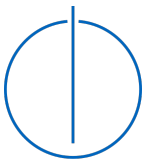
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Applying the Spatially Adaptive Combination Technique to Uncertainty Quantification

Fritz Hofmeier



DEPARTMENT OF INFORMATICS

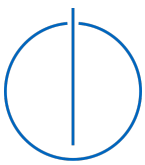
TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Applying the Spatially Adaptive Combination Technique to Uncertainty Quantification

Anwendung der räumlich-adaptiven Kombinationstechnik im Kontext der Uncertainty Quantification

Author:	Fritz Hofmeier
Supervisor:	Prof. Dr. rer. nat. habil. Hans-Joachim Bungartz
Advisor:	M.Sc. Michael Obersteiner
Submission Date:	September 16, 2019



I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, September 16, 2019 Fritz Hofmeier

Abstract

The topic of this bachelor thesis is the utilization of the single-dimension spatially adaptive sparse grid refinement strategy for uncertainty quantification. Frequent uncertainty quantification tasks are the calculation of moments and polynomial chaos expansion coefficents. Since these calculations require weighted multidimensional integration, the sparseSpACE Python framework is extended to support probability distributions as weight functions for high-dimensional adaptive quadrature. The adaptive refinement strategy operates on the individual problem function's input dimensions. This could make it especially suitable for uncertainty quantification if the independent uncertain parameters are loosely coupled. The test results indicate that for composite trapezoidal quadrature the implemented weighted integration is more accurate than integration with the inverse transformation method.

In dieser Bachelorarbeit wird die Single-Dimension räumlich-adaptive Sparse Grid Verfeinerungsstrategie für Uncertainty Quantification verwendet. Bei Uncertainty Quantification werden oft Momente und Polynomial Chaos Expansion Koeffizienten berechnet. Da dazu eine mehrdimensionale Funktion mit einer Gewichtungsfunktion integriert wird, wurde das sparseSpACE Python Framework erweitert, damit es Wahrscheinlichkeitsfunktionen für die Gewichtung bei hochdimensionaler adaptiver Quadratur unterstützt. Die adaptive Verfeinerung wird bei den individuellen Eingangsparametern der Funktion durchgeführt. Das Verfahren könnte sich insbesondere für Uncertainty Quantification eignen, wenn die unabhängigen unsicheren Parameter lose gekoppelt sind. Testergebnisse mit Trapezquadratur zeigen, dass die implementierte gewichtete Integration genauere Ergebnisse erzielt als Integration mittels der Inversionsmethode.

Contents

1. Introduction

When probability is involved in a problem, a frequent task is the integration of some function multiplied with a weight function. In uncertainty quantification (UQ), some input parameters of a function do not have a fixed value; instead they have a probability distribution. Here, we may need to calculate moments, with which we can then obtain the expectation and variance. In addition to the function's expectation and variance, we may want to obtain a polynomial chaos expansion, which involves weighted integration when calculating polynomial coefficients. Often more than one function parameter is uncertain which leads to a multidimensional integration task. To perform this task efficiently, it is possible to employ sparse grid integration methods. There exist a big number of variants of these methods; for example the sparse grid can be based on Composite Trapezoidal or on Clenshaw Curtis quadrature rules. Some methods refine the sparse grids adaptively, which means that they find quadrature poins and weights which fit well for a specific function. In this Bachelor Thesis, the single dimension spatially adaptive refinement method is employed for uncertainty quantification. The refinement and grid point and weight calculation differ from constant-weight integration because they need to be adjusted for the probability distributions.

As the name implies, the Background chapter explains background information about uncertainty quantification and sparse grids. The Spatially Adaptive Refinement for Uncertainty Quantification chapter describes how quadrature weight calculations and grid refinements work in theory. Some numerical issues with these calculations and information about how the Python code is structured can be found in the implementation chapter. Finally, the Test Results chapter shows how the adaptive refinement program performs with the Predator Prey differential equation, a simple test function where analytic solutions are available, and a discontinuous function.

2. Background

The first section in this chapter introduces Uncertainty Quantification. It describes the polynomial chaos expansion calculations, which can involve sensitivity analysis. The sparse grids section gives information about the sparse grid concept and hierarchical bases, and then it explaines the adaptive refinement method which is used in this thesis and is based on the combination technique. The last section shortly describes a dimension-adaptive refinement for uncertainty quantification from a related master thesis.

2.1. Uncertainty Quantification

When calculating a function output, we may not know the exact value of some input parameters. For example, a car crash test simulation could involve four uncertain tyre pressures and the overall vehicle weight. Some of these parameters have a common impact on the function output; in the example, the car may drive to the left when the left tyres have less pressure than the right ones. Uncertainty Quantification (UQ) addresses this vague input parameter problem if we can assign a probability density function (PDF) to these parameters and the parameters are independently distributed. Given the probability distributions, we can use the expectation as output value of the function and the variance vaguely represents the reliability of this output value. To determine how much impact an uncertain parameter has on the variance, we can employ sensitivity analysis. Further information, such as statistical moments, may also be of interest. [Nec18]

The deterministic function parameters, i.e. parameters without uncertainty, are fixed when calculating the expectation, variance or other UQ related values; therefore they are omitted here for brevity.

2.1.1. Polynomial Chaos Expansion

To efficiently calculate the sensitivity analysis, we can approximate the function with Polynomial Chaos Expansion (PCE). For simplicity, let us first assume that there is only one uncertain parameter x. In PCE, the function is transformed into a sum of pairwise orthogonal polynomials ϕ_k multiplied with weights a_k. The index k defines the degree

of the polynomial. Two polynomials ϕ_k and ϕ_l are orthogonal with respect to a PDF p if their inner product is zero:

$$\langle \phi_k, \phi_l \rangle_p = \int_\Omega \phi_k(\omega)\phi_l(\omega)p(\omega)d\omega = 0$$

The optimal choice of the polynomials depends on the probability distributions; for example, Hermite polynomials work best with normal distributions and Gauß-Legendre polynomials are optimal for Uniform distributions. A summand's impact on the output value depends on the polynomial degree, so as an approximation, the sum is truncated to exclude high order polynomials.

$$f_{pce}(x) = \sum_{k=0}^{P-1} a_k \phi_k(x)$$

The calculation of the coefficients a_k depends on the application. The Galerkin projection minimizes the error norm $\|f(x) - f_{pce}(x)\|_2$, which gives good results for smooth functions, whereas least squares regression minimizes a mean squared error over specified sampling points, which works well for noisy functions. [al15]

In case of Galerkin projection, the coefficient can be calculated as follows:

$$a_k = \frac{\langle f, \phi_k \rangle_p}{\langle \phi_k, \phi_k \rangle_p}$$

The denominator can be omitted if the polynomials are normalized. In the so-called pseudospectral approach, the numerator is approximated with numerical quadrature:

$$\langle f, \phi_k \rangle_p \approx \sum_{j=0}^{C-1} f(x_j)\phi_k(x_j)w_j dx$$

If f has d uncertain parameters instead of only one, $\vec{k} \in \mathbb{N}_0^d$ is a multidimensional index and the polynomials are the products of the respective single-dimensional polynomials. The degree of $\phi_{\vec{k}}$ is then the manhattan norm of k: $\|\vec{k}\|_1$, and the truncation omits the polynomials which have a degree higher than or equal to P.

$$\phi_{\vec{k}}(\vec{x}) = \prod_{i=1}^{d} \phi_{k_i}(x_i)$$
$$f_{pce}(\vec{x}) = \sum_{\|\vec{k}\|_1 < P} a_{\vec{k}} \phi_{\vec{k}}(\vec{x})$$

The pseudospectral calculation of the coefficients $a_{\vec{k}}$ now constitutes a multidimensional numerical quadrature where we can employ, for example, the Monte Carlo method or

sparse grids. If the maximum polynomial degree P goes to infinity and the coefficients are analytically calculated, the approximation f_{pce} is the same as f.

Since the polynomials are orthogonal and the first one is a constant function, we can calculate the expectation and variance from the PCE coefficients. If the polynomials are normed, all norms $\langle \phi_{\vec{k}}, \phi_{\vec{k}} \rangle_p$ are one in the following equations.

$$\mathbb{E}[f_{pce}(\vec{x})] = \int_\Omega f_{pce}(\vec{x}) p(\vec{x}) d\vec{x} = \int_\Omega f_{pce}(\vec{x}) \frac{\phi_{\vec{0}}(\vec{x})}{\langle \phi_{\vec{0}}, \phi_{\vec{0}} \rangle_p} p(\vec{x}) d\vec{x} = \frac{\langle f, \phi_{\vec{0}} \rangle_p}{\langle \phi_{\vec{0}}, \phi_{\vec{0}} \rangle_p} = a_{\vec{0}}$$

$$Var[f_{pce}(\vec{x})] = \mathbb{E}[(f_{pce}(\vec{x}) - \mathbb{E}[f_{pce}(\vec{x})])^2] = \mathbb{E}[(\sum_{\|\vec{k}\|_1 < P} a_{\vec{k}} \phi_{\vec{k}}(\vec{x}) - a_{\vec{0}})^2]$$

$$= \mathbb{E}[(\sum_{0 < \|\vec{k}\|_1 < P} a_{\vec{k}} \phi_{\vec{k}}(\vec{x}))^2] = \mathbb{E}[\sum_{0 < \|\vec{k}\|_1 < P} (a_{\vec{k}} \phi_{\vec{k}}(\vec{x}))^2] = \sum_{0 < \|\vec{k}\|_1 < P} a_{\vec{k}}^2 \langle \phi_{\vec{k}}, \phi_{\vec{k}} \rangle_p$$

2.1.2. Global Sensitivity Analysis

To analyse how much impact specific uncertain parameters have on the variance, we can calculate the sobol indices, which are based on the analysis of variance (ANOVA) decomposition. The ANOVA decomposition is defined as follows:

$$f(\vec{x}) = f_\emptyset() + \sum_{i=1}^{d} f_i(x_i) + \sum_{1 <= i < k <= d} f_{i,k}(x_i, x_k) + \ldots + f_{1,\ldots,d}(\vec{x})$$

$$= \sum_{u \subset U} f_u(\vec{x}_u), U = \{1, \ldots, d\}$$

$$\forall u \subset U, i \in u : \int f_u(\vec{x}_u) p(x_i) dx_i = 0$$

It is possible to calculate the functions f_u recursively:

$$f_\emptyset() = \mathbb{E}[f] = \int f(\vec{x}) p(\vec{x}) d\vec{x}$$

$$f_u(\vec{x}_u) = \mathbb{E}[f|X_u] - f_{U \setminus u} = \int f(\vec{x}_U) p(\vec{x}_{U \setminus u}) d\vec{x}_{U \setminus u} - \sum_{m \subsetneq u} f_m(\vec{x}_m)$$

The sum of the functions $f_m, m \subset u$ corresponds to an adjusted function which has only the parameters $\vec{x}_u$, and the summand f_u represents the effect of all parameters $\vec{x}_u$ together without the effect of subsets of these parameters. Therefore the variance of f_u, which is denoted by the Sobol variance D_u, estimates the common impact of the parameters $\vec{x}_u$. [SGL15] The first-order Sobol index S_u is calculated by normalizing the Sobol variance so that it lies within $[0, 1]$; to this end, it is divided by the function's variance, which is the same as the sum of all Sobol variances $D_u, u \subset U$. The total-order

Sobol index S_u^T contains the variances of all combinations of uncertain parameters which include the parameters $\vec{x}_u$; it is the sum of all S_m where m is a subset of u.

$$D_u = \int f_u(\vec{x}_u)^2 p(\vec{x}_u) d\vec{x}_u = \int \left(\int f(\vec{x}) p(\vec{x}_{U \setminus u}) d\vec{x}_{U \setminus u} \right)^2 p(\vec{x}_u) d\vec{x}_u - \sum_{m \subsetneq u} D_m$$

$$D_u^T = \sum_{u \subset m \subset U} D_m$$

$$S_u = \frac{D_u}{D_U} = \frac{D_u}{Var[f(\vec{x})]}$$

$$S_u^T = \sum_{u \subset m \subset U} S_m = \frac{D_u^T}{Var[f(\vec{x})]}$$

After a polynomial chaos expansion, it is possible to calculate the approximate Sobol variances with the coefficients, which is similar to the calculation of the variance of the PCE approximation. The formula sums up the squared coefficients of the polynomials which have zero degree in all dimensions which are not in u and a positive degree in all the other dimensions.

$$A_u = \{\vec{k} \in \mathbb{N}_0^d \mid \forall i \in U : k_i > 0 \Leftrightarrow i \in u\}$$

$$D_u = \sum_{\vec{k} \in A_u} a_{\vec{k}}^2 \langle \phi_{\vec{k}}, \phi_{\vec{k}} \rangle_p$$

When calculating the total-order Sobol index with the PCE coefficients, the set of coefficient indices for the sobol variances is $\{\vec{k} \in \mathbb{N}_0^d \mid \forall i \in u : k_i > 0\}$ instead of A_u. [al15]
The total order indices are generally more suited to estimate the parameter's importantances. If the function has a big number of uncertain parameters, these Sobol indices help to make a decision on which parameter can be replaced with a fixed value so that, for instance, an accurate expectation computation takes less time.

2.2. Sparse Grids

Sparse grids constitute a mitigation against the so-called curse of dimensionality, which is a common problem in numerical methods. In comparison to Monte Carlo sampling or similar methods, they retain a high convergence rate. In particular, when comparing sparse and full grids and N is the maximum number of grid points in one dimension, standard sparse grids require only $O(N \log(N)^{d-1})$ instead of $O(N^d)$ function evaluations. [BG04] The Monte Carlo method does not rely on the creation of a grid, so it is exempt from the curse of dimensionality; however, its accuracy is in

$O(\frac{1}{\sqrt{M}})$ for M function evaluations. Sparse grids have many application areas, such as interpolation, quadrature and solving partial differential equations.

In this thesis, we employ sparse grids only for quadrature. For simplicity, let us assume that we use composite trapezoidal, d-dimensional quadrature. Let us first define the hat function and full grids. The mother hat function $\phi(\vec{x})$ is defined as follows:

$$\phi(\vec{x}) = \prod_{i=1}^{d} \max\{1 - |x_i|, 0\}$$

A grid defines how a multi-dimensional function input is discretised for numerical quadrature. A full grid is a tensor product of single-dimensional point lists. The full grids considered in this section have equidistant points in the single dimensions, so the full grid of level $\vec{l} \in \mathbb{N}_0^d$ is a rectangular grid that has $2^{l_i} + 1$ points in the i-th dimension; for example, a full grid of level $(2, 3)$ is a 5x9 grid. A point of a full grid can be indexed with a vector $\vec{i}$ which fulfils $\vec{i} \in \mathbb{N}_0^d \wedge \forall k \in [d] : 0 \leq i_k \leq 2^{l_k}$. To obtain a basis for the full grid, a transformed version of the hat function is set as basis function for each grid point. The set which contains all the transformed functions spans the nodal space $V_{\vec{l}}$ for this grid.

$$\phi_{\vec{l},\vec{i}}(x_1, \ldots, x_d) = \phi(x_1 2^{l_1} - i_1, \ldots, x_d 2^{l_d} - i_d)$$
$$V_{\vec{l}} = span\{\phi_{\vec{l},\vec{i}} \mid \vec{i} \in \mathbb{N}_0^d \wedge \forall k \in [d] : 0 \leq i_k \leq 2^{l_k}\}$$

With a given full grid, the quadrature result is the sum of the function values at each point multiplied with the volume of the corresponding $\phi_{\vec{l},\vec{i}}$ function; with homogeneous boundaries this volume is the same for each grid point as it depends only on $\vec{l}$.

To define a sparse grid, we firstly decompose the full grid into disjoint hierarchical increment spaces:

$$W_{\vec{l}} = span\{\phi_{\vec{l},\vec{i}} \mid \forall k \in [d] : i_k \in \{0, \ldots, 2^{l_k}\} \wedge i_k \text{ is odd}\}$$

A sparse grid of level $lmax$ uses a specific subset of points of the level $\vec{l} = (lmax, lmax, ..., lmax)$ full grid. Whereas the full grid space $V_{\vec{l}}$ is the combination of all $W_{\vec{k}}, \forall i : k_i <= lmax$, the standard sparse grid uses only the grid points where the transformed hat function's volume is below a threshold which minimizes the mean squared error:

$$V_{\vec{l}} = \oplus_{|\vec{k}|_\infty \leq lmax} W_{\vec{k}}$$
$$V_{\vec{l}}^{sparse} = \oplus_{|\vec{k}|_1 \leq lmax+d-1} W_{\vec{k}}$$

Figure 2.1 shows hierarchical and non-hierarchical bases and the combination of the hierarchical bases to obtain a full or sparse grid.

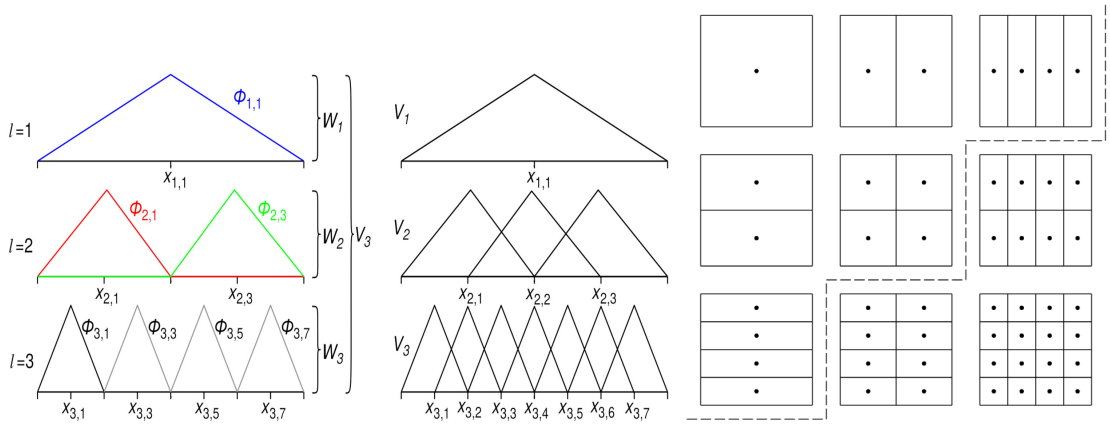


Figure 2.1.: Basis functions for the composite trapezoidal rule without boundary points;
left: single-dimensional hierarchical and non-hierarchical bases;
right: two-dimensional hierarchical bases, the sparse grid uses only the bases above the dashed line.
The images are originally taken from [Bad17]

The calculation of an integral with the hierarchical basis requires a different formula than the calculation with the non-hierarchical basis because the grid points do not all belong to the same level, which results in a strong overlapping of the basis functions with basis functions of neighbouring points. Instead of the function values of the respective grid points, the difference to the function value of the parent point is used. Let $u_{\vec{l},\vec{i}}$ be the function value at the grid point indexed by $\vec{i}$ in the full grid of level $\vec{l}$, and let the surplus $\hat{u}_{\vec{l},\vec{i}}$ be this difference. In the single-dimensional case, $\hat{u}_{\vec{l},\vec{i}}$ can be calculated with a formula which is similar to this one:

$$\hat{u}_{l,i} = u_{l,i} - 0.5(u_{parent\_left} + u_{parent\_right})$$

The left and right parents are the points with level $l-1$ that have the lowest distance to point i with level l. With multiple dimensions, the formula is applied in the direction of each dimension one after another; here is the two-dimensional case:

$$u^1_{(l1,l2),(i1,i2)} = u_{(l1,l2),(i1,i2)} - 0.5(u_{(l1-1,l2),(i\_left,i2)} + u_{(l1-1,l2),(i\_right,i2)})$$
$$\hat{u}_{(l1,l2),(i1,i2)} = u^1_{(l1,l2),(i1,i2)} - 0.5(u^1_{(l1,l2-1),(i1,i\_below)} + u^1_{(l1,l2-1),(i1,i\_above)})$$

These formulas are very simplificated because they ignore boundaries and only work for the linear hat basis function. In the numerical quadrature sum, the surplus is multiplied with the volume of the transformed hat function. When the boundaries are

not zero, we can either add boundary points to the initial sparse grid or use a modified hat function, which is explained in [Pfl10]. Sparse grids quadrature also works with other basis functions. Detailed information on sparse grids can be found in [BG04].

2.2.1. Combination Scheme

We can construct a sparse grid by linearly combining specific full grids, which are denoted component grids [GSZ92]. To calculate the final quadrature solution I, it is sufficient to sum up the solutions of these component grids $v_{\vec{l}}$ multiplied with a coefficient.

$$I = \sum_{q=0}^{d-1} (-1)^q \binom{d-1}{q} \sum_{|\vec{k}|_1 = lmax + (lmin(d-1)) - q} v_{\vec{k}}$$

The coefficient leads to the removal of duplicate points in the resulting sparse grid; it is one for the solutions with the highest levels and, depending on the dimension, positive or negative for lower level component grids. Figure 2.2 shows a combination scheme example with boundary points, composite trapezoidal rule and $lmax$ set to three. Since for quadrature the solutions of the full grids are the same with hierarchical and non-hierarchical basis, we can use the non-hierarchical basis for simplicity. The combination scheme has several advantages over directly calculating a sparse grid. An algorithm which employs the combination scheme can use simple arrays for the full grids, which can lead to straightforward and efficient code. It can calculate the full grid solutions independently, which makes parallelization possible. Furthermore, methods which are specialized for full grids can be applied on the component grids.
In general the combination technique leads to different results than the direct sparse grid approximation but the solutions should be qualitively similar. For quadrature both representations are equal.

2.2.2. Single-Dimension Spatially Adaptive Refinement Method

The topic of this Bachelor Thesis is the application of the single-dimension spatially adaptive refinement method to solve UQ problems efficiently. A detailed explanation of this refinement method can be found in [Möl18]. The method is based on the standard combination scheme. For each dimension the algorithm stores a list of points and their hierarchical basis level; in the beginning these lists are initialized based on the maximum level setting $lmax$ and later the refinement adds or changes points and levels of some lists. After each refinement step, the algorithm constructs a combination scheme. To this end, conceptionally it initializes each component grid with level vector $\vec{l}$ to the tensor product of the points (and levels) lists and then it removes all points whose level vector exceeds $\vec{l}$ in at least one dimension from the component grid. The

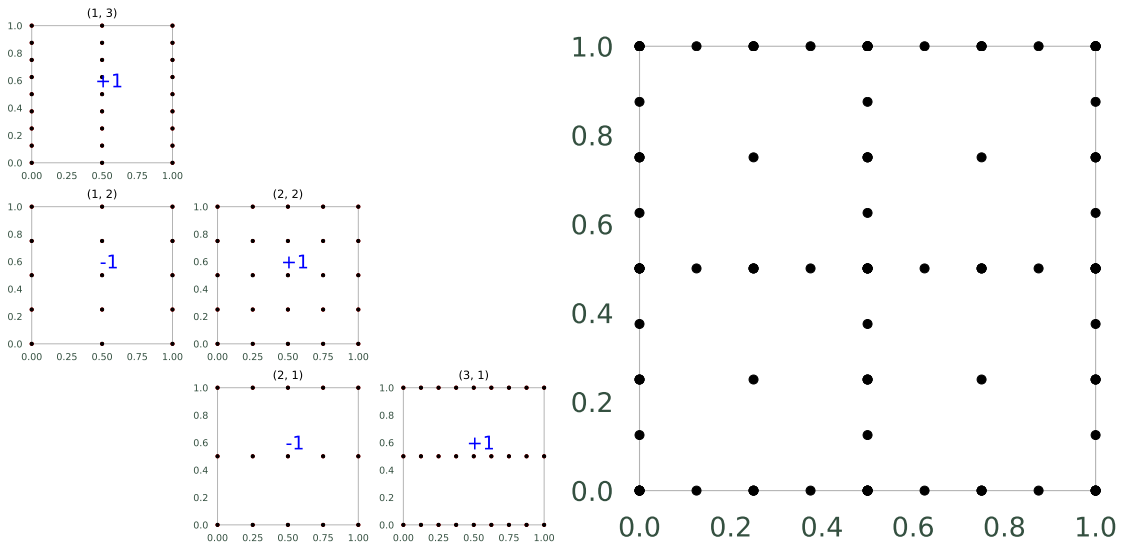


Figure 2.2.: The standard combination scheme with lmax set to three, boundary points and composite trapezoidal grids;
left: the full grids and their coefficients (blue) which are involved in the standard combination scheme; right: the resulting sparse grid

maximum level of this combination scheme is the vector which contains the maxima of the individual dimension's level lists. An example of the adaptive refinement is depicted in Figure 2.3. The adaptive sparse grid refinement happens in single dimensions. To refine the grid, the algorithm adds a new point and its corresponding level into a point list. In the implementation, instead of having a list of points, the list contains abstract refinement objects, which constitute the edges between two neighbouring points; this means that a refinement object stores two points and their levels from the conceptual point list. To make a decision on where to add a new point, the algorithm calculates errors for the grid points and propagates them to the two refinement objects which contain the point. The volume-guided error for a grid point is the hierarchical surplus value, which was denoted $\hat{u}_{\vec{l},\vec{i}}$ in a previous subsection, multiplied with the volume of its hat function $\phi_{\vec{l},\vec{i}}$. The refinement objects whose error is above a specified tolerance are split into half, i.e. the middle between two neighbouring points is added as a new point to the conceptional point list. The new point's level is the maximum of the neighbouring points' levels plus one. The initial grid points and levels may not be distributed optimally, so newly added points may be more important than points with a lower level. To compensate for this, the algorithm has a rebalancing strategy which can change the grid point levels during the refinement. Since the adaptive refinement

method is based on the sparse grid combination scheme, the quadrature weights are calculated in the single dimensions for each component grid. To obtain a result integral, the program sums up component grid solutions, as explained in the previous section. The quadrature points and weights of a component grid are defined by the tensor product of the one-dimensional points and weights.

2.3. Related Work

There exist many papers about sparse grids and uncertainty quantification. For instance, "An overview of uncertainty quantification techniques with application to oceanic and oil-spill simulations" [al15] gives information about PCE coefficient calculation for various applications. The master thesis "Dimension-adaptive sparse grid for industrial applications using Sobol variances" [SGL15] presents the utilization of a dimension-adaptive sparse grid method to uncertainty quantification, so it can be considered a highly related work. The method employs the Clenshaw Curtis quadrature rule, or Fejer when boundaries are omitted, for the grid point positions in the combination scheme, and for the basis functions it uses Lagrange polynomials. The dimension-wise refinement resembles the well-known Gerstner & Griebel refinement algorithm [GG03]. It increments an existing component grid's level in one dimension and adds the component grid with the new level into a forward neighbour list if it does not yet exist in the current combination scheme. Instead of calculating which forward neighbours lead to a good refinement decision, the algorithm from the master thesis calculates Sobol indices for the current sparse grid approximation, sorts these indices, and then adds the forward neighbours where in the respective dimensions the sobol indices are above a threshold. This means that in comparison to the Gerstner & Griebel approach, there are no error estimation calculations for the forward neighbours because the algorithm makes its refinement decisions with only the current grids. The sobol indices which are considered for the refinement decisions depend on the component grids' levels.

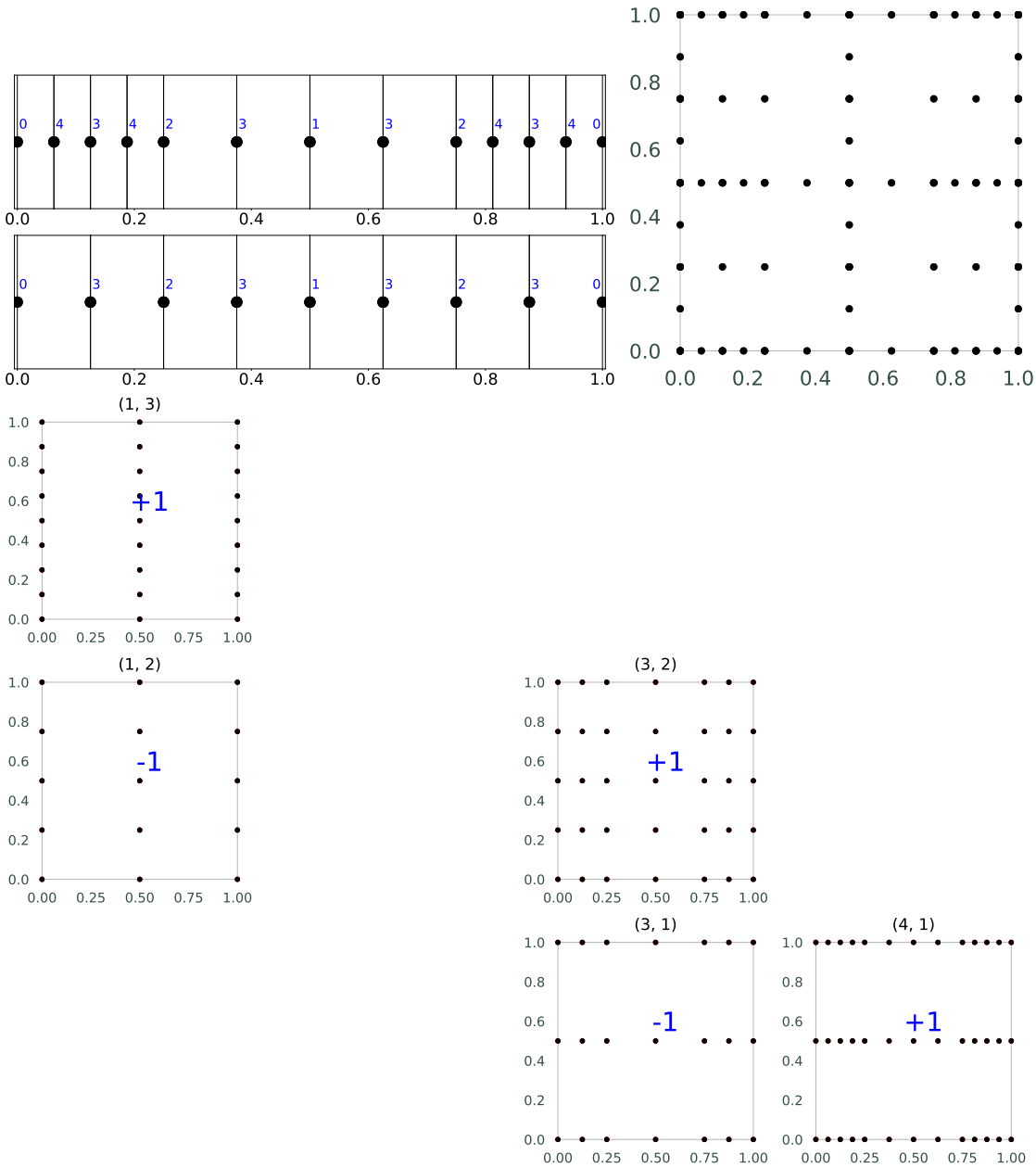


Figure 2.3.: An example combination scheme for the dimension adaptive refinement with boundary points;
top left: the grid points in the single dimensions;
top right: the resulting sparse grid;
bottom: the combination scheme for the single dimensional points and levels

3. Spatially Adaptive Refinement for Uncertainty Quantification

Applying the single dimension spatially adaptive refinement strategy for uncertainty quantification purposes is similar to quadrature. In addition to a function to be integrated, the refinement method has to consider a weight function, which is the joint PDF of the uncertain parameters. Since the parameters are independently distributed, and the refinements, quadrature weights and nodes calculation happen in single dimensions, the program mostly uses the marginal probability density functions. It calculates the quadrature weights for the individual dimensions and then multiplies the weights in a tensor product.

This chapter firstly describes how the quadrature weight and grid refinement differ from non-weighted quadrature by explaining how the program calculates trapezoidal quadrature weights. After that it describes how the algorithm performs the refinement when it needs to calculate multiple integrals at once. In the end, and in the next chapter, the weight calculation for higher order quadrature rules is discussed.

3.1. Composite Trapezoidal Quadrature Weight Calculation

As already mentioned, the quadrature weights are calculated for single-dimensional point lists, which are used by the adaptive refinement method to generate the combination scheme component grids. To explain the weight calculation, this section firstly describes the concept of common composite trapezoidal quadrature.

A simple composite trapezoidal quadrature assumes equidistant points including the boundaries and a weight function which is 1 everywhere inside the boundaries; the quadrature weights are then $\frac{h}{2}$ for the two boundary points and h for all the other points, where h is the distance between two neighbouring points. Conceptually, for the composite quadrature rule, the whole interval is partitioned into subintervals bounded by neighbouring points. For each subinterval, trapezoidal quadrature weights are assigned to the boundaries so that a linear function can be integrated exactly within this subinterval. Since the weight function is 1, both trapezoidal weights are $\frac{h}{2}$. The composite quadrature weights mentioned before are the trapezoidal weights, where

due to the overlapping of the subinterval boundaries, for an inner point the composite weight is the sum of two trapezoidal weights.

In our case, the points are not equidistant, the weight function is a PDF, and the boundaries can be excluded from the quadrature points. Trapezoidal quadrature calculates polynomials up to degree one exactly. Therefore, when the weight function is a PDF p, following conditions must hold for a subinterval $[x_1, x_2]$ and weights w_1 and w_2:

$$1w_1 + 1w_2 = \int_{x_1}^{x_2} 1p(x)dx =: m_0$$
$$x_1w_1 + x_2w_2 = \int_{x_1}^{x_2} xp(x)dx =: m_1$$

The integrals m_0 and m_1 on the right are denoted moments and calculating the weights by fulfilling these conditions is called the method of undetermined coefficients [Hea02]. The weights calculation happens for a single dimension, so it does not suffer from the curse of dimensionality. When the moments are known, w_1 and w_2 are the values of interest. One can show that these can be calculated as follows:

$$w_2 = \frac{m_1 - m_0x_1}{x_2 - x_1}$$
$$w_1 = m_0 - w_2$$

The composite quadrature weight calculation works like the simple composite trapezoidal quadrature mentioned before. The calculation of the moments is explained in section 4.2.3.

Some probability distributions are defined on infinite boundaries, for example the normal distribution. Due to the initialisation of the component grids, the list of points always has at least three entries, so x_1 and x_2 cannot both be infinite at the same time. If one of them is infinite, the aforementioned formula cannot calculate w_2. After applying L'Hospital to calculate the limit, w_2 is m_0 for an infinite x_1, respectively 0 for an infinite x_2. This means that for infinite boundaries the weights are always zero at the boundary points; the trapezoidal rule in this subinterval integrates only constant functions.

The problem function should usually not be evaluated on infinite boundary points or far boundary points where the probabilities or the function are very small. Therefore the sparse grid refinement scheme allows excluding the boundary points, which leads to significantly fewer points in the resulting sparse grid. When the omitted boundaries are finite, the algorithm needs to normalize the relevant quadrature weights so that their sum is one.

3.2. Grid Refinement

The single dimension adaptive refinement strategy refines the grid indirectly by splitting a refinement object into two new ones. Since the points are weighted with a probability distribution, the refinement objects are no longer split in the spatial middle, but at a point which makes the two new ones have the same probability. For this purpose, the positions are normalized and denormalized with the cumulative distribution function (CDF) and percent-point function (PPF), which is the inverse CDF. When the refinement object's points are x_1 and x_2, the new refinement objects have the points x_1 and mid, and mid and x_2, where mid is calculated with this formula:

$$ppf = cdf^{-1}$$
$$mid = ppf(\frac{cdf(a) + cdf(b)}{2})$$

Figure 3.1 depicts an example of refinement objects and the corresponding sparse grid with points which are located based on a triangle probability distribution.
In the surplus volume error calculation, the distance between a point's two parents x_1 and x_2 can no longer be used for the width of the pagoda volume because it does not incorporate the probability density; therefore for weighted integration, the width is set to $cdf(x_2) - cdf(x_1)$.

3.3. Refined Functions

The weight function in the adaptive refinement is always the PDF. In addition, the user chooses the function for which the grid is refined depending on what he/she wants to calculate. Except for the expectation, the function in the weighted integral is not the problem function. For instance, when calculating the second moment, which is the integral of the squared function weighted with the PDF, the function for which the grid needs to be refined is the squared problem function.

The aim of the adaptive refinement is minimizing the overall number of evaluations of the problem function. For this reason, it is also possible to refine for multiple functions at once. This is required when the user is interested in multiple moments and when calculating the pseudospectral PCE coefficients, where the problem function is multiplied with each individual polynomial. To this end, these functions are merged into one multidimensional function, which is then adaptively refined. The refinement should treat each output dimension as equally important. Therefore the error err which is used for the refinement decision is a user-defined norm $norm$ of the errors $\vec{e}$ for each

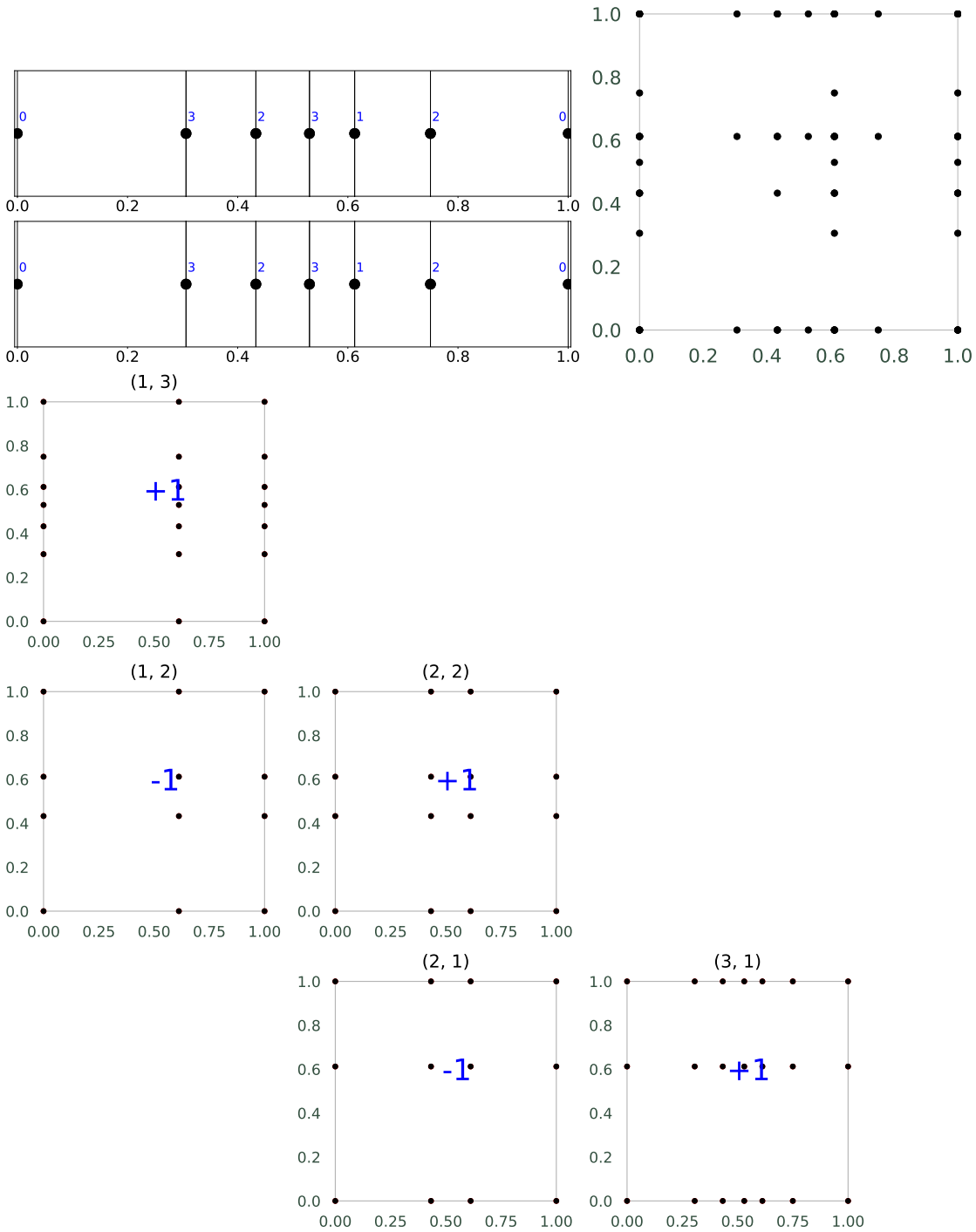


Figure 3.1.: An example for the dimension adaptive refinement with grid points calculated for a triangle probability distribution;
the images are arranged as in Figure 2.3

dimension normalized by the previous integral solution $\vec{s}$:

$$e_{normalized,i} = \begin{cases} \frac{e_i}{s_i} & s_i \neq 0 \\ e_i & \text{otherwise} \end{cases}$$
$$err = norm(\vec{e}_{normalized})$$

A frequent UQ-related task is the expectation and variance calculation. If the expectation was calculated independently in the refinement procedure, the refinement would happen two times or would not be optimal for both values. Therefore, the variance is calculated from the second moment and the first moment, which is the expectation.

$$Var[w] = \mathbb{E}[w^2] - \mathbb{E}[w]^2$$

In this case the program adaptively refines for the first and second moment at once, so the refined function has a two-dimensional output if the problem function's output is scalar.

3.4. Higher Order Quadrature

In addition to the weighted trapezoidal grid, weighted versions of grids for higher order quadrature rules have been implemented; these are explained in section 4.2.3 and section 4.2.3. The addition of weigthed grids is not the only way to obtain high order weights. When the distributions are uniform, an algorithm can use integration without weighting and then multiply the result integral with the non-zero joint distribution's constant value, which is the inverse of the volume of the integration domain $\prod_{d=0}^{dim}(b_d - a_d)$, where $\vec{a}$ and $\vec{b}$ are the domain boundaries. Furthermore it is possible to employ inverse transform sampling, where the single-dimensional distributions' PPFs are applied to the input arguments before passing them to the problem function and each PPF input argument is uniformly distributed in $[0, 1]$. The inverse transformation method is generally less accurate.

4. Implementation in Python

The support for uncertainty quantification is implemented in an existing sparse grid framework. The first section describes the structure of this framework, which constitutes the background information for the following section. The second section then covers the implementation of the UQ Operation and details on the quadrature weight calculation.

4.1. sparseSpACE Framework

The sparseSpACE apronym stands for The Sparse Grid Spatially Adaptive Combination Environment. [Obe] This framework offers several spatially adaptive sparse grid refinement strategies, for example the cell and the extend-split strategies; in this thesis only the single dimension strategy is employed.
Sparse grids can be based on several different grid types, such as the composite trapezoidal grid or a Clenshaw-Curtis grid. The functionality to select from various grids is implemented in the grid classes; the global grids, for example `GlobalHighOrderGrid` and `GlobalTrapezoidalGrid`, are applicable to the single dimension strategy. These grids have methods for quadrature weight calculation and finding the middle between points, and some helper methods. The `GlobalHighOrderGrid` class contains the implementation of the quadrature weight enhancement algorithm which is discussed in section 4.2.3. Global grids which do not use the linear hat basis function are implemented as subclasses of `GlobalBasisGrid`; they calculate weights for the hierarchical surplus values.
The grid operation classes define what the sparse grid is used for; `Integration`, for instance, has methods for integral calculation and surplus volume error estimation.
Since the aim of sparse grids is to reduce evaluations of expensive functions, the framework has `Function` classes, which cache the problem function's output and thus also keep track of the number of evaluations. The `eval` method calculates the problem function output without caching and it can return a scalar or an array. The `__call__` method is indirectly invoked when calling the class' instance. It uses the cache and always returns an array so that the adaptive refinement does not need to treat Functions with multidimensional output differently. `Functions` also have methods for calculating reference solutions, determining the output dimension, or showing a function plot.

4.2. Extension for UQ

To support uncertainty quantification, sparseSpACE now has the new grid operation class `UncertaintyQuantification`, which inherits from the `Integration` grid operation. Furthermore, weighted grids were added as subclasses of `GlobalTrapezoidalGrid`, `GlobalHighOrderGrid` and `GlobalLagrangeGrid`. The new grids support a PDF weighting function instead of a constant weight; the calculation of, for example, composite trapezoidal quadrature weights is different, which has been explained in chapter 3.

4.2.1. UncertaintyQuantification Grid Operation

The grid opertation is related to what the user wants to calculate. Since there are many possible calculations in the context of uncertainty quantification, the `UncertaintyQuantification` grid operation offers methods for calculating moments, expectation and variance, sobol indices and more. Its constructor is similar to Integration's constructor; it additionally prepares distribution functions from the user's input. These distributions are created with the help of the python libraries Chaospy and SciPy. For flexibility, the user does not directly pass distribution functions but only the distribution name and parameters for each dimension. UncertaintyQuantification uses Chaospy's pseudo-spectral PCE calculations, e.g. the creation of orthogonal polynomials, which require Chaospy distributions. For the quadrature weights and RefinementObject splitting, the distributions' PDF, CDF and PPF serve as weight functions.

The user needs to pass a `Function` to the spatially adaptive refinement which the refinement algorithm evaluates at the grid points for the error estimation. As explained in chapter 3, except for the expectation, the function or functions to be integrated is not the problem function. Therefore dedicated methods return a `Function` which addresses this issue; they are explained in detail in the following subsections.

After an adaptive refinement has been performed, the user can invoke methods to calculate a moment, variance, sobol indices, and more. The user may have selected a function for the refinement whose integral is not the value which should be calculated in the end; in this case he/she has to pass a specific argument to the method so that it uses the generated quadrature points and weights and not directly the integral from the refinement. Before invoking methods which use the PCE approximation, the user has to call the `calculate_PCE` method. It creates the PCE proxy function with Chaospy and accepts a boolean argument to restrict the polynomial degrees in the dimensions where the adaptive refinement generated only a few points. The method uses orthogonal polynomials and norms returned by Chaospy's `orth_ttr` function. If the method can use the integral from the refinement, it creates a Chaospy polynomial

directly with the integral results and polynomial norms. If this is not the case, the method passes quadrature points and weigths and the polynomial norms to Chaospy's `fit_quadrature` function. If the norms were not passed, Chaospy would calculate the polynomial norms with the quadrature points and weights, which can be inaccurate.

4.2.2. Additional Function Classes

This section describes the `Function` subclasses for the common adaptive refining, which are returned by `UncertaintyQuantification`'s methods, and then it explains `Functions` which mainly have testing purposes.

Concatenation

The `FunctionConcatenate` class concatenates the return values of several `Functions` when evaluated, which means that it is suited to merge multidimensional functions to a new higher-dimensional function. This higher-dimensional function may be passed to the adaptive refinement algorithm so that it calculates the grid points for the integral calculations of all of the functions which have been merged. Additionally, with `FunctionConcatenate`, the user can transform his/her problem function(s) so that it returns a flat array and the adaptive refinement accepts it.

Moment Calculation

The grid operation's `get_moment_Function` method returns an instance of the `FunctionPower` class, except when calculating the first moment. This `Function` is designed for the refinement for the k-th moment calculation. The `eval` method of this class first calls the problem `Function`, whose cached result is reused when possible, and then it takes each value in the returned array to the power of `k`; in the end it returns this new array. `UncertaintyQuantification` also has a method to obtain a `Function` for the simultaneous refinement of multiple moments: `get_moments_Function`. This method creates instances of `FunctionPower` for each moment and then it uses `FunctionConcatenate` to merge them together. If this function is passed to the adaptive refinement, the resulting sparse grid should be suitable for the calculation of all of the requested moments.

PCE Coefficient Function

For the pseudo-spectral PCE coefficient calculation, the grid operation's `get_PCE_Function` method returns an instance of `FunctionPolysPCE`. The constructor of this class takes a `Function`, Chaospy polynomials and their norms. Its `eval`

method multiplies the evaluation of the problem function with evaluations of all polynomials and then merges the results into a flat array, so for a m-dimensional function and k polynomials, it returns an array with $k \cdot m$ entries.

Other Functions

Some new `Functions` are for testing or to help the user with specific tasks. The `FunctionCustom` class takes one or multiple python functions in its constructor and calls them when evaluated. With this class the user can convert his/her problem function to be compatible with `Function` and does not have to create a new subclass. An example for a test function is the G-function; it is explained in detail in the next chapter.

4.2.3. Weighted Global Grids

The UQ operation supports a weighted trapezoidal, HighOrder and Lagrange grid, and, if there are only uniform distributions, other basis grids such as the B-Spline grid. This subsection gives implementation-related information about the refinement object splitting, which is currently the same for all weighted grids. Then it describes the implementation of quadrature weight calculation for the weighted trapezoidal grid and other high order grids.

Refinement Object Splitting

As explained in the section 3.2, during the adaptive refinement, the `RefinementObjects` need to be divided to obtain a new grid point and two new `RefinementObjects`. The information about where and how points are located in the grid should belong to the `Grid` object and not to an abstract refinement object or container. Therefore, a method of the employed global grid performs the calculation of the weighted middle between two points; it uses the method shown in Figure 4.1. In some situations, the CDF and its inverse are not accurate enough to calculate the middle between points, for instance when the points are very close to each other. If this problem occurs, the method falls back to a simple non-weighted middle calculation; if one of the two points is infinite, the other point offset by a small constant is used to avoid a crash.

To make a decision on where to refine, the program calculates volume-based surplus errors for each refinement object. As explained in chapter 3, when the function's output is multidimensional, the program divides the errors by the previous integral solutions. This only works if the exact integral solution is not zero. Since the exact solution is not known beforehand, the volume weight calculation tests if it is near

```
@staticmethod
def get_middle_weighted(a, b, cdf, ppf):
        cdf_mid = 0.5 * (cdf(a) + cdf(b))
        mid = ppf(cdf_mid)
        if not a < mid < b:
                print("Could not calculate the weighted middle")
                mid = 0.5 * (a + b)
                if not a < mid < b:
                        print("Warning: Could not calculate the "
                                f"middle between {a} and {b}")
                        if isinf(a):
                                mid = b - 10 ** -14
                        elif isinf(b):
                                mid = a + 10 ** -14
        return mid
```

Figure 4.1.: Refinement object middle calculation

zero with a tolerance of for example 10^{-10}. If this tolerance is too big, the refinement can overlook the function output dimension where the exact integral is small but not zero, and if it is too small, the overall error estimation can show huge values and the algorithm tries to refine for the output dimension where the exact integral is zero. The volume normalization may make it difficult for the user to abort the refinement when a specified tolerance has been met; for instance, with `FunctionPolysPCE`, the error refers to the relative errors of the gPCE polynomial coefficents and not to the error of the expectation or another value of interest.

Weighted Global Trapezoidal Grid

The `GlobalTrapezoidalGridWeighted` grid implements the weighted composite trapezoidal quadrature explained in section 3.1; its `compute_weights` method performs the quadrature weights calculation. The formula requires the moments m_0 and m_1 in each subinterval. The program calculates the zeroth moment with the CDF, and for the the first moment it calls SciPy's numerical quadrature function `integrate.quad` because the distribution functions from Chaospy or SciPy do not offer a method which calculate the first moment between specified boundaries, so m_0 is generally more accurate than m_1. Figure 4.2 shows Python code for the trapezoidal weight calculation. The actual code in the implementation is slightly different: it reuses previously calculated moments to increase the performance.

```
moment_0 = cdf(x2) - cdf(x1)
moment_1 = integrate.quad(lambda x: x * pdf(x), x1, x2,
        epsrel=10 ** -2, epsabs=np.inf)[0]
if math.isinf(x1):
        w2 = moment_0
elif math.isinf(x2):
        w2 = 0
else:
        w2 = (moment_1 - moment_0 * x1) / (x2 - x1)
w1 = moment_0 - w2
```

Figure 4.2.: Trapezoidal weight calculation

After calculating the composite weights, the method sets negative near-zero weights to zero; these weights sometimes occur due to numerical errors. Furthermore, it adjusts the weights when boundaries are excluded, and tests if all weights sum up to approximately one.

Weighted HighOrder Grid

Composite trapezoidal quadrature can only integrate functions up to degree one exactly. However, given n arbitrary placed distinct points, it is possible to calculate a quadrature rule with an order of at least n with only positive weights. The method described in [Huy09] is implemented in the `GlobalHighOrderGrid` class; the `GlobalHighOrderGridWeighted` extends this class for weighted integration. This quadrature method is denoted HighOrder in the next sections and chapters. It calculates higher order weights from start values of weights and points, which can be obtained with, for example, the composite trapezoidal quadrature explained in section 3.1. The algorithm creates discrete orthogonal polynomials, calculates their moments, and then uses these moments to compute the new weights. The moments are the integrals of the polynomials multiplied with the weight function, which is the PDF here. In comparison to integration without a weight function, the numerical moment calculation uses Gauß quadrature points and weights which are tailored to the distribution. Since the method calculates the moments over the whole distribution's domain, Chaospy's `generate_quadrature` function can return these points and weights. If, however, at least one of the composite trapezoidal weights is zero, and the zero weight cannot be truncated away, i.e. it does not belong to the boundary points, the polynomial calculation does not work and the weight calculation returns the trapezoidal weights.

Weighted Basis Grids

Another possibility to increase the quadrature order is the use of different basis functions, which is implemented in the `GlobalBasisGrid` subclasses.
The `GlobalLagrangeGridWeighted` grid has constrained Lagrange polynomials as basis functions, which need to be integrated over subintervals with a weight function to obtain quadrature weights. When the probability distribution is uniform, Gauß Legendre points and weights can be shifted and scaled into the integral boundaries to obtain an exact integral. This transformation does not work with, for example, the normal distribution and Gauß Hermite quadrature, so the weights calculation currently uses SciPy's numerical integration, which is slow. Furthermore, the numerical integration errors are directly passed on to the final grid point weights.
The global B-Spline basis grid calculates Spline knots outside the domain boundaries; these points cannot be transformed for a probability distribution, so there is currently no weighted global B-Spline grid subclass.

5. Test Results

This chapter presents test results for Lotka-Volterra predator-prey model test cases and functions with discontinuities. One of the predator-prey model tests is designed to obtain solutions for all time steps at once, which requires the weighted integration of a vector-valued function. Another test calculates only the sheep population in a single time step, which should be easier to refine adaptively. In the G-function tests, the adaptive refinement outperforms Gaußquadrature; the reason for this may be the discontinuity in the derivations of the problem function. In all test cases, if not mentioned otherwise, the maximum level of the initial combination scheme `lmax` is two for uniform distributions and three for normal distributions. Without boundaries and `lmax` set to two, the adaptive refinement would not have points which constitute the interaction between input dimensions; this can result in missing or inaccurate refinement steps. Furthermore, in the tests the HighOrder grid's `max_degree` argument is set to its default value of five, the B-Spline and Lagrange grids, which are mostly used for uniform distributions, have the polynomial degree argument `p` set to its default value of three, the spatially adaptive refinement algorithm version is three, the error norm used on function output values is the L_2 norm, and other configuration parameters which were not mentioned are set to their default values.

5.1. Predator-Prey Model

The predator-prey differential equations describe the problem where for an initial population of predator and prey animals, the population size of both animals changes over time. The prey population is increasing when there are many prey animals, and decreasing when there is a big number of attacking predators. On the other hand the predator population is decreasing when there are a many predator and only a few prey animals, and increasing when a few predators can feed on many prey animals. This leads to a periodic behaviour regarding the number of prey and predator animals. For a predator death rate d, voracity v, augmentation rate a and current population n_{Pred}, and prey birth rate b and current population n_{Prey}, the change in predator and prey populations are as follows:

$$f(n_{Pred}, n_{Prey}, t) = (change_{Pred}, change_{Prey}) = ((an_{Prey} - d)n_{Pred}, (b - vn_{Pred})n_{Prey})$$

An instance of this problem is the sheep-coyote model; here numbers are assigned to the parameters:

$$d = 0.0005,\ b = 0.005,\ v = 0.00012,\ a = 0.002v$$
$$n_{PreyInitial} = 2000,\ n_{PredInitial} = 50$$

To employ this model for an UQ test, some parameters need to be uncertain. A farmer probably knows much about the sheep but has only vague knowledge about wild coyotes. Therefore, let the voracity and initial coyote population be normally distributed.

$$v \sim \mathcal{N}(0.00012, 0.000002^2)$$
$$n_{PredInitial} \sim \mathcal{N}(50, 5^2)$$

Now it is possible to compare the spatially adaptive sparse grid method with conventional integration methods to calculate an expectation, variance and more. The next subsections describe the problem of obtaining a reference solution and the comparison between weighted integration methods.

5.1.1. Reference Solution

We could be interested in how the expected populations of sheep and coyotes develop in time and how reliable these values are. This means that the program needs to evaluate the stochastic values for many time steps. In the implementation the problem function is set to a multidimensional Function which returns a flat array containing the sheep population and predator population for each time step. Two of the test programs calculate the expectation, variance and percentiles for both population sizes with PCE with a maximum polynomial degree of one; this means that in these tests quadrature points and weights are used to calculate the three multidimensional PCE coefficients.

To obtain a reference solution for comparisons with the adaptive sparse grid method, several quadrature methods offered by Chaospy were tested. In a previous test, the initial sheep population was normally distributed, too, and the initial predator population was less uncertain: $n_{PreyInitial} \sim \mathcal{N}(2000, 1.0^2), n_{PredInitial} \sim \mathcal{N}(50, 0.1^2)$. Since the impact of the initial number of prey on the function output was very small, the current test uses a fixed value for this number. In that previous test, the Chaospy Clenshaw-Curtis quadrature results differed significantly between when the polynomials were normed with the quadrature points, and when the norms returned by `cp.orth_ttr` were used; this happened with low and high numbers of quadrature points. With the Gauß quadrature rule, the problem did not occur and the solutions looked more promising. Nonetheless, even with a high numbers of points, the maximum of the

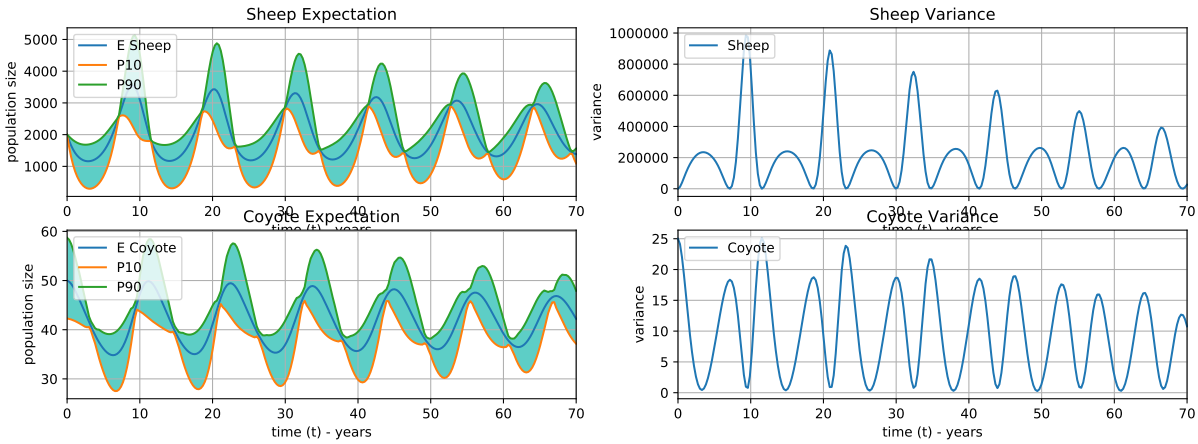


Figure 5.1.: Fullgrid Gauß reference solution for the Predator-Prey model; P10 and P90 are Percentiles

sheep population variance varied between approximately 50000 and 60000 when the number of points changed a bit, and Chaospy's Gauß quadrature works only up to order thirty in this test, probably because negative input values can appear due to the normal distribution. This suggests that finding a precise reference solution could be difficult. The standard Monte Carlo method, tested with 20000 points, and the Monte Carlo method with the Halton sequence resulted in values similar to the Gauß quadrature results.
The Gaußquadrature reference solution for the two-dimensional test can be seen in Figure 5.1. The differential equation cannot be evaluated for negative voracity or initial population values, which can appear because the normal distribution has an infinite boundary domain. A working truncated normal distribution is not yet available in Chaospy, and a uniform distribution would lead to less useful test results because it is almost a non-weighted quadrature. Therefore, the result of Chaospy's full grid Gauß quadrature with the order parameter set to 29 is used as approximate reference solution.

5.1.2. Comparison

In the following, the reference solution is compared to solutions which are obtained with the adaptive refinement. In the vector-valued test and the single time step test, the full grid Gaußquadrature surpasses the adaptive refinement. Since the reference solution is not necessarily very accurate, as explained previously, the deviations may

only be meaningful for a low number of evaluations. On the other hand, with only very few evaluations, the adaptive refinement cannot adapt to the function and thus has more resemblance to a non-adaptive sparse grid. For reliability, the maximum level $lmax$ of the initial combination scheme is set to three.

Refining All Time Steps

The Predator Prey test with 256 time steps is a use case of the adaptive refinement where the PCE coefficient refinement function output has a huge number of dimensions: $256 \cdot 2 \cdot (1 + dim) = 1536$. While the adaptive refinement results look similar to Gaußquadrature results when the number of function evaluations is big, the mean absolute deviation to the reference solution decreases faster with full grid Gaußquadrature. Figure 5.2 shows the function plot and errors for each time step when the refinement uses the weighted global trapezoidal grid with lmax set to three and 157 problem function calls. Figure 5.3 displays the development of the mean absolute deviation for various number of function evaluations. The expectation and variance error plots for the adaptive refinement resemble a staircase; the error is approximately the same for some consecutive numbers of function evaluations and then suddenly drops down. From this we may conclude that the refinement object error estimation is not suited to refine a function with a huge output dimension or that each output dimension could behave like a different function, which would make an adaptive refinement pointless.

Refining for a Single Time Step

Instead of creating quadrature points and weights for all time steps and predator and prey populations, the adaptive refinement can focus only on a single time step and the number of sheep. The problem function in this test firstly evaluates the multidimensional problem function from before, and then it takes only the prey population value of the 25th of 256 time steps from the returned array. Since only two input parameters are uncertain, the function can be plotted as shown in Figure 5.4. The parameters are normally distributed and thus have infinite boundaries; therefore the plots are truncated so that in the single dimensions the cumulative distribution lies within 0.01 and 0.99.

For the reference solution, the test reuses the Gaußquadrature result from before; it picks the prey solutions for the 25th time step from the solutions array. In figure Figure A.1 the relative deviations to the reference solution are plotted. The errors refer to the expectation and variance of the PCE approximation, so the reference variance deviates from the function's variance. The full grid Gaußquadrature still has lower errors than the proposed refinement solutions for a low number of function

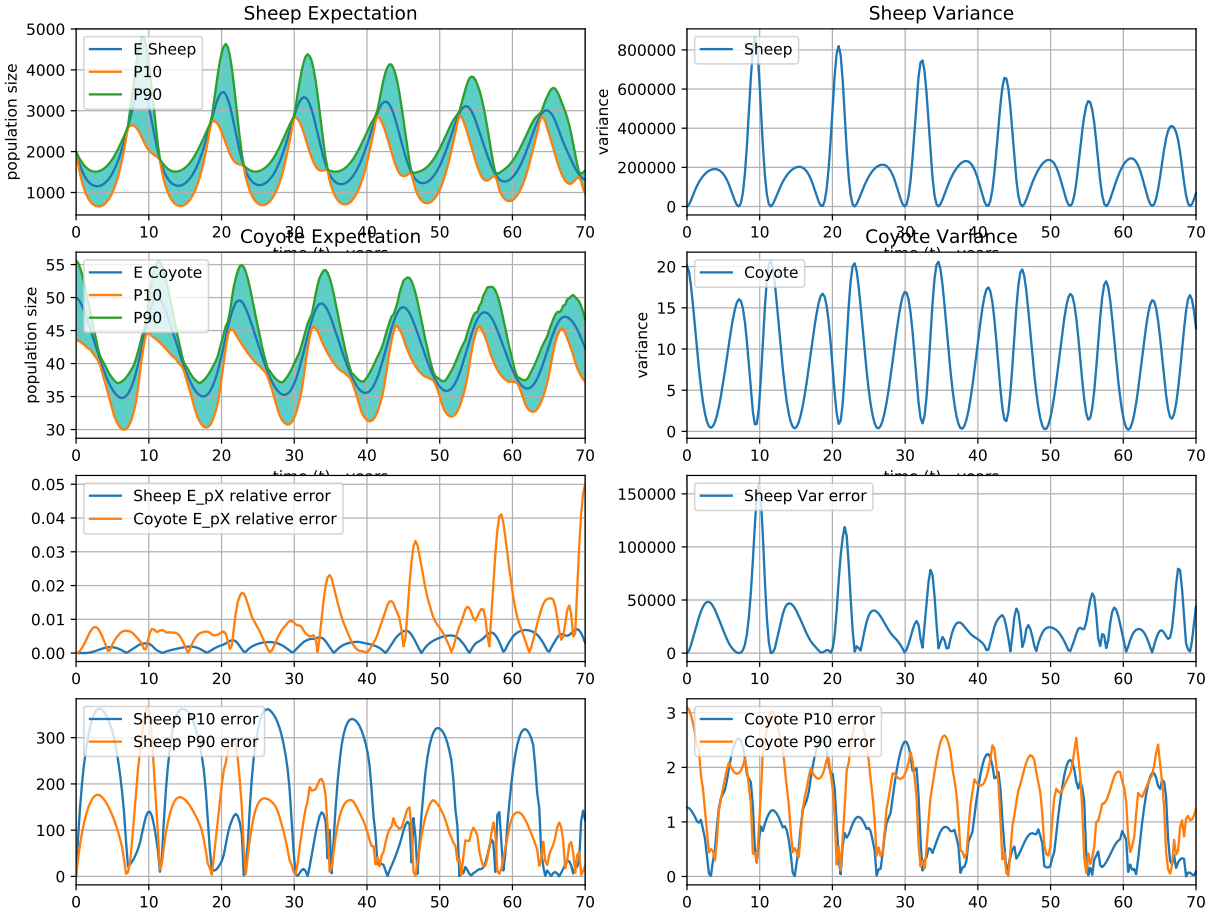


Figure 5.2.: Solution and errors, respectively deviations, for the adaptive refinement with the weighted global trapezoidal grid; lmax=3, 157 evaluations

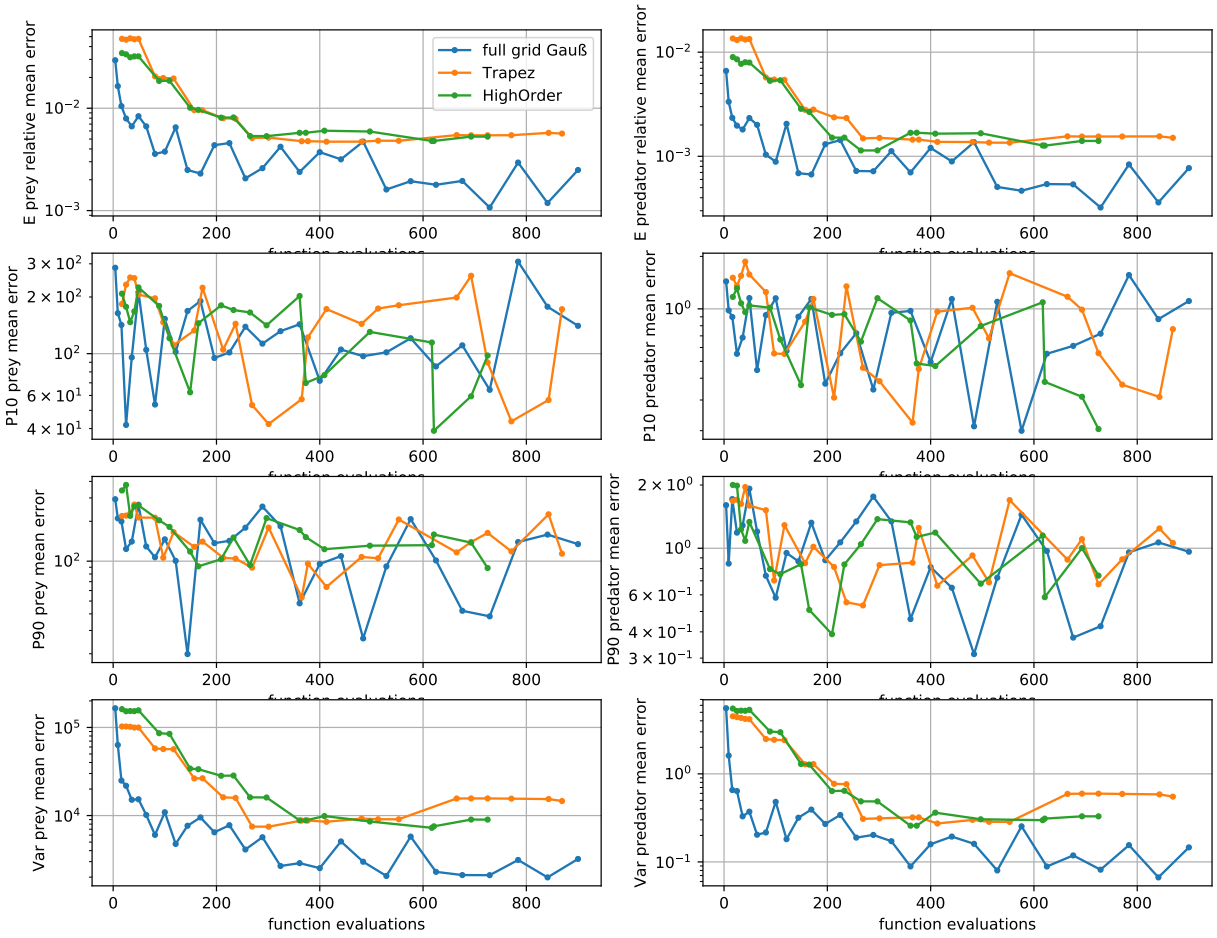


Figure 5.3.: Mean absolute deviations between adaptive refinement and Gauß quadrature results; lmax=3

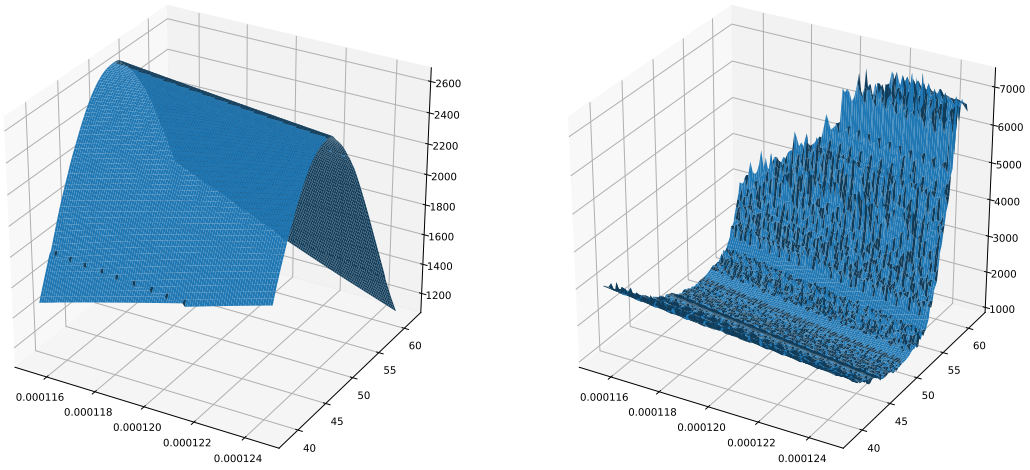


Figure 5.4.: Prey populations in various time steps; left: 25th time step, right: 210th time step

evaluations. In the beginning, the sparse grid Gauß quadrature, where the standard combination scheme uses Gauß Hermite quadrature full grids, seems to give the best results. The Gauß full and sparse grid plots end early because of the occurrence of negative input values, thus the error values for all quadrature methods with many function evaluations may not be meaningful. Nonetheless, the weighted trapezoidal quadrature is probably more accurate than the inverse transform method because it always has a lower expectation error. Chaospy's full grid Fejer quadrature, which is not shown on the plot, has relatively high error values. The Fejer solution could have the problem which occurred with the Clenshaw Curtis quadrature, mentioned in the reference solution subsection, i.e. it may generally give wrong results.

The expectation and variance have also been calculated with the zeroth and first moment for various numbers of function evaluations, as shown on the bottom in Figure A.1. Here the inverse transform has a lower accuracy again. The reference solution is the Chaospy full grid Gauß quadrature solution with the `order` parameter set to `29`; the reference expectation and variance have been calculated with the moments.

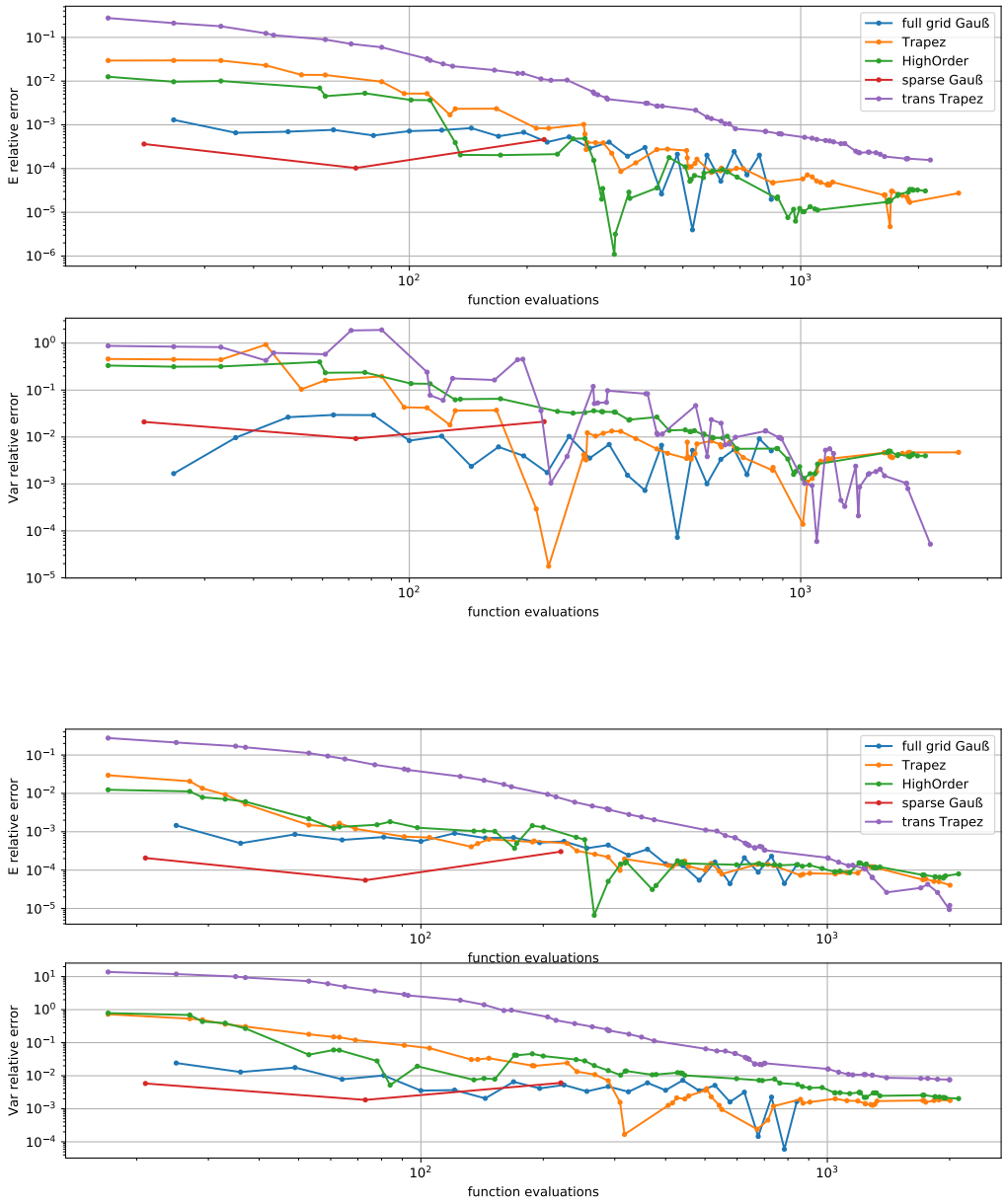


Figure 5.5.: Relative errors for the PCE approximation expectation and variance (top), and expectation and variance calculated with the moments (bottom) of the 25th time step in the sheep coyote problem;
trans Trapez refers to the trapezoidal grid with the inverse transformation method.

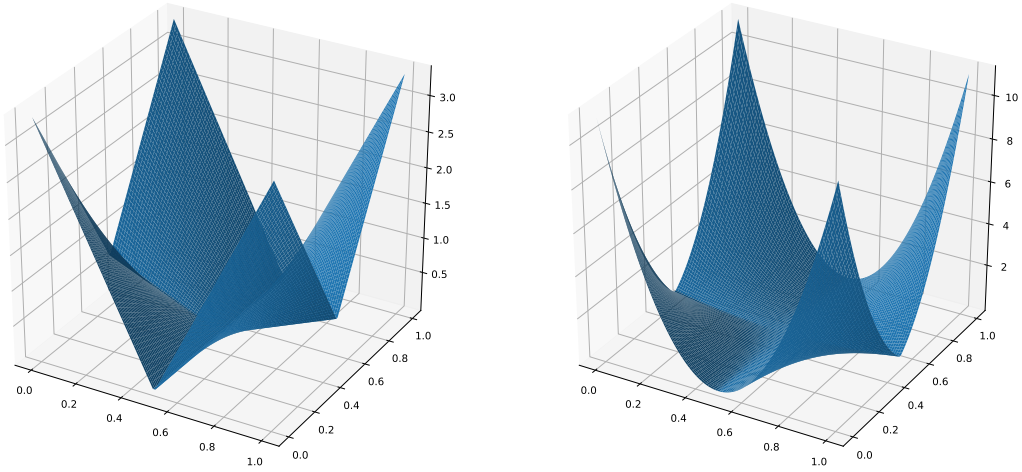


Figure 5.6.: Left: g-function, right: squared g-function (for second moment calculation)

5.2. G-Function

The g-function of Sobol is a test function with analytic solutions. [al08]

$$f(\vec{x}) = \prod_{d=0}^{dim-1} \frac{|4x_d - 2| + a_d}{1 + a_d}$$

We can configure it with the dimensionality dim and positive values a_d, which define a dimension's importance. All input parameters are uniformly distributed in $[0, 1]$. The expectation of this function is always one and the analytic solution of the variance can be calculated with this formula:

$$M_2 = \prod_{d=0}^{dim-1} \left(1 + \frac{1}{3(1 + a_d)^2}\right)$$

$$Var = M_2 - 1$$

To test this function with the adaptive sparse grid, let us assign values to a_d and dim.

$$a_0 = 0,\ a_1 = 0.5,\ dim = 2$$

These values lead to a variance of approximately 0.5309. Function plots can be seen in Figure 5.6.

5.2.1. Without Shifting

When integrating the G-function, the points of the adaptive sparse grid are aligned with the positions where the derivative is not defined. Therefore, this test is not a reliable measure to compare the integration methods. Figure 5.7 shows relative errors of the expectation and variance, which is calculated with the expectation and second moment. The global trapezoidal grid works best for the expectation; this is due to how the function is defined. In the single dimensions it behaves similar to the hat function: the other dimensions contribute only a constant factor, from 0 to 0.5 the function linearly decreases, then it linearly increases, and the first derivation is undefined at 0.5. The variance calculation seems to work better with a higher order quadrature rule. The Lagrange grid performs best for this corner case function; since its weight calculation currently takes a long time, the errors are only plotted for low numbers of function evaluations. The full grid Gauß quadrature, which is implemented by Chaospy's `generate_quadrature` method, has the highest expectation and variance error. Chaospy's sparse grid Gauß quadrature has even higher errors, which is not shown in the error plot; it has been tested by enabling the `sparse` argument.

The g-function is one of the corner cases where without boundary points and the default `lmax` value of two, the adaptive refinement cannot refine correctly at the beginning. This happens because for at least one dimension, when the input value is zero in all other dimensions, all output values of the problem function are zero or have the same value in this dimension. The adaptive refinement calculates wrong integrals when a corner case occurs, thus the maximum level lmax needs to be set to at least three so that the refinement can use enough error values.

5.2.2. Shifted G-Function

To have a test which is not a corner case, the G-function is moved by 0.2 in every dimension. If the moved coordinates are outside $[0,1]^2$, they are placed back to the other side so that the expectation and variance are still the same. This means the shifted G-function g follows this formula:

$$g(x_1, x_2) = f((x_1 + 0.2) \bmod 1,\ (x_2 + 0.2) \bmod 1)$$

The derivation now has more discontinuities, which can be seen on Figure 5.8. The relative errors shown on Figure 5.9 indicate that for a big number of function evaluations the weighted trapezoidal composite quadrature and other adaptive sparse grid methods exceed full and sparse grid Gauß quadrature. The shifted G-function has also been tested with five input dimensions. In this test, the configuration values are set to $a_d = \frac{d}{2}, 0 \leq d < 5$. With boundary points and lmax set to two, the initial sparse grid

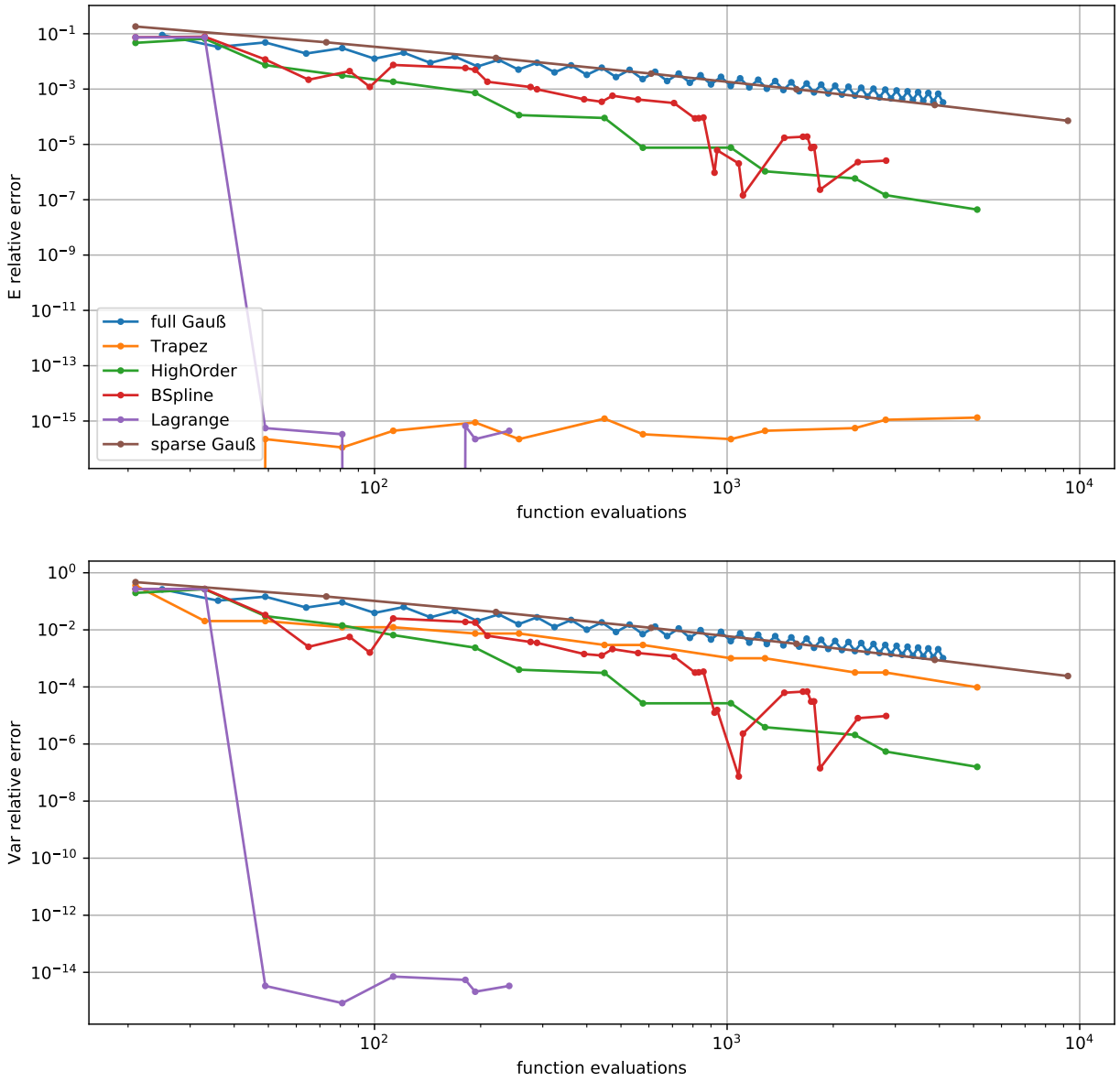


Figure 5.7.: Relative errors for the G-function when employing Chaospy's full grid Gauß quadrature, sparseSpACE's sparse grid Gauß quadrature and adaptive refinement with trapezoidal, HighOrder, B-Spline and Lagrange grids

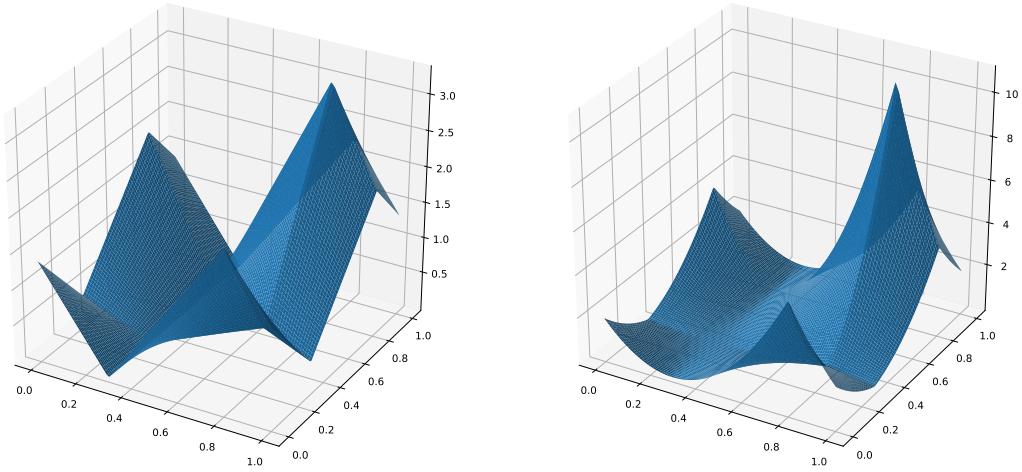


Figure 5.8.: Left: shifted g-function, right: squared shifted g-function (for second moment calculation)

already has more than thousands of points. Therefore, additionally the modified basis has been tested with lmax set to three for reliability. Figure 5.10 shows the errors for the refinement without and with modified basis. With the modified basis, the initial sparse grid has fewer points, so the adaptive refinement has more freedom in its decision about where grid points are located. High order quadrature rules cannot lead to much higher accuracy in the expectation results than low order quadrature because in the single dimensions the function has order one except for the places with discontinuous derivation.

5.3. A Discontinuous Test Function

The test function from "Sparse grid collocation schemes for stochastic natural convection problems" [GZ06], extended into three dimensions, has a discontinuity which is not in the derivation. The function is defined as follows:

$$f(x, y, z) = \exp(-x^2 + 2 \cdot \text{sign}(y)) + z$$

Figure 5.11 shows a plot of the two-dimensional version of the function, where z is set to zero. In the original problem, each parameters is uniformly distributed in $[-1, 1]$. To test adaptive weighted integration methods, in this test case the parameters are normally distributed: $x, y, z \sim \mathcal{N}(0.2, 1.0^2)$ The parameters' expectations are set to

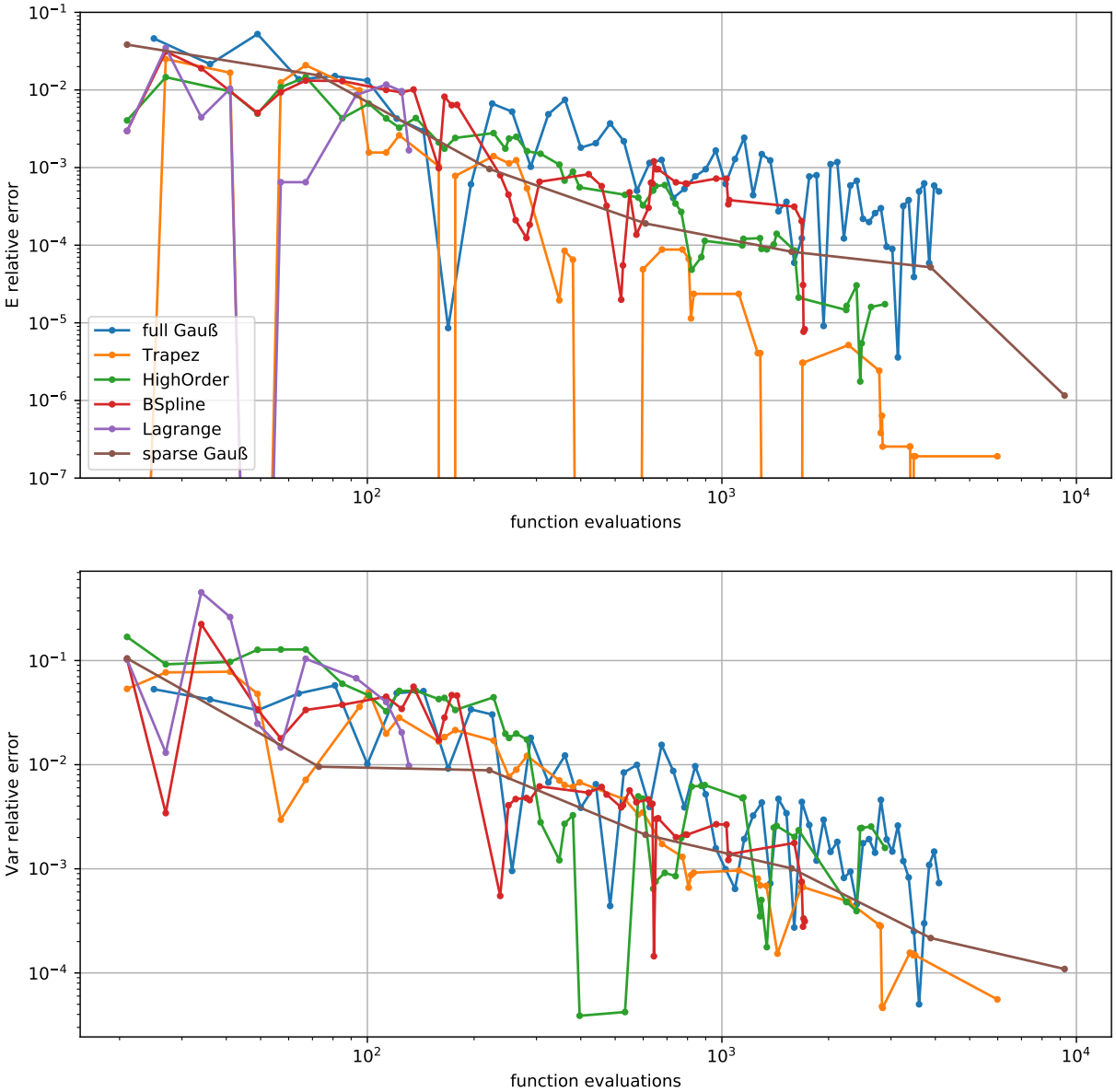


Figure 5.9.: Relative errors for the shifted G-function

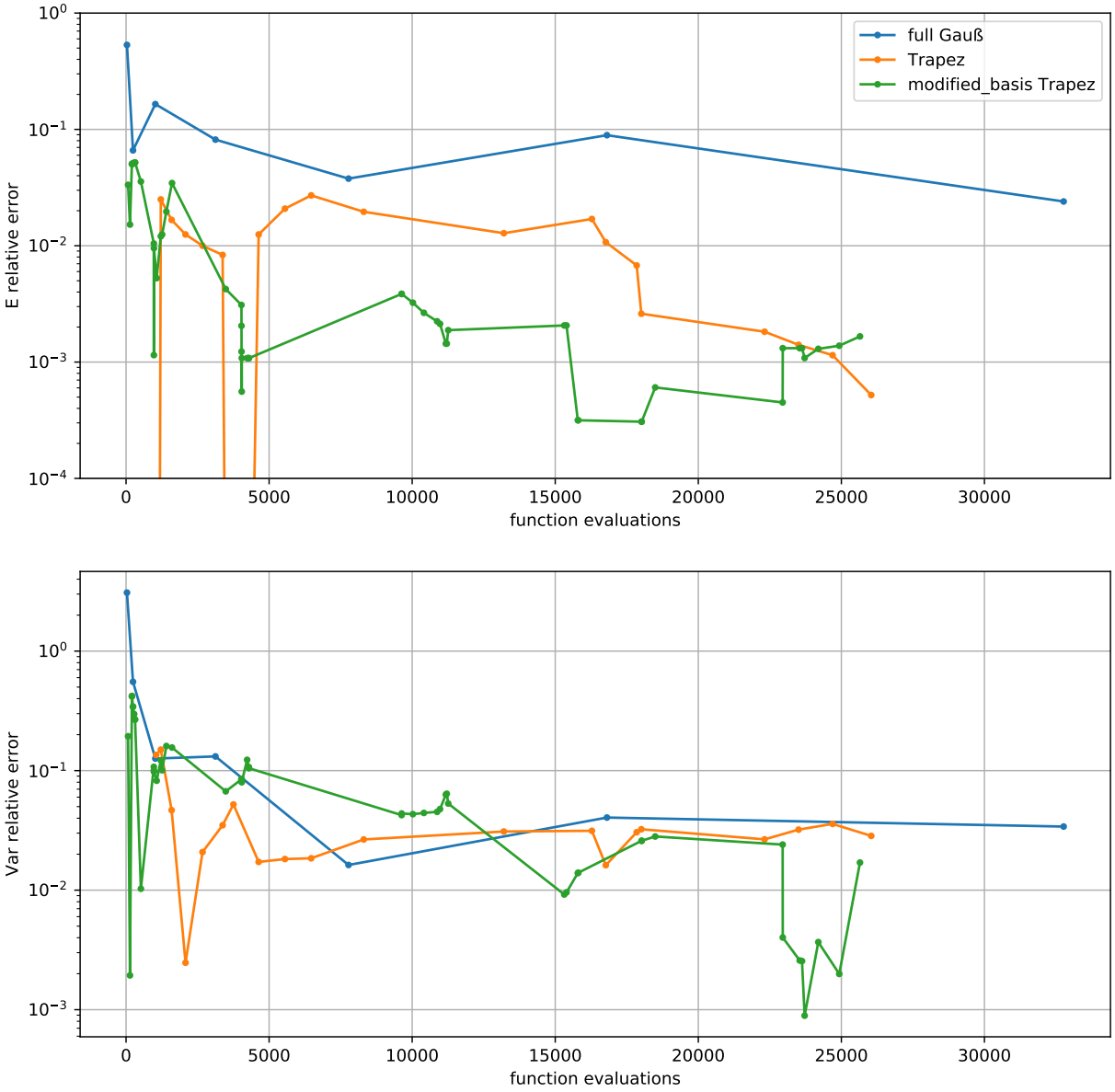


Figure 5.10.: Relative errors for the five-dimensional shifted G-function;
Trapez is the (weighted) global trapezoidal grid with boundary points and lmax is set to two;
modified_basis Trapez is the same grid but with modified basis instead of boundary points and lmax is set to three.

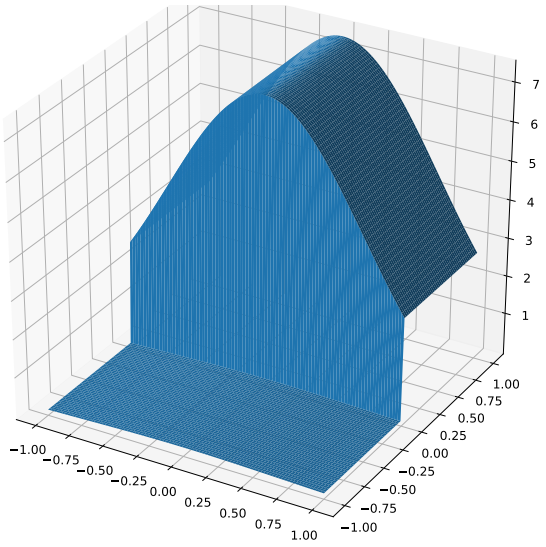


Figure 5.11.: A discontinuous test function (2D version) [GZ06]

0.2 so that the initial grid is not aligned with the discontinuity and other parts of the function which may make the test an unrealistic corner case. The standard deviations 1.0 should make the set of parameter values where the probability is high approximately overlap with the original problem domain. The reference solution is calculated with Chaospy's Monte Carlo method with the Halton Sequence and 2^{18} points. It is not a Gauß quadrature solution because Gauß and other quadrature methods cannot perform very well due to the discontinuity. The plots in Figure 5.12 show relative errors in expectation and variance for various weighted integration methods. In the adaptive refinement methods, boundary points are excluded and lmax is set to two. When the number of function evaluations is relatively low, the sparse grid Gauß solutions, where the standard combination scheme is employed with Gauß Hermite grids, are more accurate than Chaospy's full grid Gauß solutions. The adaptive trapezoidal quadrature performs best in this test case. It has been tested with the weighted global trapezoidal grid and the non-weighted version, where the function's input parameters are changed with the inverse transformation method. The weighted grid calculates the expectation more accurately; the variance is obtained with the first and second moments, so its error can depend on the expectation error.

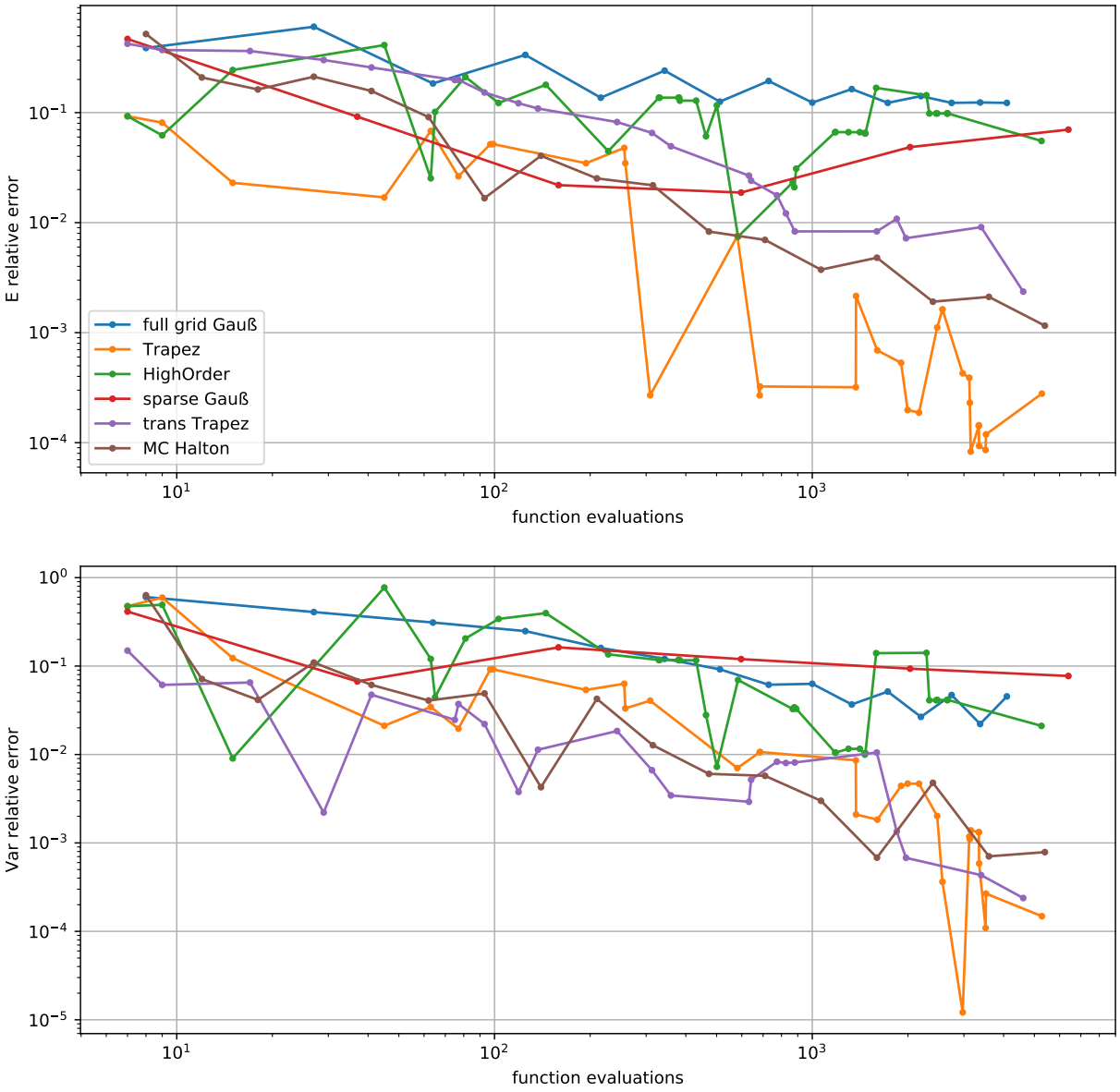


Figure 5.12.: Relative errors for the discontinuous function with normal distribution; trans Trapez refers to non-weighted quadrature with the inverse transformation method, and MC Halton is the Monte Carlo method with the Halton sequence

6. Conclusion and Future Work

Weighted multidimensional quadrature has been implemented for the single dimension spatially adaptive refinement strategy in the sparseSpACE framework; a grid operation class employs this weighted quadrature to perform UQ-related tasks with the help of the Chaospy and SciPy Python libraries. The test results reveal that the adaptive refinement strategy can outperform full and sparse grid Gauß quadrature if the problem function has discontinuities. The results also show that integration with the weighted grids is more accurate than the inverse transformation method, which should be approximately the same as directly integrating the function multiplied with the weight function. The program has a big number of degrees of freedom regarding the configuration parameters; examples are the decision on whether the program should include boundary points or extrapolate these points with a modified basis, and special grid-specific parameters, such as a maximum polynomial degree. Since the refinement is adaptive, the user needs to specify an initial `lmax` parameter, which, as explained in the test results chapter, can lead to problems in a few situations if it is too small.

The single dimension spatially adaptive refinement method is still under development, so a future version of the algorithm may obtain more accurate results. To investigate in which cases the adaptive refinement is able to surpass Gauß quadrature, the program could be adjusted to do optimal refinement steps. To this end, instead of using an error estimator to decide on which refinement objects it splits, for testing it could try every possible refinement step in advance and then split the refinement objects where the highest error reduction can be achieved. Furthermore, during the PCE coefficient calculation, the error estimation could incorporate sobol indices for the respective dimensions; instead of using only the Sobol indices for the refinement decisions, which is done in [SGL15], the algorithm could use them to weigh errors differently in each dimension. Many smooth problem functions can be integrated more accurately with a higher order quadrature rule; for these functions the weighted trapezoidal grid may not be able to surpass the weighted non-adaptive sparse grid Gauß quadrature. Weighted global grids are not yet or not completely implemented for many high order quadrature methods. This includes the `GlobalHighOrderGrid`'s nonnegative least squares mode, where for given points, the algorithm changes quadrature weights so that many weights are zero and the quadrature order does not change. The weighted Lagrange basis grid currently calculates weights with numerical quadrature which is

very slow; with some probability distributions it is possible to calculate these integrals exactly; with other distributions, faster, probability-specific quadrature methods could be implemented. A weighted B-Spline basis grid would require the weighted integration of polynomials within subintervals, too; it additionally needs to calculate knots outside the probability domain boundaries, where positions cannot be transformed with the PPF. A weighted Simpson grid could be implemented similar to the weighted Trapezoidal grid. Since the first equation in the method of undetermined coefficients uses the relatively accurate CDF, the composite quadrature rule can probably use a slightly inaccurate numerical quadrature for the first and second moment calculation without substantial quality reduction in the final quadrature weights.

A. Appendix

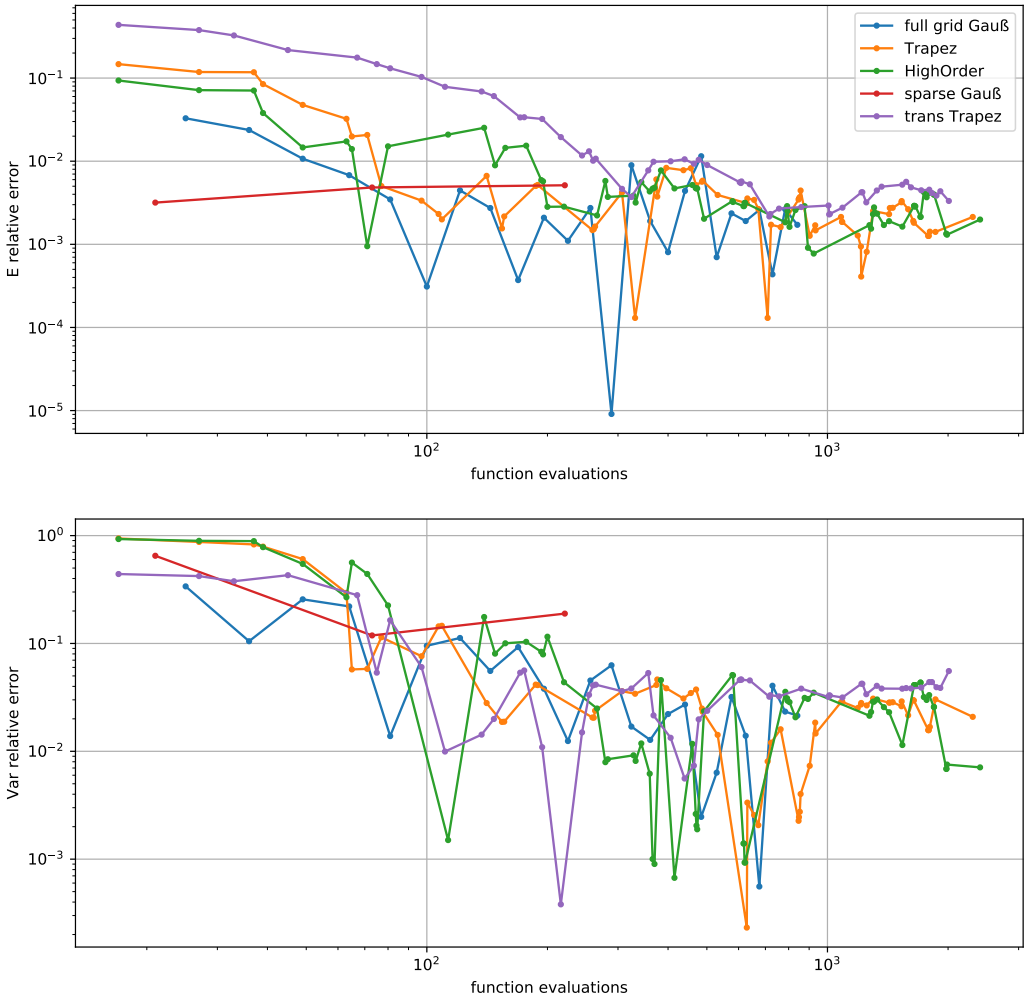


Figure A.1.: Relative errors for the expectation and variance, calculated with the moments, of the 210th time step in the sheep coyote problem;
the reference solution has been obtained with Chaospy's Monte Carlo method with the Halton Sequence with 2^{10} points.

List of Figures

Bibliography

[al08] A. M. et al. "CALCULATIONS OF SOBOL INDICES FOR THE GAUSSIAN PROCESS METAMODEL." In: (Feb. 2008). URL: https://arxiv.org/pdf/0802.1008.pdf.

[al15] M. I. et al. "An overview of uncertainty quantification techniques with application to oceanic and oil-spill simulations." In: (Aug. 2015). DOI: https://doi.org/10.1002/2015JC011366. URL: https://agupubs.onlinelibrary.wiley.com/doi/full/10.1002/2015JC011366.

[Bad17] M. Bader. *Algorithms for Scientific Computing – 1D Hierarchical Basis*. 2017. URL: https://www5.in.tum.de/lehre/vorlesungen/asc/ss17/hierbas_1D.pdf.

[BG04] H.-J. Bungartz and M. Griebel. "Sparse grids." In: (2004). DOI: https://doi.org/10.1017/S0962492904000182. URL: https://ins.uni-bonn.de/media/public/publication-media/sparsegrids.pdf.

[GG03] T. Gerstner and M. Griebel. "Dimension-Adaptive Tensor-Product Quadrature." In: (2003). URL: https://ins.uni-bonn.de/media/public/publication-media/dimadapt.pdf.

[GSZ92] M. Griebel, M. Schneider, and C. Zenger. "A Combination Technique For The Solution Of Sparse Grid Problems." In: (1992). DOI: 10.1.1.33.3530. URL: https://pdfs.semanticscholar.org/5c53/d4de8144d8f79a6c348063bd6329a53f3530.pdf.

[GZ06] B. Ganapathysubramanian and N. Zabaras. "Sparse grid collocation schemes for stochastic natural convection problems." In: (Dec. 2006). DOI: https://doi.org/10.1016/j.jcp.2006.12.014. URL: https://www.dropbox.com/s/vzapll2ynde2vvw/SmolyakCollocationA.pdf?dl=0.

[Hea02] M. T. Heath. *SCIENTIFIC COMPUTING: An Introductory Survey, Second Edition*. Elizabeth A. Jones, 2002. ISBN: 0-07-239910-4. URL: https://juarezrd.files.wordpress.com/2013/09/scientific-computing-michael-t-heath.pdf.

[Huy09] D. Huybrechs. "Stable high-order quadrature rules with equidistant points." In: (May 2009). URL: https://core.ac.uk/download/pdf/81947372.pdf.

[Möl18] H. Möller. "Dimension-wise Spatial-adaptive Refinement with the Sparse Grid Combination Technique." In: (Oct. 2018). URL: https://mediatum.ub.tum.de/doc/1463024/1463024.pdf.

[Nec18] D. T. Neckel. *TUM Lecture Slides: Algorithms for Uncertainty Quantification*. 2018. URL: https://www5.in.tum.de/wiki/index.php/Algorithms_for_Uncertainty_Quantification_-_Summer_18.

[Obe] M. Obersteiner. *sparseSpACE - The Sparse Grid Spatially Adaptive Combination Environment*. URL: https://github.com/obersteiner/sparseSpACE.

[Pfl10] D. Pflüger. "Spatially Adaptive Sparse Grids for High-Dimensional Problems." In: (Feb. 2010). URL: https://www5.in.tum.de/pub/pflueger10spatially.pdf.

[SGL15] D. S.G.L. "Dimension-adaptive sparse grid for industrial applications using Sobol variances." In: (Mar. 2015). URL: https://pdfs.semanticscholar.org/a2e4/17ed3b5903d2015d268669e2c35db9e0ea1b.pdf.