. Lett.* **173**.2 (1990), pp. 145–150. DOI: https://doi.org/10.1016/0009-2614(90)80068-O.

[91] Helmberg, G. “The Gibbs Phenomenon for Fourier Interpolation.” *J. Approx. Theory* **78**.1 (1994), pp. 41–63.

[92] Hohenberg, P. and Kohn, W. “Inhomogeneous Electron Gas.” *Phys. Rev.* **136**.3B (1964), B864–B871.

[93] Houk, K. N., Li, Y. and Evanseck, J. D. “Transition Structures of Hydrocarbon Pericyclic Reactions.” *Angew. Chem. Int. Edit.* **31**.6 (1992), pp. 682–708. DOI: `10.1002/anie.199206821`.

[94] Jackson, D. “On Approximation by Trigonometric Sums and Polynomials.” *Trans. Am. Math. Soc.* **13**.4 (1912), pp. 491–515.

[95] Jackson, D. “On the Accuracy of Trigonometric Interpolation.” *Trans. Am. Math. Soc.* **14**.4 (1913), pp. 453–461.

[96] Jakeman, J. D., Archibald, R. and Xiu, D. “Characterization of Discontinuities in High-Dimensional Stochastic Problems on Adaptive Sparse Grids.” *J. Comput. Phys.* **230**.10 (2011), pp. 3977–3997. DOI: `10.1016/j.jcp.2011.02.022`.

[97] Jakeman, J. D., Narayan, A. and Xiu, D. “Minimal Multi-Element Stochastic Collocation for Uncertainty Quantification of Discontinuous Functions.” *J. Comput. Phys.* **242** (2013), pp. 790–808. DOI: `10.1016/j.jcp.2013.02.035`.

[98] Jakeman, J. D. and Roberts, S. G. “Local and dimension adaptive stochastic collocation for uncertainty quantification.” *Sparse Grids and Applications*. Springer, 2012, pp. 181–203. DOI: `10.1007/978-3-642-31703-3_9`.

[99] Jiang, B., Li, J. and Guo, H. “Potential Energy Surfaces from High Fidelity Fitting of Ab Initio Points: The Permutation Invariant Polynomial–Neural Network Approach.” *Int. Rev. Phys. Chem.* **35**.3 (2016), pp. 479–506. DOI: `10.1080/0144235X.2016.1200347`.

[100] Judd, K. L., Maliar, L., Maliar, S. and Valero, R. “Smolyak Method for Solving Dynamic Economic Models: Lagrange Interpolation, Anisotropic Grid and Adaptive Domain.” *J. Econ. Dyn. Control* **44** (2014), pp. 92 –123. DOI: `https://doi.org/10.1016/j.jedc.2014.03.003`.

[101] Kapral, R. “Progress in the Theory of Mixed Quantum-Classical Dynamics.” *Annu. Rev. of Phys. Chem.* **57**.1 (2006), pp. 129–157. DOI: `10.1146/annurev.physchem.57.032905.104702`.

[102] Katznelson, Y. *An Introduction to Harmonic Analysis*. 3rd ed. Cambridge University Press, 2004.

[103] Kaupp, M., Schleyer, P. v. R., Stoll, H. and Preuss, H. “Pseudopotential Approaches to Ca, Sr, and Ba Hydrides. Why Are Some Alkaline Earth MX_2 Compounds Bent?” *J.*

Chem. Phys. **94**.2 (1991), pp. 1360–1366. DOI: doi:http://dx.doi.org/10.1063/1.459993.

[104] Kelley, C. T. *Iterative Methods for Linear and Nonlinear Equations.* SIAM, 1995. DOI: 10.1137/1.9781611970944.

[105] Kelley, C. T. "Numerical Methods for Nonlinear Equations." *Acta Numer.* **27** (2018), 207–287. DOI: 10.1017/S0962492917000113.

[106] Khakhutskyy, V. and Hegland, M. "Spatially-Dimension-Adaptive Sparse Grids for Online Learning." *Sparse Grids and Applications – Stuttgart 2014.* Springer, 2016, pp. 133–162. DOI: 10.1007/978-3-319-28262-6_6.

[107] Klimke, A. and Wohlmuth, B. "Algorithm 847: spinterp: Piecewise Multilinear Hierarchical Sparse Grid Interpolation in MATLAB." *ACM Trans. Math. Software* **31**.4 (2005), pp. 561–579. DOI: 10.1145/1114268.1114275.

[108] Kohn, W. and Sham, L. J. "Self-Consistent Equations Including Exchange and Correlation Effects." *Phys. Rev.* **140** (4A 1965), A1133–A1138.

[109] Komatsuzaki, T., Hoshino, K., Matsunaga, Y., Rylance, G. J., Johnston, R. L. and Wales, D. J. "How Many Dimensions Are Required to Approximate the Potential Energy Landscape of a Model Protein?" *J. Chem. Phys.* **122**.8 (2005), p. 084714. DOI: 10.1063/1.1854123.

[110] Korobov, N. M. *Exponential Sums and Their Applications.* Kluwer Academic Publishers, 1992.

[111] Kreyszig, E. *Introductory Functional Analysis with Applications.* John Wiley and Sons, 1978.

[112] Krishnan, R., Binkley, J. S., Seeger, R. and Pople, J. A. "Self-Consistent Molecular Orbital Methods. XX. A Basis Set for Correlated Wave Functions." *J. Chem. Phys.* **72**.1 (1980), pp. 650–654.

[113] Kritzer, P., Pillichshammer, F. and Woźniakowski, H. "Multivariate Integration of Infinitely Many Times Differentiable Functions in Weighted Korobov Spaces." *Math. Comp.* **83** (2014), pp. 1189–1206. DOI: 10.1090/S0025-5718-2013-02739-1.

[114] Kukovec, B.-M., Kodrin, I., Mihalić, Z., Furić, K. and Popović, Z. "Cis–trans Isomerism in Cobalt(II) Complexes with 3-hydroxypicolinic Acid. Structural, DFT and Thermal Studies." *Inorg. Chim. Acta* **363**.8 (2010), pp. 1887–1896. DOI: https://doi.org/10.1016/j.ica.2010.02.010.

[115] Lanczos, C. *The Variational Principles of Mechanics.* Courier Corporation, 2012.

[116] Landau, L. D. *Z. Phys. Sowjetunion* **1** (1932), pp. 88–98.

[117] Lee, C., Yang, W. and Parr, R. G. “Development of the Colle-Salvetti Correlation-Energy Formula into a Functional of the Electron Density.” *Phys. Rev. B* **37**.2 (1988), pp. 785–789.

[118] Leimkuhler, B. and Matthews, C. “Rational Construction of Stochastic Numerical Methods for Molecular Sampling.” *Appl. Math. Res. Express* **2013**.1 (2013), pp. 34–56.

[119] Leimkuhler, B. and Matthews, C. “Robust and Efficient Configurational Molecular Sampling via Langevin Dynamics.” *J. Chem. Phys.* **138**.17 (2013), p. 174102. DOI: 10.1063/1.4802990.

[120] Levine, I. N. *Quantum Chemistry*. 7th ed. Pearson, 2014.

[121] Li, W. and Ma, A. “Reaction Mechanism and Reaction Coordinates from the Viewpoint of Energy Flow.” *J. Chem. Phys.* **144**.11 (2016), p. 114103. DOI: 10.1063/1.4943581.

[122] Li, X. and Frisch, M. J. “Energy-Represented Direct Inversion in the Iterative Subspace within a Hybrid Geometry Optimization Method.” *J. Chem. Theory Comput.* **2**.3 (2006), pp. 835–839. DOI: 10.1021/ct050275a.

[123] Lin, Q., Zhang, L., Zhang, Y. and Jiang, B. “Searching Configurations in Uncertainty Space: Active Learning of High-Dimensional Neural Network Reactive Potentials.” *J. Chem. Theory Comput.* **17**.5 (2021), pp. 2691–2701. DOI: 10.1021/acs.jctc.1c00166.

[124] Lingerfelt, D. B., Williams-Young, D. B., Petrone, A. and Li, X. “Direct Ab Initio (Meta) Surface-Hopping Dynamics.” *J. Chem. Theory Comput.* **12**.3 (2016), pp. 935–945. DOI: 10.1021/acs.jctc.5b00697.

[125] Liu, C., Kelley, C. T. and Jakubikova, E. “Molecular Dynamics Simulations on Relaxed Reduced-Dimensional Potential Energy Surfaces.” *J. Phys. Chem. A* (2019), pp. 4543–4554. DOI: 10.1021/acs.jpca.9b02298.

[126] Liu, R. S. and Asato, A. E. “The Primary Process of Vision and the Structure of Bathorhodopsin: A Mechanism for Photoisomerization of Polyenes.” *P. Natl. Acad. Sci. USA* **82**.2 (1985), pp. 259–263. DOI: 10.1073/pnas.82.2.259.

[127] Liu, R. S. H. and Hammond, G. S. “Examples of Hula-Twist in Photochemical cis–trans Isomerization.” *Chem. – Eur. J.* **7**.21 (2001), pp. 4536–4545. DOI: 10.1002/1521-3765(20011105)7:21<4536::AID-CHEM4536>3.0.CO;2-N.

[128] Liu, R., Cui, Q., Dunn, K. M. and Morokuma, K. “Ab Initio Molecular Orbital Study of the Mechanism of Photodissociation of Trans-Azomethane.” *J. Chem. Phys.* **105**.6 (1996), pp. 2333–2345. DOI: 10.1063/1.472101.

[129] Long, R., Prezhdo, O. V. and Fang, W. “Nonadiabatic Charge Dynamics in Novel Solar Cell Materials.” *WIREs Comput. Mol. Sci.* **7**.3 (2017), e1305. DOI: 10.1002/wcms.1305.

[130] Lorenz, S., Groß, A. and Scheffler, M. "Representing High-Dimensional Potential-Energy Surfaces for Reactions at Surfaces by Neural Networks." *Chem. Phys. Lett.* **395**.4 (2004), pp. 210–215. DOI: `https://doi.org/10.1016/j.cplett.2004.07.076`.

[131] Ma, X. and Zabaras, N. "An Adaptive Hierarchical Sparse Grid Collocation Algorithm for the Solution of Stochastic Differential Equations." *J. Comput. Phys.* **228**.8 (2009), pp. 3084–3113. DOI: `10.1016/j.jcp.2009.01.006`.

[132] Maisuradze, G. G., Kawano, A., Thompson, D. L., Wagner, A. F. and Minkoff, M. "Interpolating Moving Least-Squares Methods for Fitting Potential Energy Surfaces: Analysis of an Application to a Six-Dimensional System." *J. Chem. Phys.* **121**.21 (2004), pp. 10329–10338. DOI: `10.1063/1.1810477`.

[133] Maisuradze, G. G. and Thompson, D. L. "Interpolating Moving Least-Squares Methods for Fitting Potential Energy Surfaces: Illustrative Approaches and Applications." *J. Phys. Chem. A* **107**.37 (2003), pp. 7118–7124. DOI: `10.1021/jp030144a`.

[134] Maisuradze, G. G., Thompson, D. L., Wagner, A. F. and Minkoff, M. "Interpolating Moving Least-Squares Methods for Fitting Potential Energy Surfaces: Detailed Analysis of One-Dimensional Applications." *J. Chem. Phys.* **119**.19 (2003), pp. 10002–10014. DOI: `10.1063/1.1617271`.

[135] Manzhos, S., Dawes, R. and Carrington, T. "Neural Network-Based Approaches for Building High Dimensional and Quantum Dynamics-Friendly Potential Energy Surfaces." *Int. J. Quantum Chem.* **115**.16 (2014), pp. 1012–1020. DOI: `10.1002/qua.24795`.

[136] Martínez, T. J. "Insights for Light-Driven Molecular Devices from Ab Initio Multiple Spawning Excited-State Dynamics of Organic and Biological Chromophores." *Acc. Chem. Res.* **39**.2 (2006), pp. 119–126. DOI: `10.1021/ar040202q`.

[137] Martinez, T. J., Ben-Nun, M. and Levine, R. D. "Multi-Electronic-State Molecular Dynamics: A Wave Function Approach with Applications." *J. Phys. Chem.* **100**.19 (1996), pp. 7884–7895. DOI: `10.1021/jp953105a`.

[138] Minezawa, N. and Nakajima, T. "Trajectory Surface Hopping Molecular Dynamics Simulation by Spin-Flip Time-Dependent Density Functional Theory." *J. Chem. Phys.* **150**.20 (2019), p. 204120. DOI: `10.1063/1.5096217`.

[139] Morrow, Z., Kwon, H.-Y., Kelley, C. T. and Jakubikova, E. "Efficient Approximation of Potential Energy Surfaces with Mixed-Basis Interpolation." *Submitted* (2021), to appear.

[140] Morrow, Z. and Stoyanov, M. "A Method for Dimensionally Adaptive Sparse Trigonometric Interpolation of Periodic Functions." *SIAM J. Sci. Comput.* **42**.4 (2020), A2436–A2460. DOI: `10.1137/19M1283483`.

[141] Morrow, Z., Liu, C., Kelley, C. T. and Jakubikova, E. "Approximating Periodic Potential Energy Surfaces Using Sparse Trigonometric Interpolation." *J. Phys. Chem. B* **123**.45 (2019), pp. 9677–9684. DOI: `10.1021/acs.jpcb.9b08210`.

[142] Mukherjee, R. and Diwekar, U. M. "Comparison of Monte Carlo and Quasi-Monte Carlo Technique in Structure and Relaxing Dynamics of Polymer in Dilute Solution." *Comput. Chem. Eng.* **84** (2016), pp. 28–35. DOI: `10.1016/j.compchemeng.2015.08.014`.

[143] Münster, A. *Statistical Thermodynamics*. Springer-Verlag, Berlin, 1969.

[144] Nance, J. and Kelley, C. T. "A Sparse Interpolation Algorithm for Dynamical Simulations in Computational Chemistry." *SIAM J. Sci. Comput.* **37**.5 (2015), S137–S156.

[145] Nance, J., Bowman, D. N., Mukherjee, S., Kelley, C. T. and Jakubikova, E. "Insights into the Spin-State Transitions in $[Fe(tpy)_2]_2{}^+$: Importance of the Terpyridine Rocking Motion." *Inorg. Chem.* **54**.23 (2015), pp. 11259–11268. DOI: `10.1021/acs.inorgchem.5b01747`.

[146] Nance, J., Jakubikova, E. and Kelley, C. T. "Reaction Path Following with Sparse Interpolation." *J. Chem. Theory Comput.* **10**.8 (2014), pp. 2942–2949. DOI: `10.1021/ct5004669`.

[147] Nance, J. D. "Investigating Molecular Dynamics with Sparse Grid Surrogate Models." PhD thesis. North Carolina State University, 2015.

[148] Narayan, A. and Jakeman, J. D. "Adaptive Leja sparse grid constructions for stochastic collocation and high-dimensional approximation." *SIAM J. Sci. Comput.* **36**.6 (2014), A2952–A2983. DOI: `10.1137/140966368`.

[149] Neese, F. "The ORCA Program System." *WIREs Comput. Mol. Sci.* **2**.1 (2012), pp. 73–78.

[150] Neese, F. "Software Update: The ORCA Program System, Version 4.0." *WIREs Comput. Mol. Sci.* **8**.1 (2018), e1327.

[151] Niederreiter, H. *Random Number Generation and Quasi-Monte Carlo Methods*. SIAM, 1992.

[152] Nobile, F., Tamellini, L. and Tempone, R. "Convergence of quasi-optimal sparse-grid approximation of Hilbert-space-valued functions: application to random elliptic PDEs." *Numer. Math.* **134**.2 (2016), pp. 343–388. DOI: `10.1007/s00211-015-0773-y`.

[153] Nobile, F., Tempone, R. and Webster, C. G. "An Anisotropic Sparse Grid Stochastic Collocation Method for Partial Differential Equations with Random Input Data." *SIAM J. Numer. Anal.* **46**.5 (2008), pp. 2411–2442. DOI: `10.1137/070680540`.

[154] North, S. W., Longfellow, C. A. and Lee, Y. T. "The Near Ultraviolet Photodissociation Dynamics of Azomethane." *J. Chem. Phys.* **99**.6 (1993), pp. 4423–4429. DOI: 10.1063/1.466095.

[155] Novak, E. and Ritter, K. "High Dimensional Integration of Smooth Functions over Cubes." *Numer. Math.* **75**.1 (1996), pp. 79–97. DOI: 10.1007/s002110050231.

[156] Novak, E. and Ritter, K. "Simple Cubature Formulas with High Polynomial Exactness." *Constr. Approx.* **15**.4 (1999), pp. 499–522. DOI: 10.1007/s003659900119.

[157] Novak, E. and Woźniakowski, H. *Tractability of Multivariate Problems.* European Mathematical Society, 2008.

[158] Nystrom, N. A., Levine, M. J., Roskies, R. Z. and Scott, J. R. "Bridges: A Uniquely Flexible HPC Resource for New Communities and Data Analytics." *Proc. XSEDE Conf.* Association for Computing Machinery, 2015. DOI: 10.1145/2792745.2792775.

[159] Olaso-González, G., Roca-Sanjuán, D., Serrano-Andrés, L. and Merchán, M. "Toward the Understanding of DNA Fluorescence: The Singlet Excimer of Cytosine." *J. Chem. Phys.* **125**.23 (2006), p. 231102. DOI: 10.1063/1.2408411.

[160] Papageorgiou, A. and Woźniakowski, H. "Tractability through Increasing Smoothness." *J. Complexity* **26**.5 (2010), pp. 409–421. DOI: 10.1016/j.jco.2009.12.004.

[161] Parchaňský, V., Kapitán, J., Kaminský, J., Šebestík, J. and Bouř, P. "Ramachandran Plot for Alanine Dipeptide as Determined from Raman Optical Activity." *J. Phys. Chem. Lett.* **4**.16 (2013), pp. 2763–2768. DOI: 10.1021/jz401366j.

[162] Pauli, W. "Über den Zusammenhang des Abschlusses der Elektronengruppen im Atom mit der Komplexstruktur der Spektren." *Z. Angew. Phys.* **31** (1925), pp. 765–783. DOI: 10.1007/BF02980631.

[163] Peng, C., Ayala, P. Y., Schlegel, H. B. and Frisch, M. J. "Using Redundant Internal Coordinates to Optimize Equilibrium Geometries and Transition States." *J. Comput. Chem.* **17**.1 (1996), pp. 49–56. DOI: 10.1002/(SICI)1096-987X(19960115)17:1<49::AID-JCC5>3.0.CO;2-0.

[164] Petrenko, T. and Neese, F. "Analysis and Prediction of Absorption Band Shapes, Fluorescence Band Shapes, Resonance Raman Intensities, and Excitation Profiles Using the Time-Dependent Theory of Electronic Spectroscopy." *J. Chem. Phys.* **127**.16 (2007), p. 164319. DOI: 10.1063/1.2770706.

[165] Pflüger, D. "Spatially Adaptive Refinement." *Sparse Grids and Applications.* Springer, 2012, pp. 243–262. DOI: 10.1007/978-3-642-31703-3_12.

[166] Pflüger, D., Peherstorfer, B. and Bungartz, H.-J. "Spatially Adaptive Sparse Grids for High-Dimensional Data-Driven Problems." *J. Complexity* **26**.5 (2010), pp. 508–522. DOI: 10.1016/j.jco.2010.04.001.

[167] Pietro, W. J., Francl, M. M., Hehre, W. J., DeFrees, D. J., Pople, J. A. and Binkley, J. S. "Self-Consistent Molecular Orbital Methods. 24. Supplemented Small Split-Valence Basis Sets for Second-Row Elements." *J. Am. Chem. Soc.* **104**.19 (1982), pp. 5039–5048. DOI: 10.1021/ja00383a007.

[168] Pinkus, A. "Negative Theorems in Approximation Theory." *Amer. Math. Monthly* **110**.10 (2003), pp. 900–911. DOI: 10.1080/00029890.2003.11920030.

[169] Plasser, F., Barbatti, M., Aquino, A. J. A. and Lischka, H. "Electronically Excited States and Photodynamics: A Continuing Challenge." *Theor. Chem. Acc.* **131**.1 (2012), pp. 1–14.

[170] Polli, D., Altoè, P., Weingart, O., Spillane, K. M., Manzoni, C., Brida, D., Tomasello, G., Orlandi, G., Kukura, P., Mathies, R. A., Garavelli, M. and Cerullo, G. "Conical Intersection Dynamics of the Primary Photoisomerization Event in Vision." *Nature* **467**.7314 (2010), pp. 440–443.

[171] Pöplau, G. and Sprengel, F. "Some Error Estimates for Periodic Interpolation on Full and Sparse Grids." *Curves and Surfaces with Applications in CAGD*. Vanderbilt University Press, 1997, pp. 355–362.

[172] Pople, J. A. "Nobel Lecture: Quantum chemical models." *Rev. Mod. Phys.* **71** (5 1999), pp. 1267–1274. DOI: 10.1103/RevModPhys.71.1267.

[173] Powell, M. J. D. "An Efficient Method for Finding the Minimum of a Function of Several Variables without Calculating Derivatives." *Comput. J.* **7**.2 (1964), pp. 155–162. DOI: 10.1093/comjnl/7.2.155.

[174] Preston, R. K. and Tully, J. C. "Effects of Surface Crossing in Chemical Reactions: The H_3^+ System." *J. Chem. Phys.* **54**.10 (1971), pp. 4297–4304. DOI: 10.1063/1.1674676.

[175] Prezhdo, O. V. and Kisil, V. V. "Mixing Quantum and Classical Mechanics." *Phys. Rev. A* **56** (1 1997), pp. 162–175. DOI: 10.1103/PhysRevA.56.162.

[176] Prezhdo, O. V. and Rossky, P. J. "Mean-Field Molecular Dynamics with Surface Hopping." *J. Chem. Phys.* **107**.3 (1997), pp. 825–834. DOI: 10.1063/1.474382.

[177] Purcell, K. F. "Pseudorotational Intersystem Crossing in d^6 Complexes." *J. Am. Chem. Soc.* **101**.18 (1979), pp. 5147–5152. DOI: 10.1021/ja00512a005.

[178] Qu, C., Yu, Q. and Bowman, J. M. "Permutationally Invariant Potential Energy Surfaces." *Annu. Rev. Phys. Chem.* **69**.1 (2018), pp. 151–175. DOI: 10.1146/annurev-physchem-050317-021139.

[179] Ramsperger, H. C. "The Thermal and Photochemical Decomposition of Azo Compounds and the Problem of Reaction Rates." *Proc. Nat. Acad. Sci. U.S.A.* **13**.12 (1927), pp. 849–853. DOI: 10.1073/pnas.13.12.849.

[180] Ramsperger, H. C. "The Decomposition of Azomethane. A Homogeneous, Unimolecular Reaction." *J. Am. Chem. Soc.* **49**.4 (1927), pp. 912–916. DOI: 10.1021/ja01403a003.

[181] Richard, R. M., Lao, K. U. and Herbert, J. M. "Understanding the Many-Body Expansion for Large Systems. I. Precision Considerations." *J. Chem. Phys.* **141**.1 (2014), p. 014108. DOI: 10.1063/1.4885846.

[182] Rivlin, T. J. "The Lebesgue Constants for Polynomial Interpolation." *Functional Analysis and Its Applications*. Ed. by Garnir, H. G., Unni, K. R. and Williamson, J. H. Berlin, Heidelberg: Springer Berlin Heidelberg, 1974, pp. 422–437.

[183] Ruckenbauer, M., Barbatti, M., Sellner, B., Muller, T. and Lischka, H. "Azomethane: Nonadiabatic Photodynamical Simulations in Solution." *J. Phys. Chem. A* **114**.48 (2010), pp. 12585–12590. DOI: 10.1021/jp108844g.

[184] Schmitz, G., Klinting, E. L. and Christiansen, O. "A Gaussian Process Regression Adaptive Density Guided Approach for Potential Energy Surface Construction." *J. Chem. Phys.* **153**.6 (2020), p. 064105. DOI: 10.1063/5.0015344.

[185] Sellner, B., Ruckenbauer, M., Stambolić, I., Barbatti, M., Aquino, A. J. A. and Lischka, H. "Photodynamics of Azomethane: A Nonadiabatic Surface-Hopping Study." *J. Phys. Chem. A* **114**.33 (2010), pp. 8778–8785. DOI: 10.1021/jp101745t.

[186] Send, R. and Furche, F. "First-Order Nonadiabatic Couplings from Time-Dependent Hybrid Density Functional Response Theory: Consistent Formalism, Implementation, and Performance." *J. Chem. Phys.* **132**.4 (2010), p. 044107. DOI: 10.1063/1.3292571.

[187] Shang, Y., Li, Q., Meng, L., Wang, D. and Shuai, Z. "Computational Characterization of Organic Photovoltaic Devices." *Theor. Chem. Acc.* **129**.3 (2011), pp. 291–301. DOI: 10.1007/s00214-011-0924-x.

[188] Shen, J. and Wang, L.-L. "Sparse Spectral Approximations of High-Dimensional Problems Based on Hyperbolic Cross." *SIAM J. Numer. Anal.* **48**.3 (2010), pp. 1087–1109.

[189] Shen, J. and Yu, H. "Efficient Spectral Sparse Grid Methods and Applications to High-Dimensional Elliptic Problems." English. *SIAM J. Sci. Comput.* **32**.6 (2010), pp. 3228–3250.

[190] Sickel, W. and Ullrich, T. "Tensor Products of Sobolev–Besov Spaces and Applications to Approximation from the Hyperbolic Cross." *J. Approx. Theory* **161** (2009), pp. 748–786. DOI: 10.1016/j.jat.2009.01.001.

[191] Slater, J. C. “The Theory of Complex Spectra.” *Phys. Rev.* **34** (10 1929), pp. 1293–1322. DOI: 10.1103/PhysRev.34.1293.

[192] Smolyak, S. A. “Quadrature and Interpolation Formulas for Tensor Products of Certain Classes of Functions.” *Dokl. Akad. Nauk SSSR* **148**.5 (1963), pp. 1042–1045.

[193] Sobol’, I. M. “On the Distribution of Points in a Cube and the Approximate Evaluation of Integrals.” *USSR Comput. Math. Math. Phys.* **7**.4 (1967), pp. 86–112. DOI: 10.1016/0041-5553(67)90144-9.

[194] Sousa, C., Graaf, C. de, Rudavskyi, A., Broer, R., Tatchen, J., Etinski, M. and Marian, C. M. “Ultrafast Deactivation Mechanism of the Excited Singlet in the Light-Induced Spin Crossover of $[Fe(2,2'\text{-bipyridine})_3]_2{}^+$.” *Chem. – Eur. J.* **19**.51 (2013), pp. 17541–51. DOI: 10.1002/chem.201302992.

[195] Sprengel, F. “Periodic Interpolation and Wavelets on Sparse Grids.” *Numer. Algorithms* **17**.1 (1998), pp. 147–169. DOI: 10.1023/A:1012041629709.

[196] Sprengel, F. “A Class of Periodic Function Spaces and Interpolation on Sparse Grids.” *Numer. Func. Anal. Opt.* **21**.1-2 (2000), pp. 273–293.

[197] Stenrup, M. and Larson, A. “A Computational Study of Radiationless Deactivation Mechanisms of Furan.” *Chem. Phys.* **379**.1 (2011), pp. 6–12. DOI: 10.1016/j.chemphys.2010.10.002.

[198] Stephens, P. J., Devlin, F. J., Chabalowski, C. F. and Frisch, M. J. “Ab Initio Calculation of Vibrational Absorption and Circular Dichroism Spectra Using Density Functional Force Fields.” *J. Phys. Chem.* **98**.45 (1994), pp. 11623–11627. DOI: 10.1021/j100096a001.

[199] Störmer, C. “Sur les trajectoires des corpuscules électrisés dans l’espace sous l’action du magnétisme terrestre, avec application aux aurores boréales.” *Arch. Sci. Phys. Nat.* **33** (1912), pp. 51–69.

[200] Stoyanov, M. *TASMANIAN Sparse Grids (version 6.0)*. Oak Ridge National Laboratory, Oak Ridge, TN. 2019.

[201] Stoyanov, M. *Hierarchy-Direction Selective Approach for Locally Adaptive Sparse Grids*. Tech. rep. ORNL/TM-2013/384. One Bethel Valley Road, Oak Ridge, TN: Oak Ridge National Laboratory, 2013.

[202] Stoyanov, M. “Adaptive Sparse Grid Construction in a Context of Local Anisotropy and Multiple Hierarchical Parents.” *Sparse Grids and Applications-Miami 2016*. Springer, 2018, pp. 175–199.

[203] Stoyanov, M., Lebrun-Grandie, D., Burkardt, J. and Munster, D. *Tasmanian*. 2013. DOI: 10.11578/dc.20171025.on.1087.

[204] Stoyanov, M., Seleson, P. and Webster, C. "Predicting Fracture Patterns in Simulations of Brittle Materials under Variable Load and Material Strength." *19th AIAA Non-Deterministic Approaches Conference*. 2017, p. 1326. DOI: `10.2514/6.2017-1326`.

[205] Stoyanov, M. and Webster, C. G. "A Gradient-Based Sampling Approach for Dimension Reduction of Partial Differential Equations with Stochastic Coefficients." *Int. J. Uncertain. Quantif.* **5**.1 (2015), pp. 49–72. DOI: `10.1615/Int.J.UncertaintyQuantification.2014010945`.

[206] Stoyanov, M. K. and Webster, C. G. "A Dynamically Adaptive Sparse Grids Method for Quasi-Optimal Interpolation of Multidimensional Functions." *Comput. Math. Appl.* **71**.11 (2016), pp. 2449–2465. DOI: `https://doi.org/10.1016/j.camwa.2015.12.045`.

[207] Suchan, J., Janoš, J. and Slavíček, P. "Pragmatic Approach to Photodynamics: Mixed Landau–Zener Surface Hopping with Intersystem Crossing." *J. Chem. Theory Comput.* **16**.9 (2020), pp. 5809–5820. DOI: `10.1021/acs.jctc.0c00512`.

[208] Tavadze, P., Avendaño Franco, G., Ren, P., Wen, X., Li, Y. and Lewis, J. P. "A Machine-Driven Hunt for Global Reaction Coordinates of Azobenzene Photoisomerization." *J. Am. Chem. Soc.* **140**.1 (2018), pp. 285–290. DOI: `10.1021/jacs.7b10030`.

[209] Thompson, K. C., Jordan, M. J. T. and Collins, M. A. "Polyatomic Molecular Potential Energy Surfaces by Interpolation in Local Internal Coordinates." *J. Chem. Phys.* **108**.20 (1998), pp. 8302–8316. DOI: `10.1063/1.476259`.

[210] Todor, R. A. and Schwab, C. "Convergence Rates for Sparse Chaos Approximations of Elliptic Problems with Stochastic Coefficients." *IMA J. Numer. Anal.* **27**.2 (2007), pp. 232–261. DOI: `10.1093/imanum/drl025`.

[211] Tomov, S., Dongarra, J. and Baboulin, M. "Towards Dense Linear Algebra for Hybrid GPU-Accelerated Many-Core Systems." *Parallel Comput.* **36**.5-6 (2010), pp. 232–240. DOI: `10.1016/j.parco.2009.12.005`.

[212] Towns, J., Cockerill, T., Dahan, M., Foster, I., Gaither, K., Grimshaw, A., Hazlewood, V., Lathrop, S., Lifka, D., Peterson, G. D., Roskies, R., Scott, J. and Wilkins-Diehr, N. "XSEDE: Accelerating Scientific Discovery." *Comput. Sci. Eng.* **16**.05 (2014), pp. 62–74. DOI: `10.1109/MCSE.2014.80`.

[213] Tran, H., Webster, C. G. and Zhang, G. "Analysis of Quasi-Optimal Polynomial Approximations for Parameterized PDEs with Deterministic and Stochastic Coefficients." *Numer. Math.* **137**.2 (2017), pp. 451–493. DOI: `10.1007/s00211-017-0878-6`.

[214] Tully, J. C. "Molecular Dynamics with Electronic Transitions." *J. Chem. Phys.* **93**.2 (1990), pp. 1061–1071. DOI: `10.1063/1.459170`.

[215] Uggerud, E. and Helgaker, T. “Dynamics of the Reaction $CH_2OH^+ \rightarrow CHO^+ + H_2$. Translational Energy Release from Ab Initio Trajectory Calculations.” *J. Am. Chem. Soc.* **114**.11 (1992), pp. 4265–4268. DOI: 10.1021/ja00037a033.

[216] Uteva, E., Graham, R. S., Wilkinson, R. D. and Wheatley, R. J. “Interpolation of Intermolecular Potentials Using Gaussian Processes.” *J. Chem. Phys.* **147**.16 (2017), p. 161706. DOI: 10.1063/1.4986489.

[217] Valiev, M., Bylaska, E., Govind, N., Kowalski, K., Straatsma, T., Dam, H. V., Wang, D., Nieplocha, J., Apra, E., Windus, T. and Jong, W. de. “NWChem: A Comprehensive and Scalable Open-Source Solution for Large-Scale Molecular Simulations.” *Comput. Phys. Commun.* **181**.9 (2010), pp. 1477–1489.

[218] Verlet, L. “Computer ‘Experiments’ on Classical Fluids. I. Thermodynamical Properties of Lennard–Jones Molecules.” *Phys. Rev.* **159** (1 1967), pp. 98–103. DOI: 10.1103/PhysRev.159.98.

[219] Versiani Cabral, O., Téllez S, C. A., Giannerini, T. and Felcman, J. “Fourier-Transform Infrared Spectrum of Aspartate Hydroxo-aqua Nickel (II) Complex and DFT-B3LYP/3-21G and 6-311G Structural and Vibrational Calculations.” *Spectrochim. Acta A* **61**.1 (2005), pp. 337–345. DOI: https://doi.org/10.1016/j.saa.2004.02.037.

[220] Virshup, A. M., Punwong, C., Pogorelov, T. V., Lindquist, B. A., Ko, C. and Martínez, T. J. “Photodynamics in Complex Environments: Ab Initio Multiple Spawning Quantum Mechanical/Molecular Mechanical Dynamics.” *J. Phys. Chem. B* **113**.11 (2009), pp. 3280–3291. DOI: 10.1021/jp8073464.

[221] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, I., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P. and SciPy 1.0 Contributors. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python.” *Nat. Methods* **17** (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[222] Wang, L., Akimov, A. and Prezhdo, O. V. “Recent Progress in Surface Hopping: 2011–2015.” *J. Phys. Chem. Lett.* **7**.11 (2016), pp. 2100–2112. DOI: 10.1021/acs.jpclett.6b00710.

[223] Wasilkowski, G. and Wozniakowski, H. “Explicit Cost Bounds of Algorithms for Multivariate Tensor Product Problems.” *J. Complexity* **11**.1 (1995), pp. 1–56.

[224] Wiedbrauk, S., Maerz, B., Samoylova, E., Reiner, A., Trommer, F., Mayer, P., Zinth, W. and Dube, H. “Twisted Hemithioindigo Photoswitches: Solvent Polarity Determines the

Type of Light-Induced Rotations." *J. Am. Chem. Soc.* **138**.37 (2016), pp. 12219–12227. DOI: `10.1021/jacs.6b05981`.

[225] Wittig, C. "The Landau–Zener Formula." *J. Phys. Chem. B* **109**.17 (2005), pp. 8428–8430. DOI: `10.1021/jp040627u`.

[226] Xu, K. "The Chebyshev Points of the First Kind." *Appl. Numer. Math.* **102** (2016), pp. 17–30. DOI: `https://doi.org/10.1016/j.apnum.2015.12.002`.

[227] Yagi, K., Taketsugu, T. and Hirao, K. "Generation of Full-Dimensional Potential Energy Surface of Intramolecular Hydrogen Atom Transfer in Malonaldehyde and Tunneling Dynamics." *J. Chem. Phys.* **115**.23 (2001), pp. 10647–10655. DOI: `10.1063/1.1418436`.

[228] Yamabe, T., Akagi, K., Ohzeki, K., Fukui, K. and Shirakawa, H. "Isomerization Mechanisms from cis to trans Form in Polyacetylene." *J. Phys. Chem. Solids* **43**.7 (1982), pp. 577–581. DOI: `https://doi.org/10.1016/0022-3697(82)90047-6`.

[229] Zandler, M. E. and D'Souza, F. "The Remarkable Ability of B3LYP/3-21G(*) Calculations to Describe Geometry, Spectral and Electrochemical Properties of Molecular and Supramolecular Porphyrin–Fullerene Conjugates." *C. R. Chim.* **9**.7 (2006), pp. 960–981. DOI: `https://doi.org/10.1016/j.crci.2005.12.008`.

[230] Zener, C. "Non-Adiabatic Crossing of Energy Levels." *Proc. R. Soc. Lond.* **137**.833 (1932), pp. 696–702.

APPENDICES

APPENDIX

A

SUPPORTING INFORMATION

A.1 Tungsten

A.1.1 Electronic structure calculations

Each ground-state geometry is optimized in Gaussian 16 [73]. The computing environment is XSEDE Bridges [212]. Every node on Bridges has 128 GB RAM and two Intel Haswells with 14 cores each. Each Gaussian instance requests 14 cores and 16 GB RAM. The nodes and relative energies[1] for both interpolation bases are available in XLSX format.

The route section is as follows:

```
# opt=(Z-matrix,maxStep=5,MaxCycles=500,calcFC) b3lyp/gen nosymm
pseudo=cards scf=(vtl,xqc,maxconventionalcycles=512)
```

To find the stationary points and the corresponding energies in the “Results” section of the article, we used the following route section:

```
# opt=(Z-matrix,maxStep=5,MaxCycles=500,calcALL) freq
b3lyp/gen nosymm pseudo=cards scf=(vtl,xqc,maxconventionalcycles=512)
```

[1]In kcal/mol with the global-minimum energy at 0

For minima, we use no additional keywords. For saddle points, we use `opt=(noeigentest, TS)`; for maxima, we use `opt=(noeigentest,Saddle=2)`.

We define the geometry with the following Z-matrix (x_1 and x_2 explicitly labeled):

```
W
W    1          A2B
X    1          A3B    2          A3A
X    1          A4B    2          A4A    3          A4D
X    2          A5B    1          A5A    3          x1
X    2          A6B    1          A6A    5          A6D
C    3          A7B    1          A7A    2          A7D
C    7          A8B    3          A8A    1          A8D
C    8          A9B    7          A9A    3          A9D
C    9         A10B    8         A10A    7         A10D
C   10         A11B    9         A11A    8         A11D
H    7         A12B    8         A12A    9         A12D
H    8         A13B    9         A13A   10         A13D
H    9         A14B   10         A14A   11         A14D
H   10         A15B   11         A15A    7         A15D
H   11         A16B    7         A16A    8         A16D
C    5         A17B    2         A17A    1         x2
C   17         A18B    5         A18A    2         A18D
C   18         A19B   17         A19A    5         A19D
C   19         A20B   18         A20A   17         A20D
C   20         A21B   19         A21A   18         A21D
H   17         A22B   18         A22A   19         A22D
H   18         A23B   19         A23A   20         A23D
H   19         A24B   20         A24A   21         A24D
H   20         A25B   21         A25A   17         A25D
H   21         A26B   17         A26A   18         A26D
C    6         A27B    2         A27A    1         A27D
C   27         A28B    6         A28A    2         A28D
C   28         A29B   27         A29A    6         A29D
O   27         A30B   28         A30A   29         A30D
O   28         A31B   29         A31A   27         A31D
```

```
O    29           A32B   27           A32A   28           A32D
C     4           A33B    1           A33A    2           A33D
C    33           A34B    4           A34A    1           A34D
C    34           A35B   33           A35A    4           A35D
O    33           A36B   35           A36A   34           A36D
O    34           A37B   35           A37A   33           A37D
O    35           A38B   33           A38A   34           A38D
```

We use the following frozen variables in the optimizations:

```
x1          180.00000000
A7B           1.25000018
A8A          55.74675452
A9D           0.03089568
A17B          1.25000028
x2            0.00000000
A18A         56.62702009
A19D         -0.14060370
A27B          1.40245011
A28A         36.77258089
A29D         -5.66520328
A33B          1.88220570
A34A         22.52656874
A35D          7.19375855
```

The values above are for the specimen node $(x_1, x_2) = (180, 0)$. Importantly, for the comparison of stationary points between DFT and the surrogates, we do not freeze x_1 and x_2 in the DFT calculations. We employed four dummy atoms (X) to define the Z-matrix. To eliminate the additional $4 \cdot 3 = 12$ degrees of freedom, additional constraints were introduced into the Z-matrix. Because exactly three constraints were introduced for each dummy atom with respect to three carbon atoms, no additional constraints were applied to other atoms.

Note that in principle the constraints applied to dummy atoms do not prevent the distortion of Cp rings to be coupled to the rotation of Cp ring along the x_2 coordinate. However, we did not observe any significant geometry changes of Cp rings during geometry relaxations. As a result, the dummy atoms X_1 and X_3 always remain close to the center of Cp rings, which makes the coordinates x_1 and x_2 good approximations for the rotation of the [W(Cp)(CO)$_3$] monomer and the rotation of one of the Cp rings.

Finally, we have the basis set footer at the end of the input file:

```
W 0
sdd
****
O C H 0
3-21G
****

W 0
sdd
```

A.1.2 Comparison of DFT with crystal structure

In Table A.1, we compare selected structural parameters for $[W(Cp)(CO)_3]_2$ complex obtained at the B3LYP/(SDD,3-21G) level of theory with the crystal structure [26]. The angle bracket, $\langle\cdot\rangle$, represents the average value of structural parameters of the same type. $R(\mathrm{W{-}C_{Cp}})$ and $R(\mathrm{W{-}C_{CO}})$ denote the W–C bond lengths for C in the Cp ring and CO groups, respectively. $\langle\theta(\mathrm{H{-}C{-}W{-}W})\rangle$ represents the average of $\theta(\mathrm{H1{-}C1{-}W1{-}W2})$ and $\theta(\mathrm{H2{-}C2{-}W2{-}W1})$, and $\langle\theta(\mathrm{C{-}W{-}W{-}C})\rangle$ is the average of $\langle\theta(\mathrm{C1{-}W1{-}W2{-}C4})\rangle$ and $\langle\theta(\mathrm{C2{-}W2{-}W1{-}C3})\rangle$. All atoms are labeled in the same way as in Figure 3 of Chapter 5. The crystal structure agrees closely with the optimized global minimum from Gaussian, indicating that our model chemistry is suitably accurate.

	Crystal structure	Optimized structure
R(W1 −W2) [Å]	3.22	3.29
$\langle R(\mathrm{W{-}C_{Cp}})\rangle$ [Å]	2.34	2.42
$\langle R(\mathrm{W{-}C_{CO}})\rangle$ [Å]	1.98	1.99
θ(C1−W1−W2−C2) [deg]	180.0	180.0
$\langle\theta(\mathrm{H{-}C{-}W{-}W})\rangle$ [deg]	3.9	0.0
θ(C3−W1−W2−C4) [deg]	180.0	180.0
$\langle\theta(\mathrm{C{-}W{-}W{-}C})\rangle$ [deg]	2.4	0.0

Table A.1 Comparison of crystal structure with optimized structure at B3LYP/(SDD, 3-21G) level of theory.

A.2 Azomethane

A.2.1 Details of electronic structure calculations

All electronic structure calculations occur in Gaussian 16. Each optimization requests 16 cores and 32GB RAM. We have included two example input files, corresponding to a linear and nonlinear structure. The difference between the two is that the `modredundant` section includes the `L` keyword for linear structures, while for nonlinear structures, `L` is replaced by `A` and `D`.

All calculations of excitation energies and nonadiabatic coupling vectors occur in Orca as some of them employ open-shell ground state as a reference, for which the TD-DFT implementation in Orca is more suitable.

A.2.1.1 Example 1: linear structure

```
%chk=chkfiles/scan_zmat_1.chk
%mem=32GB
%nprocshared=16
# stable=opt b3lyp/6-311g(d) pop=always integral=grid=ultrafine
scf=(xqc,vtl,maxconventionalcycles=512) symmetry=none

azomethane stability 1

0 1
N       0.00000000      0.00000000      0.00000000
N       1.21001503      0.00000000      0.00000000
C       1.94413820      1.36087129      0.00000000
C      -1.80000000      0.00000000      0.00000000
H       3.01529271      1.16666079      0.00000000
H       1.65967212      1.92152238     -0.89311136
H       1.65967124      1.92152000      0.89311218
H      -2.30534096      0.96713099      0.00000000
H      -1.92067884     -0.57054140     -0.90903927
H      -1.92068356     -0.57054030      0.90903849

--Link1--
%chk=chkfiles/scan_zmat_1.chk
%mem=32GB
%nprocshared=16
# opt=(newton,tight,modredundant,maxstep=1,calcall,maxcycles=30)
geom=check guess=read b3lyp/6-311g(d) pop=always integral=grid=ultrafine
scf=(xqc,vtl,maxconventionalcycles=512) symmetry=none

azomethane opt 1

0 1

B 1 4 F
D 1 2 3 5 F
L 2 1 4 3 F
D 8 4 1 3 F

--Link1--
%chk=chkfiles/scan_zmat_1.chk
```

```
%mem=32GB
%nprocshared=16
# stable=opt geom=check guess=read b3lyp/6-311g(d) pop=always
integral=grid=ultrafine scf=(xqc,vtl,maxconventionalcycles=512)
symmetry=none

azomethane stability 2

0 1

--Link1--
%chk=chkfiles/scan_zmat_1.chk
%mem=32GB
%nprocshared=16
# opt=(newton,tight,modredundant,maxstep=1,calcall,maxcycles=30) freq
guess=read geom=check b3lyp/6-311g(d) pop=always integral=grid=ultrafine
scf=(xqc,vtl,maxconventionalcycles=512) symmetry=none

azomethane opt 2

0 1

--Link1--
%chk=chkfiles/scan_zmat_1.chk
%mem=32GB
%nprocshared=16
# stable=opt geom=check guess=read b3lyp/6-311g(d) pop=always
integral=grid=ultrafine scf=(xqc,vtl,maxconventionalcycles=512)
symmetry=none

azomethane stability 3

0 1
```

A.2.1.2 Example 2: nonlinear structure

```
%chk=chkfiles/scan_zmat_2.chk
%mem=32GB
%nprocshared=16
# stable=opt b3lyp/6-311g(d) pop=always integral=grid=ultrafine
scf=(xqc,vtl,maxconventionalcycles=512) symmetry=none

azomethane stability 1

0 1
N      0.00000000      0.00000000      0.00000000
N      1.23874400      0.00000000      0.00000000
C      1.80434700      1.38280600      0.00000000
C      0.00000000     -1.80000000      0.00000000
H      2.89084000      1.30705300      0.00000000
H      1.47283200      1.92607900     -0.88854700
H      1.47283200      1.92607900      0.88854700
H     -1.02105000     -2.18591100      0.00000000
H      0.54010800     -2.04428300     -0.90535200
H      0.54010800     -2.04428300      0.90535200

--Link1--
%chk=chkfiles/scan_zmat_2.chk
%mem=32GB
%nprocshared=16
# opt=(newton,modredundant,maxstep=1,calcall,maxcycles=30) geom=check
guess=read b3lyp/6-311g(d) pop=always integral=grid=ultrafine
scf=(xqc,vtl,maxconventionalcycles=512) symmetry=none

azomethane opt 1

0 1

B 1 4 F
D 1 2 3 5 F
D 4 1 2 3 F
A 2 1 4 F
D 8 4 1 2 F

--Link1--
```

```
%chk=chkfiles/scan_zmat_2.chk
%mem=32GB
%nprocshared=16
# stable=opt geom=check guess=read b3lyp/6-311g(d) pop=always
integral=grid=ultrafine scf=(xqc,vtl,maxconventionalcycles=512)
symmetry=none

azomethane stability 2

0 1

--Link1--
%chk=chkfiles/scan_zmat_2.chk
%mem=32GB
%nprocshared=16
# opt=(newton,modredundant,maxstep=1,calcall,maxcycles=30) freq
guess=read geom=check b3lyp/6-311g(d) pop=always integral=grid=ultrafine
scf=(xqc,vtl,maxconventionalcycles=512) symmetry=none

azomethane opt 2

0 1

--Link1--
%chk=chkfiles/scan_zmat_2.chk
%mem=32GB
%nprocshared=16
# stable=opt geom=check guess=read b3lyp/6-311g(d) pop=always
integral=grid=ultrafine scf=(xqc,vtl,maxconventionalcycles=512)
symmetry=none

azomethane stability 3

0 1
```

A.2.1.3 Example 3: excited states

The Orca input file `azo.inp` (with stability checks) is shown below:

```
!B3LYP 6-311g(d) TightSCF
%CIS
   HFNacme true
END
%SCF
   MaxIter 100
   HFType UHF
   STABPerform true
END
%TDDFT
   NROOTS   10
END
* xyzfile 0 1 geomfiles/azo.xyz
```

The file `geomfiles/azo.xyz` contains the geometry specified in Cartesian coordinates:

```
10
azomethane
N         0.00000000    0.00000000    0.00000000
N         1.22915061    0.00000000    0.00000000
C         1.87275627    1.37687217    0.00000000
C        -1.61279073    0.01401353   -0.00653607
H         2.62923902    1.39536387   -0.79264170
H         1.13534982    2.18398066   -0.12897983
H         2.39160980    1.47760052    0.95776512
H        -1.88615071    0.21731351    1.02800251
H        -1.91964069    0.83289353   -0.66933412
H        -1.97129227   -0.96664721   -0.33283514
```

A.2.2 Final refinement

Table A.2 shows the parameters used in the grids after all refinement was completed. Table A.3 contains the stationary points and energies (relative to the *trans-* minimum on a particular surface) for the grids specified in Table A.2. The energy relative to the global minimum is within ∼5% or 1 kcal/mol of the Gaussian-optimized value. Furthermore, since the predicted mixed-basis energies are all smaller than the Gaussian-optimized values, the relative barriers in the mixed-basis surrogate remain comparable.

	Trig grid	Polynomial grid
Mixed basis	$d = 3$ $L = 5$ $\boldsymbol{\alpha} = (1, 2, 2)$	$d = 2$ $L = 8$ $\boldsymbol{\alpha} = \mathbf{1}$
All polynomial	— — —	$d = 5$ $L = 12$ $\boldsymbol{\alpha} = \mathbf{1}$

Table A.2 Specification of final grids.

Structure	Surrogate	Optimized $\boldsymbol{q}$	ΔE (kcal/mol)
trans- minimum	Gaussian	$[180.00, 122.20, 122.20, 1.46, 113.00]$	0
	mixed	$[-179.71, 121.91, 121.84, 1.46, 116.06]$	0
	polynomial	$[-179.63, 137.91, -155.80, 1.47, 113.16]$	0
cis- minimum	Gaussian	$[0.00, -119.38, -60.33, 1.48, 120.21]$	10.48
	mixed	$[0.95, -118.43, -50.84, 1.49, 124.18]$	9.57
	polynomial	$[1.29, -121.79, -63.01, 1.48, 119.32]$	10.11
Inversion TS	Gaussian	$[-180.00, -97.52, 121.18, 1.39, 180.00]$	51.02
	mixed	$[-92.16, 123.10, 121.72, 1.40, 180.00]$	50.12
	polynomial	$[-171.02, -107.46, 121.35, 1.39, 175.84]$	49.49
Torsion TS	Gaussian	$[-89.50, 145.77, 55.27, 1.47, 117.79]$	45.71
	mixed	$[-87.46, 138.92, 141.04, 1.46, 118.78]$	43.16
	polynomial	$[-88.54, 138.50, 7.08, 1.47, 114.54]$	47.80

Table A.3 Locations of minima and transition states, as well as energy barriers relative to the *trans*-minimum on a given PES.

APPENDIX

B

CODES

B.1 MD Codes: User's Guide

This purpose of this section is to describe how to use the reduced-dimensional MD codes as well as the backend software to populate the nodes.

The first step is to install Tasmanian, described in Section 3.1 and at `https://tasmanian.ornl.gov/` (online manual under "Manual" tab). Skilled Unix users may want to use the `cmake` build options; other users are encouraged to use the basic `install.sh` script or `pip`. Regardless of the installation option, make sure that the MATLAB wrapper is enabled. All codes have been verified to work with Tasmanian v7.1 and Python 3.8 (Anaconda). After installing Tasmanian, add the installation directory to your Python and MATLAB paths as described in Section 3.1.

Next, clone the Github repository for Chapter 6: `https://github.com/zbmorrow/mixed_basis_rrmd/`. In the terminal, navigate into your local copy and type the following commands:

```
cd examples
python md_rrSurf_mixed.py 1 100 0.5
python md_rrSurf_poly.py 1 100 0.5
python md_rrSurf_thermo.py 1 100 0.5
```

Do not be concerned if `md_rrSurf_poly.py` takes longer than the other two codes. These commands will run the examples with the trajectory index set to 1, the initial/target temperature set to 100 Kelvin, and the final time set to 0.5 fs. The default input values are (respectively) 1, 298.15 K, and 2.5 ps (NVE) or 20 ps (Langevin thermostat). The trajectory index keeps track of each specific member of the swarm. In the `src` folder, `rrmd_core.py` contains MD-specific functions, while interpolation backend functions are stored in `rrmd_math_utils.py`.

Lastly, clone the Github repository for the backend software: `https://github.ncsu.edu/zbmorrow/populate_nodes/`. This repository contains the codes to generate the mixed-basis grid, create Gaussian input files for each node, harvest energies, and plot the interpolant. MATLAB, the `bash` shell, and the `pcregrep` command-line package are required on the local machine; access to Gaussian is necessary to run the input files.

To have a broad example base, this repository uses 2-butene as the molecule of interest. The design variables are the C3–C1–C2–C4 torsion angle (q_1) and the C1–C2 bond length (q_2). To replicate the work-flow, use the following process:

1. Create the directories `infiles`, `logfiles`, `chkfiles`, and `seedfiles` if they do not exist.
2. In the `seedfiles` directory, create a template Gaussian input file with the design variables declared as `FOO 0.0` on their respective lines. An example, `scan_zmat.gjf`, is already there. **This file is molecule-specific.**
3. Generate the grid by running `gen_grid.m`. **While not molecule-specific, the grid will change if you do any refinement.**
4. Make the Gaussian input files by typing `bash gen_jobs.sh` into the terminal. This will automatically replace the frozen variables in the template input file with the node values at the grid. **This file is molecule-specific.**
5. Upload your local project folder to the computing environment where you will run Gaussian.
6. Run the electronic structure calculations and label the output files with the same index as the corresponding input file. The script `run_jobs.sh` will do this for you on NCSU Henry2. Be sure that `infiles`, `logfiles`, `chkfiles`, and `seedfiles` exist in the remote submit folder where `run_jobs.sh` lives.
7. Download the finished `logfiles` directory from the cluster to your local project folder. Navigate back to the project folder, and type `bash extract_energies.sh` in your local terminal.
8. Run `plot_interp.m` to plot the interpolant.

Though this repository does not address how to harvest the data for the Cartesian mapping function $\boldsymbol{X}(\boldsymbol{q})$, pre-existing codes will do the job. For example, to apply the transformations in Section 6.2.2, we used `prepare_files_c2z.py`, located in the `surface_dynamic` repository at

```
src/utilities/data_prepare/prepare_files_c2z.py
```

This will store the geometries for each node in Z-matrix format. From there, we used azomethane-specific code to convert back to Cartesian coordinates systematically, placing N1 at the origin, N2 on the positive x-axis, and C2 in Quadrant I of the xy-plane.

B.2 Tasmanian

B.2.1 Interface examples

B.2.1.1 Python

In this example, we interpolate $f(x_1, x_2) = x_1 \sin(x_2)$ on $[-2, 5] \times [-10, 10]$ in the Python interface. We will use a global grid with the Clenshaw–Curtis points, the isotropic Smolyak tensor space, $L = 4$, $d = 2$, and one output dimension on the interval using the Python interface. We will also evaluate this grid at the points $[0.5, 0.2]$ and $[1.5, 7]$.

From the Tasmanian manual [200], the user may look up the keyword that corresponds to the tensor-selection strategy for Θ. In our case, "level" is the classical Smolyak space.

In Python:

```python
import sys
sys.path.append("<tasmanian-install-directory>/share/Tasmanian/python")
import Tasmanian as tsg
import numpy as np

grid = tsg.SparseGrid()
grid.makeGlobalGrid(2,1,4,"level","clenshaw-curtis")

# set the domain to [-2,5] x [-10,10]
grid.setDomainTransform(np.array([[-2,5],[-10,10]]))

nodes = grid.getNeededPoints()
fvals = nodes[:,0] * np.sin(nodes[:,1])     # generate data
```

```
# reshape so that col dim is explicitly 1
fvals = np.reshape(fvals, (-1,1))

grid.loadNeededPoints(fvals)
y = grid.evaluateBatch(np.array([[0.5,0.2],[-1.5,7]]))
```

B.2.1.2 MATLAB

We repeat the above example with the MATLAB interface. We first verify the ordering of the input arguments:

```
help tsgMakeGlobal
```

The MATLAB code is as follows:

```
addpath('<tasmanian-install-directory>/share/Tasmanian/matlab');
[example_grid, nodes] =
↪  tsgMakeGlobal('thesis_example',2,1,'clenshaw-curtis','level', 4, [-2, 5;
↪  -10, 10]);
fvals = nodes(:,1) .* sin(nodes(:,2));    % generate data
tsgLoadValues(example_grid, fvals);    % load data
y = tsgEvaluate(example_grid, [0.5, 0.2; -1.5, 7]);    % evaluate
tsgDeleteGrid(example_grid);    % remove temp files from MATLAB folder
```

B.2.2 Source code for trigonometric grids

- Git commit hash for this version: `cce29011e624d798fd66731ed970397ab8525469`
- Pull requests on Tasmanian GitHub with my commit history: 39, 50, 51, 56, 68, 74, 102, 104, 105, 110, 122, 292, 296–303, 420, 562

B.2.2.1 `tsgGridFourier.hpp` (version as of Aug 9, 2018)

```
/*
* Copyright (c) 2017, Miroslav Stoyanov
*
* This file is part of
* Toolkit for Adaptive Stochastic Modeling And Non-Intrusive ApproximatioN: TASMANIAN
*
* Redistribution and use in source and binary forms, with or without modification, are
↪  permitted provided that the following conditions are met:
*
* 1. Redistributions of source code must retain the above copyright notice, this list of
↪  conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright notice, this list of
↪  conditions
*    and the following disclaimer in the documentation and/or other materials provided with the
↪  distribution.
*
* 3. Neither the name of the copyright holder nor the names of its contributors may be used to
↪  endorse
*    or promote products derived from this software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS
↪  OR IMPLIED WARRANTIES,
* INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
↪  PARTICULAR PURPOSE ARE DISCLAIMED.
* IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
↪  INCIDENTAL, SPECIAL, EXEMPLARY,
* OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
↪  SERVICES; LOSS OF USE, DATA,
* OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
↪  IN CONTRACT, STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
↪  SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
```

```
* UT-BATTELLE, LLC AND THE UNITED STATES GOVERNMENT MAKE NO REPRESENTATIONS AND DISCLAIM ALL
↪  WARRANTIES, BOTH EXPRESSED AND IMPLIED.
* THERE ARE NO EXPRESS OR IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR
↪  PURPOSE, OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY PATENT,
* COPYRIGHT, TRADEMARK, OR OTHER PROPRIETARY RIGHTS, OR THAT THE SOFTWARE WILL ACCOMPLISH THE
↪  INTENDED RESULTS OR THAT THE SOFTWARE OR ITS USE WILL NOT RESULT IN INJURY OR DAMAGE.
* THE USER ASSUMES RESPONSIBILITY FOR ALL LIABILITIES, PENALTIES, FINES, CLAIMS, CAUSES OF
↪  ACTION, AND COSTS AND EXPENSES, CAUSED BY, RESULTING FROM OR ARISING OUT OF,
* IN WHOLE OR IN PART THE USE, STORAGE OR DISPOSAL OF THE SOFTWARE.
*/

#ifndef __TASMANIAN_SPARSE_GRID_FOURIER_HPP
#define __TASMANIAN_SPARSE_GRID_FOURIER_HPP

#include <cstdlib>
#include <math.h>
#include <complex>

#include "tsgEnumerates.hpp"
#include "tsgIndexSets.hpp"
#include "tsgCoreOneDimensional.hpp"
#include "tsgIndexManipulator.hpp"
#include "tsgLinearSolvers.hpp"
#include "tsgOneDimensionalWrapper.hpp"
#include "tsgGridCore.hpp"

#include "tsgAcceleratedDataStructures.hpp"

namespace TasGrid{

        class GridFourier : public BaseCanonicalGrid {
                public:
                GridFourier();
                GridFourier(const GridFourier &fourier);
                ~GridFourier();

                void write(std::ofstream &ofs) const;
                void read(std::ifstream &ifs, std::ostream *logstream = 0);

                void writeBinary(std::ofstream &ofs) const;
                void readBinary(std::ifstream &ifs, std::ostream *logstream = 0);

```

```cpp
    void makeGrid(int cnum_dimensions, int cnum_outputs, int depth, TypeDepth type, const int* anisotropic_weights = 0, const int* level_limits = 0);
    void copyGrid(const GridFourier *fourier);

    void setTensors(IndexSet* &tset, int cnum_outputs);
    int* referenceExponents(const int levels[], const IndexSet *list);

    int getNumDimensions() const;
    int getNumOutputs() const;
    TypeOneDRule getRule() const;

    int getNumLoaded() const;
    int getNumNeeded() const;
    int getNumPoints() const; // returns the number of loaded points unless no points are loaded, then returns the number of needed points

    void loadNeededPoints(const double *vals, TypeAcceleration acc = accel_none);

    void getLoadedPoints(double *x) const;
    void getNeededPoints(double *x) const;
    void getPoints(double *x) const; // returns the loaded points unless no points are loaded, then returns the needed points

    void getInterpolationWeights(const double x[], double weights[]) const;

    void getQuadratureWeights(double weights[]) const;

    void evaluate(const double x[], double y[]) const;
    void evaluateBatch(const double x[], int num_x, double y[]) const;

    void evaluateFastCPUblas(const double x[], double y[]) const;
    void evaluateFastGPUcublas(const double x[], double y[], std::ostream *os) const;
    void evaluateFastGPUcuda(const double x[], double y[], std::ostream *os) const;
    void evaluateFastGPUmagma(int gpuID, const double x[], double y[], std::ostream *os) const;

    void evaluateBatchCPUblas(const double x[], int num_x, double y[]) const;
    void evaluateBatchGPUcublas(const double x[], int num_x, double y[], std::ostream *os) const;
    void evaluateBatchGPUcuda(const double x[], int num_x, double y[], std::ostream *os) const;
```

```
        void evaluateBatchGPUmagma(int gpuID, const double x[], int num_x, double y[],
        ↪  std::ostream *os) const;

        void integrate(double q[], double *conformal_correction) const;

        void evaluateHierarchicalFunctions(const double x[], int num_x, double y[])
        ↪  const;
        void evaluateHierarchicalFunctionsInternal(const double x[], int num_x, double
        ↪  M_real[], double M_imag[]) const;
        void setHierarchicalCoefficients(const double c[], TypeAcceleration acc,
        ↪  std::ostream *os);

        void clearAccelerationData();
        void clearRefinement();
        void mergeRefinement();

        const int* getPointIndexes() const;
        const IndexSet* getExponents() const;
        const double* getFourierCoefs() const;

        protected:
        void reset();
        void calculateFourierCoefficients();

        int convertIndexes(const int i, const int levels[]) const;

        template<bool interwoven>
        void computeExponentials(const double x[], double w[]) const{
                std::complex<double> unit_imag(0.0,1.0);
                std::complex<double> **cache = new
                ↪  std::complex<double>*[num_dimensions];
                int *middles = new int[num_dimensions];
                for (int j=0; j<num_dimensions; j++){
                        int num_level_points = wrapper->getNumPoints(max_levels[j]);
                        int middle = (num_level_points - 1)/2;
                        middles[j] = middle;
                        cache[j] = new std::complex<double>[num_level_points];
                        cache[j][middle] = std::complex<double>(1.0,0.0);
                        if (num_level_points > 1){
                                cache[j][middle+1] = std::exp(2*M_PI*unit_imag*x[j]);
                                for (int i=middle+2; i<num_level_points; i++){
                                        cache[j][i] = cache[j][middle+1] *
                                        ↪  cache[j][i-1];
                }

                cache[j][middle-1] = std::conj(cache[j][middle+1]);
                for (int i=middle-2; i>=0; i--){
                    cache[j][i] = cache[j][middle-1] * cache[j][i+1];
                }
            }
        }

        IndexSet *work = (points == 0 ? needed : points);
        int num_points = work->getNumIndexes();

        for (int i=0; i<num_points; i++){
            std::complex<double> basis_entry(1.0,0.0);
            for (int j=0; j<num_dimensions; j++) basis_entry *= cache[j][exponents->getIndex(i)[j] + middles[j]];
            if (interwoven){
                w[2*i] = basis_entry.real();
                w[2*i + 1] = basis_entry.imag();
            }else{
                w[i] = basis_entry.real();
                w[i + num_points] = basis_entry.imag();
            }
        }

        for (int j=0; j<num_dimensions; j++) { delete[] cache[j]; }
        delete[] cache;
        delete[] middles;
    }

    private:
    int num_dimensions, num_outputs;

    OneDimensionalWrapper *wrapper;

    IndexSet *tensors;
    IndexSet *active_tensors;
    int *active_w;
    int *max_levels;
    IndexSet *points;
    IndexSet *needed;
    IndexSet *exponents;
```

```

                double *fourier_coefs;
                int **exponent_refs;
                int **tensor_refs;

                StorageSet *values;

                mutable BaseAccelerationData *accel;

        };

}

#endif
```

B.2.2.2 tsgGridFourier.cpp (version as of Aug 9, 2018)

```
/*
* Copyright (c) 2017, Miroslav Stoyanov
*
* This file is part of
* Toolkit for Adaptive Stochastic Modeling And Non-Intrusive ApproximatioN: TASMANIAN
*
* Redistribution and use in source and binary forms, with or without modification, are
↪  permitted provided that the following conditions are met:
*
* 1. Redistributions of source code must retain the above copyright notice, this list of
↪  conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright notice, this list of
↪  conditions
*    and the following disclaimer in the documentation and/or other materials provided with the
↪  distribution.
*
* 3. Neither the name of the copyright holder nor the names of its contributors may be used to
↪  endorse
*    or promote products derived from this software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS
↪  OR IMPLIED WARRANTIES,
* INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
↪  PARTICULAR PURPOSE ARE DISCLAIMED.
* IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
↪  INCIDENTAL, SPECIAL, EXEMPLARY,
* OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
↪  SERVICES; LOSS OF USE, DATA,
* OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
↪  IN CONTRACT, STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
↪  SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
* UT-BATTELLE, LLC AND THE UNITED STATES GOVERNMENT MAKE NO REPRESENTATIONS AND DISCLAIM ALL
↪  WARRANTIES, BOTH EXPRESSED AND IMPLIED.
* THERE ARE NO EXPRESS OR IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR
↪  PURPOSE, OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY PATENT,
* COPYRIGHT, TRADEMARK, OR OTHER PROPRIETARY RIGHTS, OR THAT THE SOFTWARE WILL ACCOMPLISH THE
↪  INTENDED RESULTS OR THAT THE SOFTWARE OR ITS USE WILL NOT RESULT IN INJURY OR DAMAGE.
```

```
* THE USER ASSUMES RESPONSIBILITY FOR ALL LIABILITIES, PENALTIES, FINES, CLAIMS, CAUSES OF
↪  ACTION, AND COSTS AND EXPENSES, CAUSED BY, RESULTING FROM OR ARISING OUT OF,
* IN WHOLE OR IN PART THE USE, STORAGE OR DISPOSAL OF THE SOFTWARE.
*/

#ifndef __TASMANIAN_SPARSE_GRID_FOURIER_CPP
#define __TASMANIAN_SPARSE_GRID_FOURIER_CPP

#include "tsgGridFourier.hpp"
#include "tsgHiddenExternals.hpp"

namespace TasGrid{

        GridFourier::GridFourier() : num_dimensions(0), num_outputs(0), wrapper(0), tensors(0),
        ↪  active_tensors(0), active_w(0),
        max_levels(0), points(0), needed(0), exponents(0), fourier_coefs(0), exponent_refs(0),
        ↪  tensor_refs(0), values(0), accel(0)
        {}

        GridFourier::GridFourier(const GridFourier &fourier) : num_dimensions(0),
        ↪  num_outputs(0), wrapper(0), tensors(0), active_tensors(0),
        active_w(0), max_levels(0), points(0), needed(0), exponents(0), fourier_coefs(0),
        ↪  exponent_refs(0), tensor_refs(0), values(0), accel(0){
                copyGrid(&fourier);
        }

        GridFourier::~GridFourier(){ reset(); }

        void GridFourier::write(std::ofstream &ofs) const{
                ofs << std::scientific; ofs.precision(17);
                ofs << num_dimensions << " " << num_outputs << endl;
                if (num_dimensions > 0){
                        tensors->write(ofs);
                        active_tensors->write(ofs);
                        ofs << active_w[0];
                        for(int i=1; i<active_tensors->getNumIndexes(); i++){
                                ofs << " " << active_w[i];
                        }
                        ofs << endl;
                        if (points == 0){
                                ofs << "0" << endl;
                        }else{
                                ofs << "1 ";
```

```
				points->write(ofs);
			}
			if (needed == 0){
				ofs << "0" << endl;
			}else{
				ofs << "1 ";
				needed->write(ofs);
			}
			ofs << max_levels[0];
			for(int j=1; j<num_dimensions; j++){
				ofs << " " << max_levels[j];
			}
			ofs << endl;
			if (num_outputs > 0){
				values->write(ofs);
				if (fourier_coefs != 0){
					ofs << "1";
					for(int i=0; i < 2*num_outputs*getNumPoints(); i++){
						ofs << " " << fourier_coefs[i];
					}
				}else{
					ofs << "0";
				}
			}

			/* not needed right now; will need later for refinement
			if (updated_tensors != 0){
				ofs << "1" << endl;
				updated_tensors->write(ofs);
				updated_active_tensors->write(ofs);
				ofs << updated_active_w[0];
				for(int i=1; i<updated_active_tensors->getNumIndexes(); i++){
					ofs << " " << updated_active_w[i];
				}
			}else{
				ofs << "0";
			}
			*/

			ofs << endl;
		}
	}

```

```
void GridFourier::read(std::ifstream &ifs, std::ostream *logstream){
        reset();
        ifs >> num_dimensions >> num_outputs;
        if (num_dimensions > 0){
                int flag;

                tensors = new IndexSet(num_dimensions);  tensors->read(ifs);
                active_tensors = new IndexSet(num_dimensions); active_tensors->read(ifs);
                active_w = new int[active_tensors->getNumIndexes()];  for(int i=0; i<active_tensors->getNumIndexes(); i++){ ifs >> active_w[i]; }
                ifs >> flag; if (flag == 1){ points = new IndexSet(num_dimensions); points->read(ifs); }
                ifs >> flag; if (flag == 1){ needed = new IndexSet(num_dimensions); needed->read(ifs); }
                max_levels = new int[num_dimensions];  for(int j=0; j<num_dimensions; j++){ ifs >> max_levels[j]; }

                IndexSet *work = (points != 0) ? points : needed;

                if (num_outputs > 0){
                        values = new StorageSet(0, 0); values->read(ifs);
                        ifs >> flag;
                        if (flag == 1){
                                fourier_coefs = new double[2 * num_outputs * work->getNumIndexes()];
                                for(int i=0; i<2*num_outputs*work->getNumIndexes(); i++){
                                        ifs >> fourier_coefs[i];
                                }
                        }else{
                                fourier_coefs = 0;
                        }
                }

                IndexManipulator IM(num_dimensions);
                int oned_max_level = max_levels[0];
                int nz_weights = active_tensors->getNumIndexes();
                for(int j=1; j<num_dimensions; j++){ if (oned_max_level < max_levels[j]) oned_max_level = max_levels[j]; }

                OneDimensionalMeta meta(0);
                wrapper = new OneDimensionalWrapper(&meta, oned_max_level, rule_fourier, 0.0, 0.0, logstream);

                UnsortedIndexSet *exponents_unsorted = new UnsortedIndexSet(num_dimensions, work->getNumIndexes());
                int *exponent = new int[num_dimensions];

                for (int i=0; i<work->getNumIndexes(); i++){
                        for(int j=0; j<work->getNumDimensions(); j++){
                                exponent[j] = (work->getIndex(i)[j] % 2 == 0 ? -work->getIndex(i)[j]/2 : (work->getIndex(i)[j]+1)/2);
                        }
                        exponents_unsorted->addIndex(exponent);
                }

                exponents = new IndexSet(exponents_unsorted);
                delete[] exponent;
                delete exponents_unsorted;

                exponent_refs = new int*[nz_weights];
                tensor_refs = new int*[nz_weights];
                #pragma omp parallel for schedule(dynamic)
                for(int i=0; i<nz_weights; i++){
                        exponent_refs[i] = referenceExponents(active_tensors->getIndex(i), exponents);
                        tensor_refs[i] = IM.referenceNestedPoints(active_tensors->getIndex(i), wrapper, work);
                }
                work = 0;
        }
}

void GridFourier::writeBinary(std::ofstream &ofs) const{
        int num_dim_out[2];
        num_dim_out[0] = num_dimensions;
        num_dim_out[1] = num_outputs;
        ofs.write((char*) num_dim_out, 2*sizeof(int));
        if (num_dimensions > 0){
                tensors->writeBinary(ofs);
                active_tensors->writeBinary(ofs);
```

```
            ofs.write((char*) active_w, active_tensors->getNumIndexes() * sizeof(int));
            char flag;
            if (points == 0){
                flag = 'n'; ofs.write(&flag, sizeof(char));
            }else{
                flag = 'y'; ofs.write(&flag, sizeof(char));
                points->writeBinary(ofs);
            }
            if (needed == 0){
                flag = 'n'; ofs.write(&flag, sizeof(char));
            }else{
                flag = 'y'; ofs.write(&flag, sizeof(char));
                needed->writeBinary(ofs);
            }
            ofs.write((char*) max_levels, num_dimensions * sizeof(int));

            if (num_outputs > 0){
                values->writeBinary(ofs);
                if (fourier_coefs != 0){
                    flag = 'y'; ofs.write(&flag, sizeof(char));
                    ofs.write((char*) fourier_coefs, 2 * getNumPoints() * num_outputs * sizeof(double));
                }else{
                    flag = 'n'; ofs.write(&flag, sizeof(char));
                }
            }

            /* don't need this right now; will need later when refinement is added
            if (updated_tensors != 0){
                flag = 'y'; ofs.write(&flag, sizeof(char));
                updated_tensors->writeBinary(ofs);
                updated_active_tensors->writeBinary(ofs);
                ofs.write((char*) updated_active_w, updated_active_tensors->getNumIndexes() * sizeof(int));
            }else{
                flag = 'n'; ofs.write(&flag, sizeof(char));
            }
            */
        }
    }

    void GridFourier::readBinary(std::ifstream &ifs, std::ostream *logstream){
```

```
                reset();
                int num_dim_out[2];
                ifs.read((char*) num_dim_out, 2*sizeof(int));
                num_dimensions = num_dim_out[0];
                num_outputs = num_dim_out[1];

                if (num_dimensions > 0){
                        tensors = new IndexSet(num_dimensions);  tensors->readBinary(ifs);
                        active_tensors = new IndexSet(num_dimensions);
                        ↪  active_tensors->readBinary(ifs);
                        active_w = new int[active_tensors->getNumIndexes()];
                        ifs.read((char*) active_w, active_tensors->getNumIndexes() *
                        ↪  sizeof(int));

                        char flag;
                        ifs.read((char*) &flag, sizeof(char)); if (flag == 'y'){ points = new
                        ↪  IndexSet(num_dimensions); points->readBinary(ifs); }
                        ifs.read((char*) &flag, sizeof(char)); if (flag == 'y'){ needed = new
                        ↪  IndexSet(num_dimensions); needed->readBinary(ifs); }

                        IndexSet *work = (points != 0) ? points : needed;

                        max_levels = new int[num_dimensions];
                        ifs.read((char*) max_levels, num_dimensions * sizeof(int));

                        if (num_outputs > 0){
                                values = new StorageSet(0, 0); values->readBinary(ifs);
                                ifs.read((char*) &flag, sizeof(char));
                                if (flag == 'y'){
                                        fourier_coefs = new double[2* num_outputs *
                                        ↪  work->getNumIndexes()];
                                        ifs.read((char*) fourier_coefs, 2 * num_outputs *
                                        ↪  work->getNumIndexes() * sizeof(double));
                                }else{
                                        fourier_coefs = 0;
                                }
                        }

                        IndexManipulator IM(num_dimensions);
                        int nz_weights = active_tensors->getNumIndexes();
                        int oned_max_level;
                        oned_max_level = max_levels[0];
                        for(int j=1; j<num_dimensions; j++) if (oned_max_level < max_levels[j])
                        ↪  oned_max_level = max_levels[j];

                        OneDimensionalMeta meta(0);
                        wrapper = new OneDimensionalWrapper(&meta, oned_max_level,
                        ↪  rule_fourier, 0.0, 0.0, logstream);

                        UnsortedIndexSet* exponents_unsorted = new
                        ↪  UnsortedIndexSet(num_dimensions, work->getNumIndexes());
                        int *exponent = new int[num_dimensions];

                        for (int i=0; i<work->getNumIndexes(); i++){
                                for(int j=0; j<work->getNumDimensions(); j++){
                                        exponent[j] = (work->getIndex(i)[j] % 2 == 0 ?
                                        ↪  -work->getIndex(i)[j]/2 :
                                        ↪  (work->getIndex(i)[j]+1)/2);
                                }
                                exponents_unsorted->addIndex(exponent);
                        }
                        exponents = new IndexSet(exponents_unsorted);
                        delete[] exponent;
                        delete exponents_unsorted;

                        exponent_refs = new int*[nz_weights];
                        tensor_refs = new int*[nz_weights];
                        #pragma omp parallel for schedule(dynamic)
                        for(int i=0; i<nz_weights; i++){
                                exponent_refs[i] =
                                ↪  referenceExponents(active_tensors->getIndex(i), exponents);
                                tensor_refs[i] =
                                ↪  IM.referenceNestedPoints(active_tensors->getIndex(i),
                                ↪  wrapper, work);
                        }
                        work = 0;
                }
        }

        void GridFourier::reset(){
                clearAccelerationData();
                if (exponent_refs != 0){ for(int i=0; i<active_tensors->getNumIndexes(); i++){
                ↪  delete[] exponent_refs[i]; exponent_refs[i] = 0; } delete[] exponent_refs;
                ↪  exponent_refs = 0; }
                if (tensor_refs != 0){ for(int i=0; i<active_tensors->getNumIndexes(); i++){
                ↪  delete[] tensor_refs[i]; tensor_refs[i] = 0; } delete[] tensor_refs;
                ↪  tensor_refs = 0; }
                if (wrapper != 0){ delete wrapper; wrapper = 0; }
                if (tensors != 0){ delete tensors; tensors = 0; }
                if (active_tensors != 0){ delete active_tensors; active_tensors = 0; }
                if (active_w != 0){ delete[] active_w; active_w = 0; }
                if (max_levels != 0){ delete[] max_levels; max_levels = 0; }
                if (points != 0){ delete points; points = 0; }
                if (needed != 0){ delete needed; needed = 0; }
                if (exponents != 0){ delete exponents; exponents = 0; }
                if (values != 0){ delete values; values = 0; }
                if (fourier_coefs != 0){ delete[] fourier_coefs; fourier_coefs = 0; }
                num_dimensions = 0;
                num_outputs = 0;
        }

        void GridFourier::makeGrid(int cnum_dimensions, int cnum_outputs, int depth, TypeDepth
        ↪  type, const int* anisotropic_weights, const int* level_limits){
                IndexManipulator IM(cnum_dimensions);
                IndexSet *tset = IM.selectTensors(depth, type, anisotropic_weights,
                ↪  rule_fourier);
                if (level_limits != 0){
                        IndexSet *limited = IM.removeIndexesByLimit(tset, level_limits);
                        if (limited != 0){
                                delete tset;
                                tset = limited;
                        }
                }

                setTensors(tset, cnum_outputs);
        }

        void GridFourier::copyGrid(const GridFourier *fourier){
                IndexSet *tset = new IndexSet(fourier->tensors);
                setTensors(tset, fourier->num_outputs);
                if ((num_outputs > 0) && (fourier->points != 0)){ // if there are values inside
                ↪  the source object
                        loadNeededPoints(fourier->values->getValues(0));
                }
        }

        void GridFourier::setTensors(IndexSet* &tset, int cnum_outputs){
                reset();
                num_dimensions = tset->getNumDimensions();
                num_outputs = cnum_outputs;

                tensors = tset;
                tset = 0;

                IndexManipulator IM(num_dimensions);

                OneDimensionalMeta meta(0);
                max_levels = new int[num_dimensions];
                int max_level; IM.getMaxLevels(tensors, max_levels, max_level);
                wrapper = new OneDimensionalWrapper(&meta, max_level, rule_fourier);

                int* tensors_w = IM.makeTensorWeights(tensors);
                active_tensors = IM.nonzeroSubset(tensors, tensors_w);

                int nz_weights = active_tensors->getNumIndexes();

                active_w = new int[nz_weights];
                tensor_refs = new int*[nz_weights];
                int count = 0;
                for(int i=0; i<tensors->getNumIndexes(); i++){ if (tensors_w[i] != 0)
                ↪  active_w[count++] = tensors_w[i]; }

                delete[] tensors_w;

                needed = IM.generateNestedPoints(tensors, wrapper); // nested grids exploit
                ↪  nesting

                UnsortedIndexSet* exponents_unsorted = new UnsortedIndexSet(num_dimensions,
                ↪  needed->getNumIndexes());
                int *exponent = new int[num_dimensions];

                for (int i=0; i<needed->getNumIndexes(); i++){
                        for(int j=0; j<needed->getNumDimensions(); j++){
                                exponent[j] = (needed->getIndex(i)[j] % 2 == 0 ?
                                ↪  -needed->getIndex(i)[j]/2 : (needed->getIndex(i)[j]+1)/2);
                        }
                        exponents_unsorted->addIndex(exponent);
                }

                exponents = new IndexSet(exponents_unsorted);
```

```cpp
                delete[] exponent;
                delete exponents_unsorted;

                exponent_refs = new int*[nz_weights];
                #pragma omp parallel for schedule(dynamic)
                for(int i=0; i<nz_weights; i++){
                        exponent_refs[i] = referenceExponents(active_tensors->getIndex(i), exponents);
                        tensor_refs[i] = IM.referenceNestedPoints(active_tensors->getIndex(i), wrapper, needed);
                }

                if (num_outputs == 0){
                        points = needed;
                        needed = 0;
                }else{
                        values = new StorageSet(num_outputs, needed->getNumIndexes());
                }

        }

        int* GridFourier::referenceExponents(const int levels[], const IndexSet *list){

                /*
                * This function ensures the correct match-up between Fourier coefficients and basis functions.
                * The basis exponents are stored in an IndexSet whose ordering is different from the needed/points
                * IndexSet (since exponents may be negative).
                *
                * The 1D Fourier coefficients are \hat{f}^l_j, where j = -(3^l-1)/2, ..., 0, ..., (3^l-1)/2 and
                *
                * \hat{f}^l_j = \sum_{n=0}^{3^l-1} f(x_n) exp(-2 * pi * sqrt(-1) * j * x_n)
                *             = \sum_{n=0}^{3^l-1} f(x_n) exp(-2 * pi * sqrt(-1) * j * n/3^l)
                *
                * Now, \hat{f}^l_j is the coefficient of the basis function exp(2 * pi * sqrt(-1) * j * x).
                * However, the Fourier transform code returns the coefficients with the indexing j' = 0, 1,
                * ..., 3^l-1.  For j' = 0, 1, ..., (3^l-1)/2, the corresponding basis function has exponent j'.
```

```
                 * For j' = (3^l-1)/2 + 1, ..., 3^l-1, we march clockwise around the unit circle
 ↪   to see
                 * the corresponding exponent is -3^l + j'. Algebraically, for j' > (3^l-1)/2,
                 *
                 * \hat{f}^l_{j'} = \sum_{n=0}^{3^l-1} f(x_n) [exp(-2 * pi * sqrt(-1) * j'
 ↪   /3^l)]^n
                 *                = \sum_{n=0}^{3^l-1} f(x_n) [exp(-2 * pi * sqrt(-1) * (j' -
 ↪   3^l) /3^l) * exp(-2 * pi * sqrt(-1))]^n
                 *                = \sum_{n=0}^{3^l-1} f(x_n) [exp(-2 * pi * sqrt(-1) * (j' -
 ↪   3^l) /3^l)]^n
                 *                = \hat{f}^l_{j' - 3^l}
                 */
                int *num_points = new int[num_dimensions];
                int num_total = 1;
                for(int j=0; j<num_dimensions; j++){  num_points[j] =
                ↪  wrapper->getNumPoints(levels[j]); num_total *= num_points[j];  }

                int* refs = new int[num_total];
                int *p = new int[num_dimensions];

                for(int i=0; i<num_total; i++){
                        int t = i;
                        for(int j=num_dimensions-1; j>=0; j--){
                                int tmp = t % num_points[j];
                                p[j] = (tmp <= (num_points[j]-1)/2 ? tmp : -num_points[j] +
                                ↪  tmp);
                                t /= num_points[j];
                        }
                        refs[i] = list->getSlot(p);
                }

                delete[] p;
                delete[] num_points;

                return refs;
        }

        int GridFourier::getNumDimensions() const{ return num_dimensions; }
        int GridFourier::getNumOutputs() const{ return num_outputs; }
        TypeOneDRule GridFourier::getRule() const{ return rule_fourier; }

        int GridFourier::getNumLoaded() const{ return (((points == 0) || (num_outputs == 0)) ?
        ↪  0 : points->getNumIndexes()); }
```

```
int GridFourier::getNumNeeded() const{ return ((needed == 0) ? 0 :
↪  needed->getNumIndexes()); }
int GridFourier::getNumPoints() const{ return ((points == 0) ? getNumNeeded() :
↪  points->getNumIndexes()); }

void GridFourier::loadNeededPoints(const double *vals, TypeAcceleration){
        if (accel != 0) accel->resetGPULoadedData();

        if (points == 0){ //setting points for the first time
                values->setValues(vals);
                points = needed;
                needed = 0;
        }else{ //resetting the points
                values->setValues(vals);
        }
        //if we add anisotropic or surplus refinement, I'll need to add a third case
        ↪  here

        calculateFourierCoefficients();
}

void GridFourier::getLoadedPoints(double *x) const{
        int num_points = points->getNumIndexes();
        #pragma omp parallel for schedule(static)
        for(int i=0; i<num_points; i++){
                const int *p = points->getIndex(i);
                for(int j=0; j<num_dimensions; j++){
                        x[i*num_dimensions + j] = wrapper->getNode(p[j]);
                }
        }
}
void GridFourier::getNeededPoints(double *x) const{
        int num_points = needed->getNumIndexes();
        #pragma omp parallel for schedule(static)
        for(int i=0; i<num_points; i++){
                const int *p = needed->getIndex(i);
                for(int j=0; j<num_dimensions; j++){
                        x[i*num_dimensions + j] = wrapper->getNode(p[j]);
                }
        }
}
void GridFourier::getPoints(double *x) const{
        if (points == 0){ getNeededPoints(x); }else{ getLoadedPoints(x); };
```

```
    }

    int GridFourier::convertIndexes(const int i, const int levels[]) const {

        /*
        * This routine ensures that function values are loaded into the Fourier transform in order of
        * increasing SPATIAL x-values.
        *
        * Take the example of the level-2 1D grid. Tasmanian's internal indexing reports the points as
        *
        * [0, 1, 2, ..., 8]
        *
        * which corresponds spatially to
        *
        * [0, 1/3, 2/3, 1/9, 2/9, 4/9, 5/9, 7/9, 8/9].
        *
        * That is, the new points added on level-2 appear after the level-1 points, even though spatially
        * they are interwoven. We now order the grid spatially, in terms of increasing x-values:
        *
        * [0, 1/9, ..., 8/9]
        *
        * The input parameter i is the i-th entry in the spatially ordered list. The output is the corresponding
        * internal index used by Tasmanian. For example, with p[0] = 2, convertIndexes(1, p) would return 3 (the
        * index of 1/9 in the internally ordered list of points).
        *
        * ANOTHER EXAMPLE: consider spatial index 48 on a grid of level 4. There are 81 points total, so spatial
        * index 48 is 48/81 = 16/27. So the point first appears on the level with 27 points (level 3). Internal
        * indexing first loads the 9 points of the level-2 grid. Spatially, there are floor(16/3) full subintervals
        * prior to arriving at 16/27 on the level-3 grid. In each of these 5 intervals, there are 2 new points,
        * and 16 is the first point in the 6th subinterval. So the internal index is 9 + 5*2 + 1 - 1 = 19 (minus
        * 1 since C++ indexing begins at 0).
        */

        IndexSet *work = (points == 0 ? needed : points);

        int* cnum_oned_points = new int[num_dimensions];
        for(int j=0; j<num_dimensions; j++){
                cnum_oned_points[j] = wrapper->getNumPoints(levels[j]);
        }

        // This i is spatial indexing
        int t=i;
        int* p=new int[num_dimensions];
        for(int j=num_dimensions-1; j>=0; j--){
                p[j] = t % cnum_oned_points[j];
                t /= cnum_oned_points[j];
        }
        // p[] stores the spatial index as a tensor address

        // Now we move from spatial to internal indexing
        int *p_internal = new int[num_dimensions];
        for(int j=0; j<num_dimensions; j++){
                if (p[j] == 0){
                        p_internal[j] = 0;
                }else{
                        int spatial_idx_when_added = p[j];
                        int division_count = 0;
                        while(spatial_idx_when_added % 3 == 0){ spatial_idx_when_added
                        ↪  /= 3; division_count++; }

                        int level_when_added = levels[j] - division_count;

                        // 3 spatial points in each added full subinterval; 2 new
                        ↪  points for internal indexing
                        int offset = 2 * (spatial_idx_when_added/3) +
                        ↪  (spatial_idx_when_added % 3);
                        p_internal[j] = wrapper->getNumPoints(level_when_added-1) +
                        ↪  offset - 1;
                }
        }

        int result = work->getSlot(p_internal);
        delete[] cnum_oned_points;
        delete[] p_internal;
        delete[] p;
        return result;
    }

    void GridFourier::calculateFourierCoefficients(){
        int num_points = getNumPoints();

        if (fourier_coefs != 0){ delete[] fourier_coefs; fourier_coefs = 0; }
        fourier_coefs = new double[2 * num_outputs * num_points];
        std::fill(fourier_coefs, fourier_coefs + 2*num_outputs*num_points, 0.0);
        for(int k=0; k<num_outputs; k++){
            for(int n=0; n<active_tensors->getNumIndexes(); n++){
                const int* levels = active_tensors->getIndex(n);
                int num_tensor_points = 1;
                int* num_oned_points = new int[num_dimensions];
                for(int j=0; j<num_dimensions; j++){
                    num_oned_points[j] = wrapper->getNumPoints(levels[j]);
                    num_tensor_points *= num_oned_points[j];
                }

                std::complex<double> *in = new
                ↪  std::complex<double>[num_tensor_points];
                std::complex<double> *out = new
                ↪  std::complex<double>[num_tensor_points];
                for(int i=0; i<num_tensor_points; i++){
                    // We interpret this "i" as running through the spatial
                    ↪  indexing; convert to internal
                    int key = convertIndexes(i, levels);
                    const double *v = values->getValues(key);
                    in[i] = v[k];
                }

                // Execute FFT
                TasmanianFourierTransform::discrete_fourier_transform(num_dimensions,
                ↪  num_oned_points, in, out);

                for(int i=0; i<num_tensor_points; i++){
                    // Combine with tensor weights
                    fourier_coefs[num_outputs*(exponent_refs[n][i]) + k] +=
                    ↪  ((double) active_w[n]) * out[i].real() / ((double)
                    ↪  num_tensor_points);
                    fourier_coefs[num_outputs*(exponent_refs[n][i] +
                    ↪  num_points) + k] += ((double) active_w[n]) *
                    ↪  out[i].imag() / ((double) num_tensor_points);
                }
                delete[] in;
                delete[] out;
                delete[] num_oned_points;
            }
        }
    }

    void GridFourier::getInterpolationWeights(const double x[], double weights[]) const {
        /*
        I[f](x) = c^T * \Phi(x) = (U*P*f)^T * \Phi(x)          (U represents normalized forward Fourier transform; P represents reordering of f_i before going into FT)
        = f^T * (P^T * U^T * \Phi(x))    (P^T = P^(-1) since P is a permutation matrix)

        Note that U is the DFT operator (complex) and the transposes are ONLY REAL transposes, so U^T = U.
        */

        std::fill(weights, weights+getNumPoints(), 0.0);
        double *basisFuncs = new double[2* getNumPoints()];
        computeExponentials<true>(x, basisFuncs);

        for(int n=0; n<active_tensors->getNumIndexes(); n++){
            const int *levels = active_tensors->getIndex(n);
            int num_tensor_points = 1;
            int *num_oned_points = new int[num_dimensions];

            for(int j=0; j<num_dimensions; j++){
                num_oned_points[j] = wrapper->getNumPoints(levels[j]);
                num_tensor_points *= num_oned_points[j];
            }

            std::complex<double> *in = new std::complex<double>[num_tensor_points];
            std::complex<double> *out = new std::complex<double>[num_tensor_points];

            for(int i=0; i<num_tensor_points; i++){
                in[i] = std::complex<double>(basisFuncs[2*exponent_refs[n][i]], basisFuncs[2*exponent_refs[n][i] + 1]);
            }

            TasmanianFourierTransform::discrete_fourier_transform(num_dimensions, num_oned_points, in, out);
```

```

                for(int i=0; i<num_tensor_points; i++){
                        int key = convertIndexes(i, levels);
                        weights[key] += ((double) active_w[n]) *
                        ↪  out[i].real()/((double) num_tensor_points);
                }
                delete[] in;
                delete[] out;
                delete[] num_oned_points;
        }
        delete[] basisFuncs;
}

void GridFourier::getQuadratureWeights(double weights[]) const{

        /*
        * When integrating the Fourier series on a tensored grid, all the
        * nonzero modes vanish, and we're left with the normalized Fourier
        * coeff for e^0 (sum of the data divided by number of points)
        */

        int num_points = getNumPoints();
        std::fill(weights, weights+num_points, 0.0);
        for(int n=0; n<active_tensors->getNumIndexes(); n++){
                const int *levels = active_tensors->getIndex(n);
                int num_tensor_points = 1;
                for(int j=0; j<num_dimensions; j++){
                        num_tensor_points *= wrapper->getNumPoints(levels[j]);
                }
                for(int i=0; i<num_tensor_points; i++){
                        weights[tensor_refs[n][i]] += ((double) active_w[n])/((double)
                        ↪  num_tensor_points);
                }
        }
}

void GridFourier::evaluate(const double x[], double y[]) const{
        int num_points = getNumPoints();
        double *w = new double[2 * num_points];
        computeExponentials<false>(x, w);
        TasBLAS::setzero(num_outputs, y);
        for(int k=0; k<num_outputs; k++){
                for(int i=0; i<num_points; i++){
```

```
                                y[k] += (w[i] * fourier_coefs[i*num_outputs+k] -
                                ↪  w[i+num_points] *
                                ↪  fourier_coefs[(i+num_points)*num_outputs+k]);
                        }
                }
                delete[] w;
        }
        void GridFourier::evaluateBatch(const double x[], int num_x, double y[]) const{
                #pragma omp parallel for
                for(int i=0; i<num_x; i++){
                        evaluate(&(x[((size_t) i) * ((size_t) num_dimensions)]), &(y[((size_t)
                        ↪  i) * ((size_t) num_outputs)]));
                }
        }

        void GridFourier::evaluateFastCPUblas(const double x[], double y[]) const{
                evaluate(x,y);
        }
        void GridFourier::evaluateFastGPUcublas(const double x[], double y[], std::ostream*)
        ↪  const{
                evaluate(x,y);
        }
        void GridFourier::evaluateFastGPUcuda(const double x[], double y[], std::ostream*)
        ↪  const{
                evaluate(x,y);
        }
        void GridFourier::evaluateFastGPUmagma(int, const double x[], double y[],
        ↪  std::ostream*) const{
                evaluate(x,y);
        }

        void GridFourier::evaluateBatchCPUblas(const double x[], int num_x, double y[]) const{
                evaluateBatch(x, num_x, y);
        }
        void GridFourier::evaluateBatchGPUcublas(const double x[], int num_x, double y[],
        ↪  std::ostream*) const {
                evaluateBatch(x, num_x, y);
        }
        void GridFourier::evaluateBatchGPUcuda(const double x[], int num_x, double y[],
        ↪  std::ostream*) const {
                evaluateBatch(x, num_x, y);
        }
```

```
void GridFourier::evaluateBatchGPUmagma(int, const double x[], int num_x, double y[],
↪  std::ostream*) const {
        evaluateBatch(x, num_x, y);
}

void GridFourier::integrate(double q[], double *conformal_correction) const{
        std::fill(q, q+num_outputs, 0.0);
        if (conformal_correction == 0){
                // everything vanishes except the Fourier coeff of e^0
                int *zeros_num_dim = new int[num_dimensions];
                std::fill(zeros_num_dim, zeros_num_dim + num_dimensions, 0);
                int idx = exponents->getSlot(zeros_num_dim);
                for(int k=0; k<num_outputs; k++){
                        q[k] = fourier_coefs[num_outputs * idx + k];
                }
                delete[] zeros_num_dim;
        }else{
                // Do the expensive computation if we have a conformal map
                double *w = new double[getNumPoints()];
                getQuadratureWeights(w);
                for(int i=0; i<points->getNumIndexes(); i++){
                        w[i] *= conformal_correction[i];
                        const double *v = values->getValues(i);
                        for(int k=0; k<num_outputs; k++){
                                q[k] += w[i] * v[k];
                        }
                }
                delete[] w;
        }
}

void GridFourier::evaluateHierarchicalFunctions(const double x[], int num_x, double
↪  y[]) const{
        // y must be of size 2*num_x*num_points
        int num_points = getNumPoints();
        #pragma omp parallel for
        for(int i=0; i<num_x; i++){
                computeExponentials<true>(&(x[((size_t) i) * ((size_t)
                ↪  num_dimensions)]), &(y[((size_t) i) * ((size_t) 2*num_points)]));
        }
}
void GridFourier::evaluateHierarchicalFunctionsInternal(const double x[], int num_x,
↪  double M_real[], double M_imag[]) const{
```

```
            // y must be of size num_x * num_nodes * 2
            int num_points = getNumPoints();
            #pragma omp parallel for
            for(int i=0; i<num_x; i++){
                    double *w = new double[2 * num_points];
                    computeExponentials<false>(&(x[((size_t) i) * ((size_t)
                    ↪  num_dimensions)]), w);
                    for(int m=0; m<num_points; m++){
                            M_real[i*num_points+m] = w[m];
                            M_imag[i*num_points+m] = w[m + num_points];
                    }
            }
    }

    void GridFourier::setHierarchicalCoefficients(const double c[], TypeAcceleration,
    ↪  std::ostream*){
            // takes c to be length 2*num_outputs*num_points
            // first num_points*num_outputs are the real part; second
            ↪  num_points*num_outputs are the imaginary part

            if (accel != 0) accel->resetGPULoadedData();
            if (points == 0){
                    points = needed;
                    needed = 0;
            }
            if (fourier_coefs != 0) delete[] fourier_coefs;
            fourier_coefs = new double[2 * getNumPoints() * num_outputs];
            for(int i=0; i<2 * getNumPoints() * num_outputs; i++) fourier_coefs[i] = c[i];
    }

    void GridFourier::clearAccelerationData(){
            if (accel != 0){
                    delete accel;
                    accel = 0;
            }
    }
    void GridFourier::clearRefinement(){ return; }     // to be expanded later
    void GridFourier::mergeRefinement(){ return; }     // to be expanded later

    const int* GridFourier::getPointIndexes() const{
            return ((points == 0) ? needed->getIndex(0) : points->getIndex(0));
    }
    const IndexSet* GridFourier::getExponents() const{
```

```
                return exponents;
        }
        const double* GridFourier::getFourierCoefs() const{
                return ((double*) fourier_coefs);
        }

} // end TasGrid

#endif
```