DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Adaptive Quadrature with the Combination Technique for UQ Applications

Anastasiya Liatsetskaya

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Adaptive Quadrature with the Combination Technique for UQ Applications

Adaptive Quadratur mit der Kombinationstechnik für UQ Anwendungen

Author:	Anastasiya Liatsetskaya
Supervisor:	Prof. Dr. rer. nat. habil. Hans-Joachim Bungartz
Advisor:	M.Sc. Michael Obersteiner
Submission Date:	15.06.2020

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

Munich, 15.06.2020 Anastasiya Liatsetskaya

Abstract

The core of this bachelor thesis is comparison of results produced by given adaptive quadrature rule with the Combination Technique and the results computed by non adaptive quadrature rules. The function f(x) for which the weighted integral is approximated represents the values of numerical simulation containing uncertain parameters. According to the results the Quasi Monte Carlo quadrature rule has shown the best results and the the adaptive quadrature rule has shown similar values with Gaussian quadrature rule in estimated error decay rate and estimated error values.

Der Hauptziel des Bachelorthesises war der Vergleich von Ergebnissen des gegebenen adaptiven Quadraturregel mit der Combination Technique und von Ergebnisses der nicht adaptiven Verfahren. Die Funktion f(x), von der man das gewichtete Integral approximiert, representiert die Werte von der numerischen Simulation mit unsicheren Parameter. Anhand der Ergebnissen die Quasi Monte Carlo quadratur Regel hat die beste Ergebnisse gezeigt,der adaptive quadratur Regel mit der Combinations Technique hat ähniches Verhalten zum Gaussquadratur in geschätzen Werten und die Fehlerverkleinerungs Rate.

Contents

1 Introduction

Some parameters of a numerical simulation can be random variables; this might be a property of a modelled system or a consequence of,for example, measurement errors which occured by the computation of the uncertain parameters. One might be interested in the influence of random parameters on the outcome of the simulation. According to [1] one approach could be to approximate the simulation result $u(x, \vec{y})$ with the vector $\vec{y}$ with the uncertain parameters by using the Generalized polynomial chaos expansion:

$$u_N^P(x, \vec{y}) = \sum_{m=1}^{M} \hat{u}_m(x)\Phi_m(\vec{y}),\, M = \binom{N+P}{N}$$

as the gPC approximation of the P^{th} order with $\Phi_m(\vec{y})$ be the orthogonal polynomial with respect to the scalar product $\int g(\vec{y})f(\vec{y})\rho(\vec{y})d\vec{y}$ induced by the joint probability density $\rho(\vec{y})$ of the uncertain parameters y_i. The coefficients $\hat{u}_m(x)$ are defined as

$$\hat{u_m}(x) = \int u(x,y)\Phi_m(\vec{y})\rho(\vec{y})d\vec{y} \text{ for } 1 \leq m \leq M$$

The pseudo spectral approach computes an approximation of the coefficients $\hat{u_m}(x)$ with

$$\hat{w_m}(x) = \sum_{j=1}^{Q} u(x, y^{(j)})\Phi_m(y^{(j)})\alpha^{(j)}$$

by choosing the evaluation points $y^(j)$ and weights $\alpha^{(j)}$ so,that the sum

$$\sum_{j}^{Q} f(y^{(j)})\alpha(j)$$

is an approximation for $\int f(\vec{y})\rho(\vec{y})d\vec{y}$. As a result, one obtains an approximation of generalized polynomial chaos expansion of $u(x, \vec{y})$ the quality of which might depend on the quality of the approximation of weighted integral $\int f(\vec{y})\rho(\vec{y})$ by using function evaluations and weights.

In this work several different quadrature rules the description of which can be found in the chapter 2 have been applied for computing the integral approximations of $\int f(\vec{y})\rho(\vec{y})$ where $f(\vec{x})$ stands for a value of numerical simulation applied to a vector with values of non-zero probability of the uncertain parameters. Two methods to compute error estimations are described in the chapter 3. The simulation is described in the chapter 4. The obtained results are compared for the used quadrature rules in the chapter 7. In order to compute the integral approximations with some of the used quadrature rules the sparse-SpACE framework was used. Some of its classes were modified in order to allow methods from the framework obtaining the values of the simulation for a given vector with values of the uncertain parameters. The brief description of a part of the functionality the framework provides is given in the chapter 5 and added modifications in the chapter 6.

2 Quadrature rules

2.0.1 Gaussian quadrature

One possibility to approximate a definite Riemann-integral $\int_a^b f(x)dx$ is to build an interpolating polynomial of the function f and integrate exactly the polynomial.For a polynomial p which interpolates the function f on n points t_i holds:

$$p(t_j) = f(t_j)\ \forall i : 1 \leq i \leq n$$

The quadrature rules used in this work can be written as a quadrature formula $\hat{I}$, which has the structure

$$\hat{I} = (b-a)\sum_0^n \lambda_i f(t_i)$$

The set $t_0 ... t_n$ is the set of support points on which the function f is evaluated. $\lambda_0 ... \lambda_n$ is the set of weights, these weights do not depend on the function f but on the choice of support points if the quadrature formula with n+1 distinct support points integrates exactly every polynomial in P_n. For the support point t_j the Lagrange polynomial L_j is defined as

$$L_j(x) = \prod_{i=0, i\neq j}^{n} (x-t_i)/(t_j-t_i)$$

If the quadrature formula integrates exactly every polynomial in P_n, then $\hat{I}(L_j(x)) = \int_a^b L_j(x)dx$ is fulfilled,and the quadrature weights can be obtained by:

$$\lambda_j = (1/(b-a))\int_a^b L_j(x)dx$$

because $L_j(t_i) = \delta_{ij}$. For n+1 pairwise different support points there exists only one polynomial $p \in P_n$ that fulfills given n+1 interpolation conditions. A quadrature formula has order n if for every polynomial $p \in P_n$ holds $I(p) = \hat{I}(p)$. Gaussian quadrature rule has order 2n+1 and only needs n+1 function evaluations in order to calculate exact integral of a polynomial with a degree less or equal to 2n. A quadrature formula which integrates interpolating polynomial of a function f cannot have order greater,than 2n+1 according to [2].

According to [2] one can view an integral

$$\int_a^b f(x) * g(x) * \omega(x)dx$$

as a scalar product (f,g). For each scalar product which has the structure $\int_a^b f(x) * g(x) * \omega(x)dx$ there exists a uniquely determined family of normalized orthogonal polynomials

$p_k \in P_k$. p_k has k real distinct roots and each root is $\in]a,b[$ as proven in [2]. One chooses $t_0, .., t_n$ as the roots of $p(n+1)$, where $p_{(n+1)}$ is orthogonal to the space P_n. A polynomial $p \in P_{2n+1}$ can be represented as

$$p = p_{(n+1)} * q + r$$

with $p, q \in P_n$. Then

$$\int_a^b p(x)w(x)dx = \int_a^b p_{(n+1)}(x) * q(x) * w(x)dx + \int_a^b r(x)w(x)dx$$

since $p_{(n+1)}$ is orthogonal to P_n with respect to scalar product (f,g) and $q \in P_n$ then

$$\int_a^b p_{(n+1)}(x) * q(x) * w(x)dx = 0$$

Hence, one obtains the following equality:

$$\int_a^b p(x)w(x)dx = \int_a^b r(x)w(x)dx$$

Orthogonal polynomials p_k for $k \leq n+1$ can be generated with the Gram-Schmidt orthogonalisation method given the monomial basis of P_{n+1}.

Gaussian quadrature is a non adaptive quadrature rule. The algorithm needs to know the integration domain and function values on n+1 evaluation points the position of which does not depend on behaviour of the function on the integration domain. The accuracy of Gaussian quadrature can be calculated by $I(f) - \hat{I}(p) = I(f) - I(p) = I(f-p)$, where $(f-p)(x)$ is a local interpolation error on the point x. According to [2] for every function $f \in C^{2n+2}$ the quadrature error $\epsilon[f]$ fulfills:

$$\epsilon[f] = f^{2n+2}(\tau)(p_{n+1}, p_{n+1})/(2n+2)!$$

for some $\tau \in [a,b]$. The small local errors of the interpolant do not guarantee, that the integral error is small. If the errors have the same sign and the integration domain is large enough, then the local errors can sum up. Respectively, there can occur the situation, where the interpolant has large local errors with different sign, but the computed integral of the interpolant is equal to the integral of the approximated function.

Let G_n be the set of grid points produced by Gaussian quadrature with n points. For two numbers $n, k \in N$ with $n \neq k$ and $n > k$ with sets of grid points G_n and G_k it can be, that $G_k \cap G_n = \emptyset$. Thus, the grid generated by Gaussian Quadrature might not reuse points that were computed in previous integral approximations. If the evaluation of the function is expensive and multiple experiments with different amounts of points per dimension need to be performed, then the reuse of already computed points can reduce computational time. If an integral approximation of a multivariate function is to be computed, then one can following the principle of Fubini's theorem along each dimension construct interpolant by using one dimensional quadrature rules. For vector valued functions the grid is constructed via tensor product of 1D grids. If the same number of points k is used in every 1D grid, then the total number of used points is k^d and it has exponential growth depending on the dimension d. As a result, for high-dimensional problems Gaussian quadrature might become inapplicable due to its computational time.

2.0.2 Monte Carlo

Monte Carlo method can be used to compute an approximation of expectation E[X], where X is a random variable with existing expectation. According to [3] a definite integral $\int_a^b f(x)dx$ of an integrable function f is equal to the

$$\int_a^b f(x)dx = E[F(X)] * (b-a)$$

,where E[F(X)] is the expectation of a uniformly distributed on the interval [a,b] random variable F(X) with F(x)=f(x). As a result integration of a function can be transformed into computation of the expected value of uniformly distributed random variable. For the chapter the integral $\int_0^1 f(x)dx$ is considered. The Monte Carlo approximation with N function evaluations is defined as

$$\hat{I}[f] = (1/N) * \sum_{n=1}^{N} f(x_n)$$

The Strong Law of Large Numbers proves, that a sequence of Monte Carlo approximations will converge to the expectation E[X] with probability 1 under assumption that X is integrable and the expectation E[X] is finite, that is

$$\lim_{N\to\infty} \hat{I}_N[f] \xrightarrow{\text{almost sure}} I[f]$$

Let ϵ_N be defined as $\epsilon_N = \hat{I}[f] - I[f]$. The Central Limit Theorem gives according to [3] the probability with which the error ϵ_N of a Monte Carlo approximation with N samples is bounded from above by the function $c(1/\sqrt{N})$ with constant c under assumption that N is large enough. More precise: Let Y_N be defined as $Y_N = X_1 + ... + X_N$; X_i have the same distribution, are independent and have defined expectation E[X] and variance Var[X]; $\Phi(x)$ is the cumulative distribution function of $\mathcal{N}(0,1)$, then

$$Pr(a < (\sqrt{N}/\sqrt{Var[X]}) \cdot (Y_N - N \cdot E[X])/N < b) \approx \Phi(b) - \Phi(a)$$

which is equivalent to

$$Pr(a < (\sqrt{N}/\sqrt{Var[X]}) \cdot \epsilon_N < b) \approx \Phi(b) - \Phi(a)$$

Exact formulations and proofs of the Central Limit Theorem and the Strong Law of Large Numbers can be found in [4].

In order to compute an approximation $\hat{I}_N$ a sequence of samples x_i need to be generated according to the distribution of the random variable X. If the inverse of the cumulative distribution function $F_X^{-1}(x)$ of random variable X can be computed or numerically approximated, then the inverse Rosenblatt transformation can be used in order to obtain from a uniformly distributed random variable on the unit interval [0,1] the required random variable X.

In the upper bound for the Monte Carlo Quadrature error derived by the Central Limit Theorem the dimension of a problem does not appear as an explicit argument, this allows to use Monte Carlo algorithm for high-dimensional problems. However, if one chooses a sequence of functions (f_i) with growing dimension d_i and growing variance in dependence from the dimension.Then the constant $b\sqrt{Var[X]}$ in the upper bound of the quadrature error can also grow with i and can cause slower convergence of the method applied to the functions f_i with growing i.

2.0.3 Low-discrepancy sequences

In a sequence of points sampled from a uniformly distributed random variable there can occur clumping of points. This effect is also present in pseudo-random sequences and is illustrated for a 2D sequence with 1000 samples in Figure 1. Such grouping of points can reduce speed of convergence of Monte Carlo method. Discrepancy measures how uniform the points of a given sequence are distributed in the unit cube. For a finite sequence (x_i) with N elements defined on the unit interval [0,1) the discrepancy D_N is defined as

$$D_N = \sup_{0 \le \alpha < \beta \le 1} |\#x_i \in [\alpha, \beta)/N - (\beta - \alpha)|$$

The star discrepancy D_N^* considers only intervals with the left border $\alpha = 0$:

$$D_N^* = \sup_{0 < \beta \le 1} |\#x_i \in [0, \beta)/N - \beta|$$

For d-dimensional sequences defined on the unit cube $[0,1)^d$ these definitions are generalized by using instead of the 1D intervals $[\alpha, \beta)$ rectangular sets

$$J = \{(a_1 ... a_d) \colon \forall k \ : 1 \le k \le d \ : \ 0 \le \alpha_k \le a_k < \beta_k \le 1\}$$

and by subtracting instead of the length of the interval $(\beta - \alpha)$ the volume V(J) of a set J, where $V(J) = \prod_{k=1}^{d} (\beta_k - \alpha_k)$

According to ([5]) the following inequalities hold for d-dimensional sequences:

$$D_N^* \le D_N \le 2^d D_N^*$$

The Koksma-Hlawka theorem shows, that there exist an upper bound for the error

$$|\epsilon[f]| = |\int_{I^d} f(x)dx - (1/N) \cdot \sum_{n=1}^{N} f(x_n)|$$

which depends on the star discrepancy of the used for approximation sequence of points

$$|\epsilon[f]| \le V[f] D_N^*$$

for a function f with bounded variation V[f]. The value of variation V[f] does not depend on the sequence $(x_i)_{i \le N}$ used for the f-integral approximation of the form $(1/N) \cdot \sum_{n=1}^{N} f(x_n)$. As a result a sequence $(\hat{x}_i)_{i \le N}$ with smaller star discrepancy can lead to a better approximation of I[f]. If a d-dimensional integral is to be approximated, then one can use a Halton sequence.

In a dimension k if $\sum_{n=0}^{M} \alpha_n b^n$ is the base-b representation of a natural number i, where b is the k-th prime number, then the i-th member of a Halton sequence is defined as

$$x_i = \sum_{n=0}^{M} \alpha_n b^{-n-1}$$

in the base-b representation. This approach generalizes the generation procedure of 1D Van der Corput sequence which uses prime base b=2. A 2D Halton sequence with 1000 samples is illustrated on Figure 2. As shown in ([3]) an upper bound for the discrepancy of a Halton sequence is

$$D_N(Halton) \leq c_d(logN)^d N^{-1}$$

with a constant c_d depending on the dimension d. The upper bound of $D_N(Halton)$ depends on the dimension of a problem, consequently, for high dimensional integral approximations a Halton sequence can show slower convergence. The algorithm for generation of a Halton sequence is a deterministic algorithm: for a given N it will produce the same sequence of points (x_n). If H_n and H_k denote the sets of points in the Halton sequences with n respectively k elements and n<k, then $H_n \subset H_k$.

2.0.4 Sparse Grids

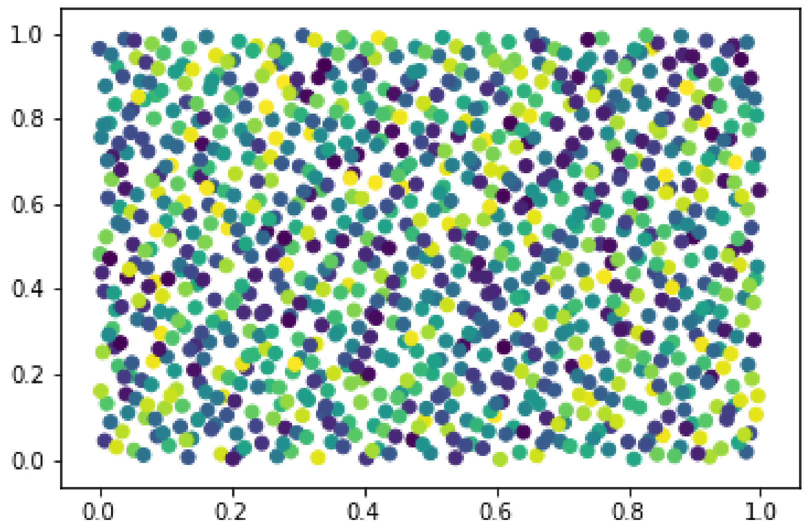
The application of 1D quadrature grid based methods for computing of the integral approximation for multivariate functions might result in full grids with high amount of points:for example, in Gaussian quadrature if one uses k points in each dimension, then the total number of points in the grid will be k^d. For a given function the accuracy of such quadrature rules might depend on the used number of points per 1D grid. If a high-dimensional function is expensive to evaluate and given error tolerance is small, this might lead to the lack of the storage for function values and long computational time.

The sparse grids technique has different applications, but in this chapter it will only be described in the context of interpolation and integration. The detailed information about sparse grids can be found in ([6]). The sparse grids technique can use hierarchical basis functions to build an interpolant on a grid with according to ([7]) $O(N \cdot log(N)^{d-1})$ points, if optimized for L_2 or L_{max} norm, compared to the number of points on the full grid $O(N^d)$. In this chapter only hierarchical linear hat basis functions will be described, however it is not the only type of hierarchical basis functions which can be used for a sparse grid construction; for example, B-spline basis functions or piecewise d-polynomial basis functions can be used depending on the problem, more detailed information about these basis functions can be found in ([8])

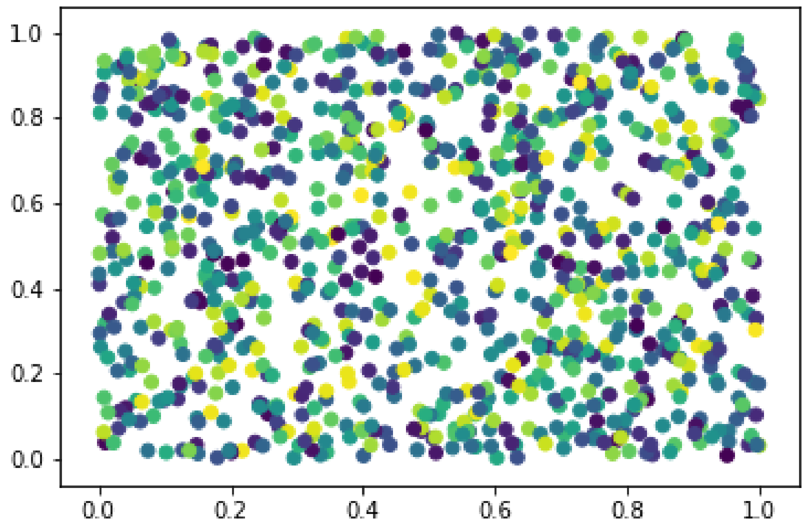
A hat function is defined as

$$\phi(x) = max(1 - |x|, 0)$$

dilation and translation applied on $\phi(x)$ allow to center the hat function around a particular point and to change the size of non-zero valued area of the function. The nodal basis V_i to a space $\hat{V}_l$ - a space of piecewise linear functions, which are zero on boundary points and each



(a) Fig.2: Points generated by the Halton sequence



(b) Fig. 1: Points generated by pseudo-random sequence function in this space can be defined with $2^i - 1$ values- can be obtained with transformations of hat function

$$\phi_{i,n} = \phi(2^i \cdot x - n)$$

As a result, one can represent a piecewise linear function f from the space $\hat{V}_l$ as

$$f(x) = \sum_{i=1}^{2^l-1} f(x_i)\phi_{l,i}(x)$$

If a set W_l is defined as

$$W_l = \{\phi_{l,i} : \phi_{l,i} \in V_l, 1 \leq i \leq 2^l - 1, \text{i is odd}\}$$

then the set

$$\cup_{i=1}^{l} W_i$$

is also a basis to the space $\hat{V}_l$. A function $f \in \hat{V}_l$ can be obtained via linear combination of hierarchical basis functions:

$$f(x) = \sum_{l=1}^{L} \sum_{i=1,\text{i odd}}^{2^l-1} \alpha_{i,l}\phi_{l,i}$$

with coefficients $\alpha_{i,l}$ called hierarchical surpluses, that depend on the function values on the point $x_{l,i}$ and on the quality, with which the interpolant build from hierarchical basis functions from levels $1 \leq i < l$ approximates the function f on the point $x_{l,i}$. The number of functions in W_l is 2^{l-1} and their support reduces by factor 2 from one level l to the next. From 1D hierarchical hat basis one can obtain by tensor product hierarchical basis functions

$$\phi_{\vec{l},\vec{i}}(\vec{x}) = \prod_{k=1}^{d} \phi_{l_k,i_k}(x_k)$$

which can be used to build d-dimensional piecewise linear functions. If a d-dimensional piecewise linear function is used to interpolate a function with bounded second mixed derivative, then according to ([6]) for the absolute value of the coefficients $\alpha_{\vec{l},\vec{i}}$ holds:

$$|\alpha_{l,i}| \leq 2^{-d} \cdot 2^{-2|l|_1} \cdot |u|_{2,\infty}$$

This result allows to estimate as shown in ([6]), how much the functions from the set W_l contribute to the interpolant; the estimation depends on the level l, on dimension of the problem d, on the used norm and on the value, which is an upper bound to the second mixed derivative of the function. If one interprets the contribution to the interpolant as benefit and the number of points, used to define functions from the set W_l as cost, then according to ([8]) a solution to the obtained optimization problem for norms L_2 and L_∞ is the set

$$W_{\vec{l}} : |\vec{l}|_1 \leq n + d - 1$$

The error of the sparse grids interpolant for functions with bounded second mixed derivative measured with L_2 or L_∞ is in $O(N^{-2} \cdot log(N)^{(d-1)})$, compared to the error of full grid interpolant in $O(N^{-2})$, as shown in [6].

Hierarchical hat basis functions allow to build a piecewise linear interpolant with the value zero on the boundary. If one would like to approximate more accurately integral of a function on areas near the boundary, then one possible method would be to add a level zero with two linear basis functions $\phi_{0,0}$ and $\phi_{0,1}$ such, that

$$\phi_{0,i}(x_{0,|i-1|}) = 0$$

$$\phi_{0,i}(x_{0,i}) = 1$$

for $i \in \{0,1\}$. An advantage of such approach is, that for the same fixed n a sparse grid with added level 0 will contain the same interior points as the sparse grid without added points on boundaries. However, as shown in [8] dependent on the dimension d of the problem the majority of used points in the grid might be located at the boundary. An alternative approach would be to modify the one dimensional hat basis functions a support boundary of which corresponds to the domain boundary so that the functions extrapolate towards the boundary.

One could also construct a sparse grid by using non nested one dimensional grids. If the one dimensional Grids with placement of points corresponding to the Gaussian quadrature rule are applied, then one becomes Sparse Gauss grid. As shown in [9] for the same n the Sparse Gauss quadrature might significantly outperform the sparse grid constructed with linear hat basis functions, however, it also might require more evaluation points.

2.0.5 Combination technique

If one denotes a solution obtained from a sparse grids space as u^s and the set of points used to build the grid as G_s, then the Combination technique [10] allows to obtain a function u^c from a linear combination of solutions constructed on full grids with sets of points $G_{c,i}$ for i-th element in the linear combination,such that $u^c(x_i) = u^s(x_i)\ \forall x_i \in G_s$ and $G_{c,i} \subseteq G_s\ \forall G_{c,i}$ i element of the index set I of the combination. In case of d- dimensional hat basis functions one can represent exactly the interpolant u_s though a combination of solutions on full grids using linear functions to approximate the integrated function on each subarea of the domain: $u^c(x) = u^s(x)$ due to fulfillment of the interpolation condition by u^c. According to [7] one possible combination to obtain a function from the sparse grids space $\bigoplus_{|\vec{l}|_1 \leq n+d-1} W_l$ is

$$u_l^c = \sum_{q=0}^{d-1} (-1)^q \binom{d-1}{q} \sum_{\vec{l} \in I_{l,q}} u_l$$

with the index set

$$I_{l,q} = \{\vec{l} \in \mathbb{N}_0^d \,|\, |\vec{l}|_1 = l + d - 1 - q\}$$

One point x_i can occur in several full grids, but no full grid in the combination uses points other, than that contained in the sparse grid. The truncated Combination Technique allows to use a truncation parameter $\tau_i \in N_0 \cup -1$ for a dimension i for selection of level vectors $\vec{l}$ in the index set:

$$I_{l,q} = \{\vec{l} \in N_0^d \,|\, |\vec{l}|_1 = l + d - 1 + (\sum_{i=1}^{d} \tau_i) - q, l_i > \tau_i\}$$

High positive values of τ_i exclude grids with sparse one dimensional grids from the scheme leaving grids with high amount of points across every dimension depending on the values of τ_i.

The choice of subspaces in $\bigoplus_{|\vec{l}|_1 \leq n+d-1} W_l$ is optimal with respect to the norms L_2 and L_∞ for functions with bounded second mixed derivative. Depending on the problem one might chose different subspaces to build a sparse grid. If the level vectors of chosen grids form a downward closed index set with

$$\vec{l} \in I \wedge l_j > 0 \implies \vec{l} - \vec{e}_j \in I \text{ for } 1 \leq j \leq d$$

then one possible choice of the coefficients $c_{\vec{l}}$ for the combination would be according to [7]

$$c_{\vec{l}} = \sum_{\vec{l} \leq \vec{i} \leq \vec{l}+\vec{e}, \vec{i} \in I} (-1)^{|\vec{i}-\vec{l}|_1}$$

The sparse grids technique allows to compute solutions in parallel on different full grids using the quadrature rules developed for full grids and then to combine appropriately the obtained functions. The data structures used to represent the combined result might be simpler compared to the data structures which preserve the hierarchical structure of the basis functions used to construct a function from a sparse grids space.

2.0.6 Dimension-wise Spatial Refinement with the Sparse Grid Combination Technique

Some functions have different behaviour depending on the considered area of the domain, for example, a spike impulse $f(t) = 1/(10^{-4} + t^2)$ defined on the interval [-1,1]. If on the most parts of the integration domain the integral of a function is approximated well by the integral of an interpolant and only on some small areas the approximation produces large errors, then in non adaptive grid-based methods raising the number of points in the whole grid might result in significant increase of computational load by achieving only small benefit. In fact, many new points might be located in the areas which already were approximated so that the integration error in those areas is small. One possible solution is to chose the position of the new points in the domain according to the behaviour of the function. In case of multivariate functions the local errors along different dimensions might significantly differ.

A Dimension-wise Spatial Refinement with the Sparse Grid Combination Technique [11], described in this chapter is implemented in the SparseSpACE framework, refines along single dimensions according to values of an error estimator and then represents with help of Combination Technique a sparse grid as a linear combination of full grids. The algorithm starts with a sparse grid of level l; 1 dimensional vector of points $\vec{P^k}$ for a dimension k and a vector with levels of this points L^k are used for each dimension, based on the vectors $\vec{P^k}$ and L^k one can construct a tree representing a hierarchical structure of basis functions centered around the points in $\vec{P^k}$. For each child node an error estimation is calculated. Based on this value, the algorithm might add hierarchical children to the leafs of the refinement trees. The stopping criterion for the refinement in the algorithm can be a user defined tolerance value for the integration-error or a number of points p such that, when more points than p are already contained in the grid then further refinement is not applied. A Tensor product applied directly to the grids from one-dimensional refinement trees would result in a full grid. In order to reduce the amount of points in the grid a combination scheme is constructed. The index set I of a combination scheme contains the following level vectors l:

$$I = \{\vec{l} \in N^d | |\vec{l}|_1 \leq max(\vec{l}^{max}) + d - 1, l_i < l_i^{max} \vee (l_i = l_i^{max}, l_k = 1, k \in [d]/i\}$$

with $l_k^{max} = max(\vec{L}^k)$. For each level vector $\vec{l}$ in the index set I a one dimensional set of coordinates is constructed $\vec{P}^{k,\vec{l}}$ for each $k \in [d]$, by applying Tensor product construction to the grids $\vec{P}^{k,\vec{l}}$ one obtains a component grid of the combination scheme. In order to construct a valid combination scheme the following two relations must hold for the sets $P^{k,\vec{i}}$ and $P^{k,\vec{j}}$ with $\vec{i}, \vec{j} \in I, k \in [d]$:

$$P^{k,\vec{i}} \subset P^{k,\vec{j}} \text{ if } \vec{j} \geq \vec{i}$$

$$\vec{P}^{k,\vec{i}} = \vec{P}^{k,\vec{j}} \text{ if } i_k = j_k$$

According to the first property the increase of the level vector $\vec{l}$ adds new points to the existing sets of points without removing elements of this sets. There are multiple possibilities, how to chose points from the vectors $\vec{P}^k, k \in [d]$ to obtain sets $\vec{P}^{k,\vec{l}}k \in [d], \vec{l} \in I$ which produce a valid combination scheme. The selection method of points used by the algorithm is

$$\vec{P}^{k,\vec{l}} = \{P_j^k | j \in [|\vec{P}^k|], L_j^k - \hat{c}_j^k \leq l_k \vee L_j^k \leq 1, (L_j^k < max(D_{lv}(j)) \vee l_k = l_k^{max})\}$$

where

$$\hat{c}_j^k = \max(\{m \in N | \sum_{i=1}^{m} \chi_{k,l}(i,d) \leq c_j^k + \chi_{k,l}(m,k)\}$$

$$\chi_{k,l}(i,j) = \sum_{n=1}^{j} h(c_l^k - i, n)$$

$$h(x,k) = \begin{cases} 1 & \text{if } max(\vec{c}^k) \geq x \\ 0 & otherwise \end{cases}$$

with $D_{lv}(j)$ is a set of levels of every point contained in a subtree of hierarchical refinement tree with root at j-th point of the vector P^k.

Depending on a function f there might be areas in domain on which an error estimator will estimate a large error and a large number of points is needed to compute sufficiently good integral approximation of the area, as a result, the algorithm might start refining a specific area of the domain. One dimensional refinement trees can become in such case unbalanced. By rebalancing of the trees, one can reassign the levels of nodes in the tree; as a result the root with level 1 of a tree can move from point corresponding to the center of the integration domain towards the area, which is intensively refined by the algorithm. This procedure does not change leafs of the refinement trees, the algorithm will consider for the refinement the same points as without tree rebalancing. However, the set of grids selected for the combination scheme might change and the error estimations for the leafs of the refinement trees.

3 Error estimator

3.1 Error estimator

The exact value of an integral might not available and some quadrature rules need information, how much the integral of an interpolant differs in value from the approximated solution for example, the Dimension-wise Spatial Refinement with the Sparse Grid Combination Technique needs that information to make a decision whether to refine or not in a certain subarea of the integration domain. One possible solution would be to use an error estimator $\bar{\epsilon}$ for the error $\epsilon = |I[f] - \hat{I}[f]$. According to the definition of an error estimator in [2] there exist constants $k_1 \leq 1 \leq k_2$ such,that the following relation holds:

$$k_1\epsilon \leq \bar{\epsilon} \leq k_2\epsilon$$

One possibility to construct an error estimator for the value ϵ_2 -an error that is produced by the approximation $\hat{I}_2[f]$- is to compute $\hat{I}_1[f]$ and $\hat{I}_2[f]$ by using two methods with different approximation quality such, that $\epsilon_1 << \epsilon_2$, and then to compare the obtained values:

$$\bar{\epsilon} = |\hat{I}_2[f] - \hat{I}_1[f]| = |\hat{I}_2[f] - I[f] - \hat{I}_1[f] + I[f]| = |\epsilon_2 - \epsilon_1| \qquad (1)$$

As shown in [2] one obtains the following upper and lower bounds for the value $\bar{\epsilon}$:

$$(1 - \epsilon_1/\epsilon_2)\epsilon_2 \leq \bar{\epsilon} \leq (1 + \epsilon_1/\epsilon_2)\epsilon_2$$

An error estimator $\bar{\epsilon}$ constructed according to this principle relies on smaller error ϵ_i of one method compared to the error produced by the second method; the information which data structures are used by the algorithm and which steps are performed to compute an approximation $\hat{I}[f]$ is not used to build $\bar{\epsilon}$.

The Dimension-wise Spatial Refinement with the Sparse Grid Combination Technique computes error estimations for leafs of one-dimensional refinement trees $\vec{P}^k$. For each grid in the combination scheme an error estimation for each leaf in the $\vec{P}^{k,\vec{l}}$ is computed by using surplus values associated with points $\vec{x}^i$ of a grid. A leaf node in the refinement tree $\vec{P}^{k,\vec{l}}$ might not be a leaf node in the tree $\vec{P}^k$ but an ancestor of a leaf node. In case of such node p the algorithm divides equally the obtained error estimation among all descendant leaf nodes of p, whereas the factor m in

$$\epsilon_{leaf}^{k,\vec{l}} += volume * |\alpha_{i_k}| / |leaves(P_{i_k}^{k,\vec{l}}, k)|^m$$

allows to decrease contribution to the error of leaf nodes from hierarchical ancestors with many descendent leaf nodes. A hierarchical surplus value in case of used hierarchical hat basis functions corresponds to a local interpolation error on the point $x_{l,i}$ of the interpolant constructed with grid with levels $\tilde{l} < l$. By multiplying a surplus value with volume of the respective pagoda function one obtains the volume of added function $\alpha_{\vec{l},i}\phi_{\vec{l},i}$. The error estimate of point $p \in \vec{P}^k$ is combined from the error estimates $\epsilon_p^{k,\vec{l}}$ of this points in grids with coefficients $c_{\vec{l}}$ contained in the combination scheme:

$$\epsilon_p^k = |\sum_{\vec{l}\in I} c_{\vec{l}} \cdot \epsilon_p^{k,\vec{l}}|$$

From local error estimates one can obtain the global error estimate ϵ by:

$$\epsilon = \sum_{k=1}^{d} \sum_{j=1}^{|\vec{P}^k|} |\epsilon_j^k|$$

4 Simulation Scenario

The simulation describes the spatial location of persons in campus in a defined time interval. The finish time was set to 1000 and the time step length is 0.9. At each time step the number of persons at each of 9 measure points is computed. The campus layout used in the simulation is shown on Figure 4.1; the picture was generated with help of Vadere GUI. The Vadere framework [1] was used in a given scenario to simulate the movement of a crowd. A detailed information about the framework can be found in [12]. The framework contains implementations of several locomotion models, which allow to simulate the movement of an agent from starting point to the destination point while avoiding obstacles and collisions with other agents which might be present in the simulation scenario. For the simulation scenario the Optimal Steps Model was used to represent the movement of persons in campus, in which the algorithm tries to choose optimally the next step of a person in the circle around the pedestrian with radius depending on the free-flow velocity of the agent. The maximal amount of agents used in the simulation is 200, overall number of pedestrians at given timepoint in campus might be less than 200. The simulation has 4 uncertain parameters: ir with distribution $\mathcal{U}(0.1, 0.3)$ representing IPP Mensa ratio, i_{rtime} with distribution $\mathcal{N}(50, 1)$ representing residence time in IPP Mensa, t_{rtime} with distribution $\mathcal{N}(60, 1)$ representing residence time in TUM Mensa, and v with distribution $\mathcal{U}(1.3, 1.8)$ representing the mean speed of pedestrians.

The given distributions define weight functions which appear in the computation of expected value of the vector random variable representing numbers of persons on measurement points for every time step. In case of a uniform distribution one the interval [a,b] one obtains the following weight function:

$$w_1(x) = 1/(b-a)$$

in case of normal distribution $\mathcal{N}(\mu, \sigma^2)$ the weight function equals to:

$$w_2(x) = (1/(\sqrt{2\pi}\sigma)) \cdot exp(-(x-\mu)^2/(2\sigma^2))$$

If the uncertain parameters are pairwise independent, then one can obtain the weight function for calculation of expected value as:

$$w(\vec{x}) = \prod_{i=1}^{4} w_i(x_i)$$

Orthogonal polynomial systems for the scalar products induced by the obtained weight functions would be Hermite-Polynomials for w_2 as a weight function and Legendre-Polynomials

[1] http://www.vadere.org/

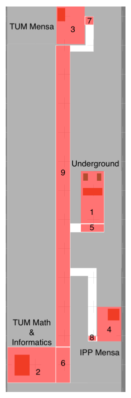


Figure 4.1: Layout of the campus. Measurement points are marked with the respective numbers

for w_1.
Vadere console is given in form of a JAR file, it takes as input a file with ".scenario" extension containing the necessary parameters to perform the simulation including the values of the uncertain parameters. Vadere console can generate multiple output files which can be used to analyze different aspects of the simulation, however in the integral approximation only the file with the number of pedestrians at measure points at each time step was used. In order to decrease memory usage the other output files were removed from the scenario configuration. The Vadere framework allows to include randomized elements in the scenario of the simulation, as a result, if one fixes values of uncertain parameters $(ir, i_{rtime}, t_{rtime}, v)^T$, in such scenario, then the algorithm might compute for equal settings different number of pedestrians to a given time step and measurement point. Since the Vadere framework may use randomized algorithms to compute for a given values of the uncertain parameters the numbers of pedestrians on measuring points, one might obtain a random variable with a probability distribution for the numbers of pedestrians at measurement points. Then one can interpret the simulation results as a function $\hat{f}$ with disturbance in its argument $x + \delta x$ causing the deviation of the value $\hat{f}(x + \delta x)$ from the value $\hat{f}(x)$. If the integral of a function f is the desired value to obtain, then one can measure the influence of the argument disturbance with help of absolute and relative condition of the quadrature task (I,f). According to [2] the absolute k_{abs} and relative k_{rel} condition of the quadrature task (I,f) with $I(f) = \int_a^b f$ with respect to L^1 norm fulfill the following equalities:

$$k_{abs} = 1$$

$$k_{rel} = I(|f|)/|I(f)|$$

5 Sparse-SpACE framework

The Sparse-SpACE framework [1] contains implementations of several quadrature rules including the Standard Combination technique and The Dimension-wise Spatial Refinement with the Sparse Grid Combination Technique. Grid class contains implementations of one dimensional GaussLegendreGrid representing the grid constructed by Gauss-Legendre quadrature rule and one dimensional GaussHermiteGrid for Gauss-Hermite quadrature rule. An instance of MixedGrid class can consist of different one dimensional grids and has points coordinates which would correspond to grid generated by tensor product applied to the attribute 1D grids. The class GlobalTrapeziodalGridWeighted contains implementations of trapezoidal grid which can be applied for weighted integration by choosing the splitting point in the subinterval [a,b] so,that the obtained halves are equally weighted with respect to a given weight function w(x); the class also contains method of computing quadrature weights. The implementation does not allow to use arbitrary distributions, Uniform and Normal distributions are accepted by the framework. More detailed information about implementation of the class can be found in [13]. This class allows to apply modified basis functions for weighted integration with piecewise linear basis functions, if uncertain parameters are uniformly distributed.

The class GridOperation allows to choose an operation to be performed on grid, one of the possible choices are Integration and UncertaintyQuantification, a class which contains among others a method for computing moments of a random variable X. From the first and second moment one can obtain the expectation and variance of X.

A Function class defines a common structure and methods of a function in framework, which can be extended by subclasses. One of the attributes of a function object is *f dict*, a dictionary containing previously evaluated points. The *call* method of the function class uses *f dict* to check, whether the function was previously evaluated at the given coordinates. If the value is abscent in the dictionary, then the function is evaluated and its value is stored in the dictionary. The class spatiallyAdaptiveSingleDimension2 implements methods necessary for the spatially adaptive algorithm described in chapter 2.0.6, among which are method for rebalancing of a refinement tree and multiple strategies for obtaining a subtraction value $\tilde{c}_j^k$ which might have an influence on the choice of points in component grids of the combination scheme. One of the parameters which an instance of the spatiallyAdaptiveSingleDimension2 class can take are a parameter *norm*, according to which a vector with error estimations is transformed into a scalar value and an integer number for version of the strategy for obtaining a subtraction value.

The framework also implements The Standard Combination Technique which allows to

[1]https://github.com/obersteiner/sparseSpACE.git

apply the chosen grid operation to the grids in the scheme with given lmin, lmax and type of used grids. The parameters lmin and lmax can be used to regulate amount of grids in the scheme and the number of used points. If one sets lmin=lmax, then the combination scheme will consist of one grid with level vector $\vec{l}$ such, that every entry in $\vec{l}$ equals to lmin. Such full grids can be used for computing integral approximation of Gaussian quadrature, if the same number of points should be applied in every one dimensional grid tensor product of which forms the desired grid. However, the number of points for one dimensional Gauss-Hermite and Gauss-Legendre grid would be computed by $2^{level} - 1$ which restricts the possible choice of number of nodes per dimension. An instance of the class SpatiallyAdaptiveBase can become as initialization parameters lmax and lmax, these parameters would define from which grid the algorithm from the chapter 2.0.6 will start adaptive refinement. One can also give error tolerance and maximal number of points such that, after determining that current number of points in the grid greater, than given number, the algorithm stops further refinements.

6 Python implementation

6.0.1 VadereSimulation class

The simulation of campus utilisation was added to the sparse-SpACE framework in form of a subclass *VadereSimulation* which inherits from the *Function* class. By initialization an object of the class *VadereSimulation* can take two parameters: path to java, giving the location of the Java Virtual Machine, and path to executable, giving path to the Vadere console file. The scenario files is assumed to be stored in the *scenarious master* directory and outputs are written to the *output* directory. By initialization is checked whether the respective directories exist, if the directories do not exist then they are created. The Scenario class is responsible for creating an input file containing simulation parameters for the Vadere-console including the values of uncertain parameters. An instance of the Scenario class can take by initialization 4 arguments: a list with values for the uncertain parameters, number of persons in the simulation, finish time and the the time step length. The simulation parameters are stored in a dictionary; the class has a function *writeScenario* which allows to store the data in the dictionary to the hard disk drive by creating with help of the *dump* function from the json library a file with scenario parameters that can be used as an input by the Vadere-console. A Scenario-class is restricted by the implementation to the Campus Utilisation scenario and cannot generate for an arbitrary simulation which can be computed by the Vadere-framework a dictionary with the necessary parameters. The *eval* function of the "VadereSimulation" subclass overrides the respective function from the Function class. The function uses a tuple of coordinates which is an argument of the function to create an object of a class Scenario with scenario parameters.

With help of the *Popen* -constructor from the *subprocess* library a new child process is created to execute the java program corresponding to the Vadere-framework. The size of a heap which can be used by the JVM is restricted to 4096 mbytes in order to reduce memory usage. The arguments passed to the *Popen* -constructor include paths to the Vadere-console and to the java library. The parent process waits until the child process is terminated by using the wait method from the Popen class. The Vadere-console creates a directory with simulation results, it contains file "timesteps.txt" in which for each time step the amount of pedestrians at each of the 9 measurement points is written. The call method of the Function class was modified: if the function values are not found in the dictionary *f dict*, then it is checked if the file "timesteps.txt" corresponding to the given values of the four uncertain parameters exists. If the searched file is present, then the algorithm will try to obtain values which are supposed to be written in "timesteps.txt". The function is responsible for converting the data contained in the file "timesteps.txt" into a one dimensional numpy array which can be returned by the eval function via *get calculated eval*. The function can take as an argument a boolean variable *use counter* which indicates whether the function was called from the *call* function or not. The function contains a variable counter and a loop; after entering the loop's body the value of the argument *use counter* is checked. In the body of the loop the function "genfromtxt" from the numpy library is used to obtain a numpy ndarray with shape (1113,10) from the "timesteps.txt"-file should the file be fully written. If there occur *ValueError* or *IndexError* exceptions, then the process waits one second, increases the value of the counter-variable by one and attempts to execute the *genfromtxt*-function one more time by reentering the body of the loop. If the function was called from the *call* function and the process have waited at least five seconds, then it is interpreted as a previously not evaluated point by the call function and the *get calculated eval* function will return an empty list. In the obtained matrix 1113x10 the data indicating time steps to which the values were computed is excluded by applying slicing to the numpy array and then the filtered matrix is transformed into one-dimensional array which is returned as a result by the *eval* function.

6.0.2 Parallelisation

The time needed for termination of one Campus Utilisation-simulation was measured on the CoolMUC2 cluster for multiple test-runs, for the sequence of performed experiments the time values in seconds in the most cases were an element of [55, 75] interval. The results of integral approximation for the four-dimensional function for several quadrature rules were required and for each quadrature rule several integral-approximations were computed with different numbers of evaluation points, including Gaussian quadrature rule with exponential dependency between the number of points per one-dimensional grid and the dimension of the function. In order to compute integral approximations with high amount of evaluated points the integral approximations were computed on CoolMUC2 cluster by using a group of processes which allows to run the simulation on a cluster with distributed memory. Functions from the *mpi4py* library were utilized to distribute a list of points to evaluate among the processes in form of messages, thus allowing to significantly reduce the computational time by parallelising points evaluation which in case of the Campus Utilisation-simulation is one of the most computationally expensive part in the computation of the integral-approximation. Evaluated points are stored on the hard disc drive, if the program, which computes the integral-approximation, was interrupted before the approximation value was computed, then by restarting of the program reuse of already computed points can reduce time and memory requirements needed for termination of the program. From a group of processes every process except the one with the rank zero was used only to evaluate every point in a given list. Only the process with rank zero builds during its execution the data structures and performs machine instructions other,than needed for evaluation of points, but corresponding to the steps of an algorithm used to compute the integral approximations. The goal of such implementation was to reduce memory usage of processes with $rank > 0$ in order to increase the number of processes which run on the same cluster-node. The processes with $rank > 0$ wait in a while loop for the messages from the process with rank 0 and depending on the type of the received data either terminate or evaluate the obtained points. The function *start parallel evaluation* implemented in the GridOperation class becomes as argument a list of coordinates of points. It calls the function *get new points* which checks for each point whether the directory identified by the coordinates of the point exists allowing to avoid recomputation of already evaluated points in previous program runs. If the required directory does not exist then the point is added to a set consisting of points which will be evaluated, a set does not contain duplicates of points. The function returns the set with coordinates converted to a list. If the list contains at least on element, then the process with rank 0 divides the list in blocks, the length of which differs at most by one, and sends the obtained lists of points with help of *send* function from the *MPI.COMM WORLD* class to the processes in the group. If the number of processes is greater,than the number of blocks, then an empty list will be sent to the remaining processes. Points are evaluated by using the function eval in order to avoid modification of *f dict*. The process with $rank > 0$ checks the type the received data from the process 0, if the data is of type integer, then it serves as an indication that the integral approximation is computed and stored by applying the function save from the numpy library to the array with computed result, in that case all remaining processes will leave the loop and terminate. If the received data is of type list, then in case of non-empty list points will be evaluated with the eval function and processes will send boolean value-indication that received data is processed- to the process with rank 0. The process with rank zero will wait until the messages from every remaining process arrives and only then continues computation of the integral approximation. After the necessary values are computed the zero process sends an integer value to every process with $rank > 0$.

In case of the Monte Carlo algorithm for a given N as the number points to evaluate coordinates of points were generated by the sample- method of distribution class from the chaospy library and stored as numpy array; implemented method montecarloParallel by computing the integral approximation with N points according to the Monte Carlo algorithm accessed the stored array and evaluated points on the coordinates of the array using the implemented function *start parallel evaluation* after filtering the duplicates of points. Stored coordinates allow to avoid resampling of N points for a fixed number N should the program exit with an exception before storing the computed result and should a restart of the program be necessary. However, for a number $N' > N$ the coordinates of the points are newly generated which can lead to disjoint sets of points for N and N′. Coordinates of points and weights for Gaussian quadrature were generated by the function *generate quadrature* from the chaospy library. For the obtained coordinates filtering of point duplicates can be spared since the roots of the used orthogonal polynomials are distinct. The points are evaluated by using the same function as in case of the Monte Carlo algorithm *start parallel evaluation*.

6.0.3 Reading a file

Both scenario and "timesteps" files are written to the hard disc drive and are accessed during function evaluation by the program. The situations were observed, when, for example, the child process have terminated and the parent process tries to read from the file "timesteps.txt",but the information is not yet contained on the hard disk drive. The parent process waits for termination of the child process and only loads the computed values after receiving a confirmation, that every maschine instruction of the child process have been processed including those responsible for storing the data on the disc. If the data is not present on the disc and the program tries to access it, then there can occur an exception resulting in program termination. In order to reduce the probability of such outcome the method *check file written* was implemented in the VadereSimulation class. During the function evaluation *check file written* is applied before the child process is created to check if the scenario is fully written and before the function *get calculated eval* tries to access the "timesteps.txt"-file. The function *check file written* contains two different approaches each of which is based on the known the structure of the file the algorithm. Therefore one of the arguments the function receives is the *mode* -argument with two possible values "timesteps" and "scenario" indicating on which type of file the function must be applied.

In case of scenario the dictionary with data which should be written to the file is known for the algorithm. The dictionary with simulation parameters can be obtained from the Scenario object created in the *eval* function and passed as an argument to the *get calculated eval*-function. The function contains a loop with the exit condition, that the given relative path exists, in the body of the loop the sleep-command is executed from the time library with argument 1 second. After it is confirmed, that the file exists the algorithm enters the second loop where in case of scenario the data from the scenario-file are obtained by using the *json.loads* function. It is expected that the function *json.loads* will return a Python dictionary; two possible exceptions *ValueError* and *json.JSONDecodeError* are handled by forcing the process to wait for 1 second and then the process reenters the loop. If the loads-function returns a dictionary object, then the obtained and the expected dictionaries are compared. In case of inequality of the dictionaries the process sleeps for one second and reenters the loop.

If the value of the *mode* argument was "timesteps",then a different strategy is deployed: the values which should be written to the "timesteps.txt"-file are computed by the Vadere framework and the parent process does not become this values from the *stdout* of the child process. For a given scenario it is known, how many measured values must be in the file, in which interval their values must be contained and of what type the obtained data must be. As a result, if the function *get calculated eval* returns a numpy array, then the algorithm checks the length of the array and for each value contained in it is checked its type and whether its value is contained in the correct interval. If one of the checks fails the loop is reentered and the function *get calculated eval* is called. This approach does not give a guarantee, that the obtained data is the same as was written by the child process; it increases the probability, that the false data will be recognized.

7 Tests

7.1 Convergence of quadrature rules and error estimations

If a deterministic algorithm is given, which operates on discrete set of machine numbers and one can use this algorithm to define a function from the non discrete set of its arguments into the value computed by the algorithm, then the obtained function might not be Riemann integrable. A trapezoidal sum builds a piecewise linear interpolant. If one denotes with h the maximal distance between grid points in the trapezoidal sum, then according to [2] should the upper and lower Riemann sums converge to the integral of a given function $f \in C^0[a,b]$, then the trapezoidal sum will converge as well under condition that $h \to 0$ for$n \to \infty$, because the following equality holds:

$$R^n_{lower} \le T^n \le R^n_{upper}$$

with T^n is a trapezoidal sum consisting of n linear functions. However, this is an asymptotical statement, if the performed sequence of experiments is finite,then the possibly observed approaching of computed results to some value might not indicate, that the value is the integral and that, the integral exists.

For some of the described in the previous chapters quadrature rules the guarantee that the method will converge and how fast in dependency from the used number of evaluation points it will happen is given under assumption the integrated function f belongs to a certain class, for example, a function with bounded second mixed derivative or for $f \in C^{2n+2}$ in case of one dimensional function f for Gauss-Christoffel-Quadrature. In case of the function $\hat{f}$ it is not known,whether it belongs to the required classes, as a result the convergence rates given for some quadrature rules in previous chapters might not hold for the $\hat{f}$ function. If for the error ϵ_N with N as the number of used points N an upper bound $c \cdot g(x)$ with constant c,which might depend on the integrated function, is given, as it is the case with sparse grids, the for a finite sequence $(\epsilon_i)_{N \le i \le N+k}$ of the errors of performed experiments the ratio of error decrease between i and i+1 will be bounded from above by g(i+1)/g(i), but if the constant c is large, then despite observing the error decay defined by the function g(x) one might become large errors for a finite sequence of performed experiments. In case of Monte Carlo one has an asymptotic statement about probability of the error being bounded from above by the function $\hat{g}(x) = c1/\sqrt{N}$. It states, that there exists point N_0 starting from which the probability of the error is bounded from above by $\hat{g}(x)$ can be estimated with an arbitrary precision by using the values of probability density function of normal distribution, but it does not give the point N_0. Therefore a longer sequence (ϵ_i) might have a higher probability, that its subsequence $(\epsilon_{i'})$ is in the convergence area.

Large number of evaluation points of a vector-valued function $\vec{f}$ might lead to the lack of memory during the computation of the integral approximation. One possible solution might be, to split the vector, compute in multiple program runs the approximations for the obtained vectors of smaller dimension, and then to unite the obtained vectors into one which then will be used as the integral- approximation for the function $\vec{f}$. This approach bases on the following property according to [14] of Riemann-integral for vector-valued functions: if

$$f(\vec{x}) = (f_1(x), ..., f_n(x))^T$$

then

$$\int_\Omega f(\vec{x})d\vec{x} = (\int_\Omega f_1(\vec{x})d\vec{x}, ..., \int_\Omega f_n(\vec{x})d\vec{x})^T$$

The Dimension-wise spatial refinement with the Sparse Grid Combination Technique chooses points to refine according to the values of the error estimator. For a vector-valued function a vector containing the error estimations is transformed with help of the user defined norm into a scalar value and with help of this value the choice of refinement points is made, such approach does not guarantee,that the refinement points are chosen optimally for each f_i in the vector $\vec{f}$ but it reduces the amount of points needed to compute the integrals $\int_a^b f_i$ for $1 \leq i \leq n$. Thus splitting vector might result in another choice of evaluation points by computing the integral-approximation for the subvectors which in turn might cause the difference between the values of the combined vector and the vector which was computed without splitting into subvectors.

If the reference solution for the integral of $\hat{f}$ is not given, then one could use an error estimator to approximate the errors ϵ_i. An error estimator constructed as shown in (1) gives good approximations if the condition $\epsilon_1 << \epsilon_2$ is met, otherwise it might give estimation which greatly differ from the value of ϵ_2. It is assumed, that an integral approximation which uses more points, as a result having more information about the behaviour of the integrated function, has higher accuracy compared to the approximation with less evaluation points computed by the same quadrature rule. However, that might not be the case for an arbitrary function f and the used numbers of evaluation points N_1 and N_2. In that case the term ϵ_1 might become dominant in the error estimation or the estimated value might become very small if both errors are of the same order.

For an arbitrary integrable function a finite sequence of integral approximations computed by a given quadrature rule might approach some value k, but it is not guaranteed, that the value equals to the value of I [f], if the reference solution is not given. In order to increase the probability,that the value k to which the integral approximations get closer with the increasing number of N can be interpreted as I[f] one could use several different quadrature rules to compute the approximations. If most of them approaching to the same value k, then the probability is higher, that k is the searched value of I[f]. For an error estimator of the form (1) one could also use the solutions of the methods approaching k with the highest numbers of points as a comparison value for the computed results of a method of interest. Depending on the obtained results for different methods one might compare the approaching rates to the value k for different methods with each other.

7.1.1 Norms

A vector-valued function

$$f(x) = (f_1(x), ..., f_n(x))^T$$

consists of scalar valued functions $f_i(x)$ which can differ significantly from each other. As a result, by non-adaptive quadrature rules one might observe, that by using the same number and position of points for functions $f_i(x)$ for$1 \leq i \leq n$ the integration error can be might be of different order depending on the function $f_i(x)$ and on the quadrature rule. The spatially adaptive algorithm described in chapter 2.0.6 applies a user defined norm on a vector with estimated errors for functions $f_i(x)$. In case of Campus Utilisation scenario on Fig 7.1 one might observe different behaviour of the error estimators for $f_4(x)$ and $f_{3198}(x)$ with the first function corresponding to number of people in IPP Mensa at the first time step and the second the number of persons in TUM Mensa at the time step 355. Since at the first time step pedestrians are spawned at the underground, $f_4(x)$ is a constant function with value zero and is integrated exactly by the used quadrature rules. The second function $f_{3198}(x)$ is influenced by the values of uncertain parameters and for the comparison value was taken the integral computed by applying Quasi Monte Carlo method with Halton sequence with 28561 evaluation points. One can observe oscillations with different amplitude for different quadrature rules and slow decay of estimated error in case of Monte Carlo, Gaussian quadrature and spatially adaptive algorithm with lmax=4. For spatially adaptive algorithm with lmax=2 one can observe oscillations around approximately the same value. Slow decrease might be the sign of very slow convergence of the selected methods or the sign, that the used comparison value has larger error, than the integral approximations computed by other quadrature rules, and becomes dominant in the error estimation. In case of different integral-approximation quality for different $f_i(x)$ one possible solution would be to optimize selection and amount of points for each $f_i(x)$ for$1 \leq i \leq n$, however, if the dimension of the vector f(x) is large, then such approach might result in a high amount of evaluation points.

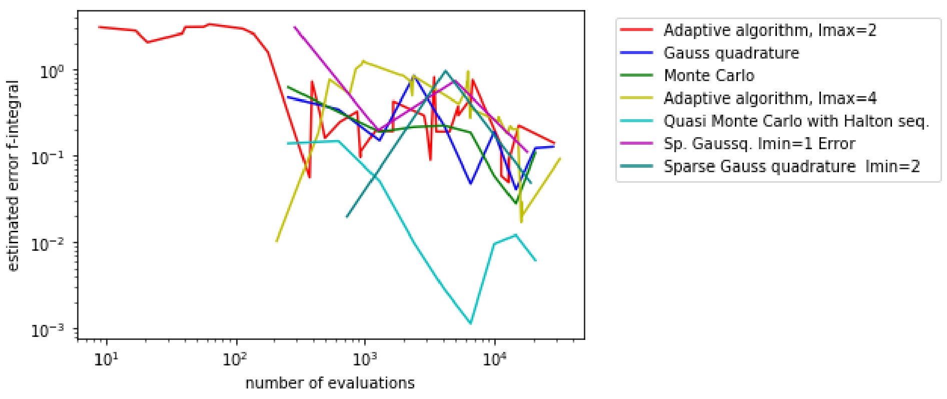
If the dimension of the output vector is large one might be interested in scalar values describing the combined quality of the obtained integral approximations. For that purpose one might use different norms depending on what the scalar value should describe. The L^1 norm applied to the vector x is defined as:

$$|x|_1 = \sum_{i=1}^{n} |x_i|$$

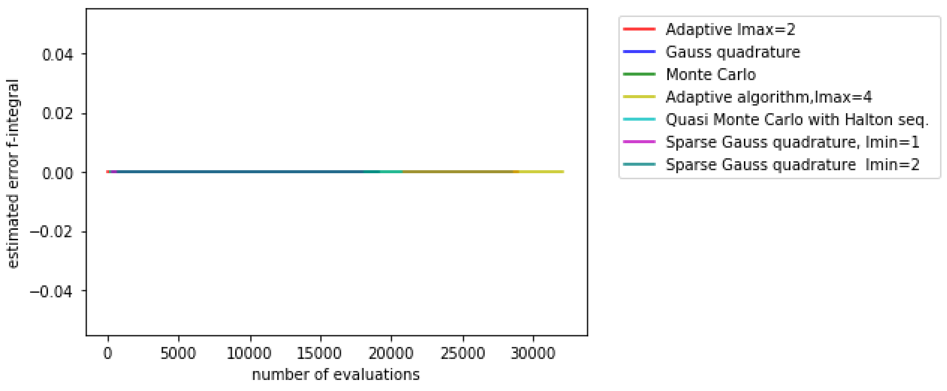
The mean error estimation can be calculated as: $|x|_1/n$. The L^2 norm is defined as

$$|x|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}$$

and can be interpreted as length of vector x.



(a) $f_{3198}(x)$



(b) $f_4(x)$ where each method has integrated exactly the constant zero

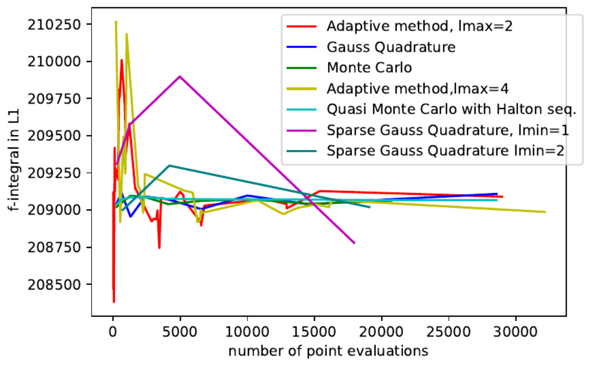
Figure 7.1

7.2 Test results

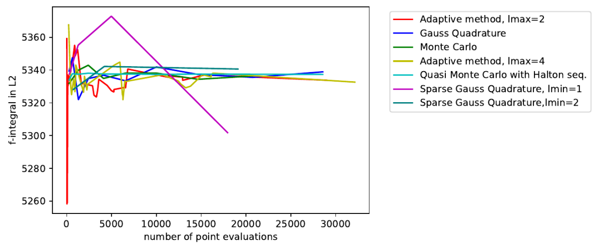
For the computed tests the weighted trapezoidal grids were used for the spatially adaptive algorithm. Boundaries represented with the additional basis functions with level zero were not applied in order to reduce the computational effort and spend more points in the inner part of the domain. For Gauss quadrature the same number of points was used for every dimension. Number of points used for Monte Carlo and Quasi Monte Carlo methods corresponds to the overall number of points used in the grids for Gaussian quadrature. On Fig.7.2 there are presented the computed values of integral approximations with applied L^1 and L^2 norms on the computed vectors. Several quadrature rules were used in order to compare results. As seen on the graph most methods approach the same value in L^1 and L^2 norms. Sparse Gauss quadrature applied with grid constructed with lmin=1 shows oscillation with high amplitude over a longer interval compared to other methods. The Gaussian quadrature shows oscillations for low polynomial orders. The oscillations of Sparse Gauss quadrature might be caused by using the low order polynomials for interpolation along some dimensions in the grids, which might result in a large quadrature error. Sparse Gauss Quadrature with lmin set to 2 can exclude depending on the parameter lmin grids from the scheme which have low resolution across at least one dimension. As a result one might obtain denser grids in the scheme the higher the parameter lmin is set. The Sparse Gauss quadrature with lmin=2 shows oscillations with lower amplitude compared to the graph constructed with lmin=1 and might approximate the same value as Quasi Monte Carlo method. However, for both lmin=1 and lmin=2 low number of experiments with different numbers of evaluated points compared to other methods were performed due to fast growth of number of needed points to compute a solution with increasing lmax parameter of the combination scheme. The discrete function constructed with results of Quasi Monte Carlo method with Halton sequence shows almost no oscillations in both norms, this might be an indication, that the method converges fast to the value, which most of the quadrature rules approach. If two functions optically approach the same value on a graph, they might have different rate of error decay, therefore the quality of the computed solution by these methods might be different depending on tolerated error. The graph constructed by the spatially adaptive algorithm, which used lmax set to 4 as the start grid deviates in L^1 and L^2 norms from the value to which Quasi Monte Carlo method approaches. It might indicate, that the method converges, but slower in comparison to Quasi Monte Carlo method.

7.2.1 Values of error estimators

In this chapter the values of error estimators are presented. The spatially adaptive algorithm has its error estimator described in the chapter 3 implemented in the sparse-SpACE framework. The framework can return a dictionary containing vectors with estimated errors for each refinement step. For other methods an error estimator is constructed according to (1) with comparison values chosen as the solution which was calculated by method with the highest number of evaluation points. The results of Sparse Gauss quadrature are excluded from the comparison values due to low number of experiments and possible slower



(a) Computed integral approximations in L^1 norm



(b) Computed integral approximations in L^2 norm

Figure 7.2

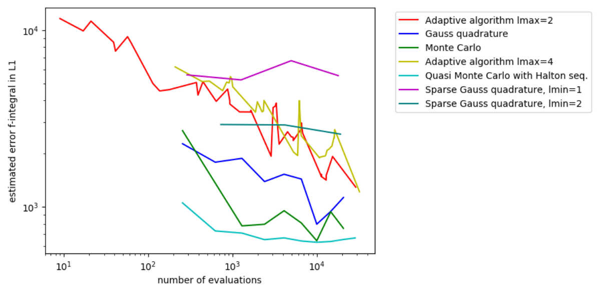
convergence based on the observations from the previous chapter. To the values of error estimators the L^1 and L^2 norms are applied. The graphs are plotted in the log-log scale in order to highlight the order of decrease of the estimated errors, should that decrease occur. Additionally an error estimator built as in (1) was applied to the results of the spatially adaptive algorithm. As comparison values were chosen the values computed with the highest amount of points by Quasi Monte Carlo, Monte Carlo, Gauss quadrature and spatially adaptive algorithm with lmax=2, as this values might converge to the same value and show oscillations with small amplitude. The obtained graphs are shown on Figures 7.3, 7.4 and 7.5.

On each of the three graphs, excluding 7.4 c) and d), Quasi Monte Carlo shows the smallest estimated error. If the value computed by Gauss quadrature rule or by spatially Adaptive method with lmin=2 is used as a value to compare, then the decrease of estimated error for Quasi Monte Carlo becomes slower compared to the other error estimators. This might be an indication, that Quasi Monte Carlo outperforms Gauss quadrature and spatially adaptive method and produces much smaller error. The solutions computed by Monte Carlo method show oscillations and slow decrease of value in error estimation computed with Gauss quadrature solution. Such behaviour of the graph might be caused by approximately the same order of errors which the quadrature rules or by oscillations of Monte Carlo quadrature. Based on the results of error estimator with Quasi Monte Carlo comparison value with Monte Carlo having faster decrease in values and smaller values compared to Gauss quadrature. As a result, Monte Carlo method might have computed a better approximation, than Gaussian quadrature. Both versions of the spatially adaptive algorithm with lmax=2 and lmax=4 show higher estimated error values, than the other methods and oscillations are present. Oscillations do not exclude the convergence of a method,however their amplitude must decrease if a sequence of experiments converges for $N \to \infty$ to the integral of the function. The observed amplitudes in case of Quasi Monte Carlo comparison value are shown lower compared to the results of Gauss comparison value. This might also indicate, since the value of Quasi Monte Carlo is assumed to be the closest to the integral value, that the used error estimator, developed for the spatially adaptive algorithm, shows more accurate results, than the error estimator with Gauss comparison value applied to the results of the adaptive algorithm. The error estimator of the spatially adaptive algorithm shows slower decay of the error, but low oscillations amplitude, which is more similar to the values obtained with Quasi Monte Carlo comparison value. The behaviour and values of a graph corresponding to spatially adaptive algorithm with lmax=4 does not significantly differ from the graph of the algorithm with lmax=2. The change of level of a start grid leads to different child nodes in refinement trees, higher level lmax might cause better initial approximation of the integration domain for the adaptive algorithm and increases the probability to refine small regions with highly localized behaviour of the function. Similarity in the estimated errors might be a result of both versions of the algorithm placing the points across the entire domain and not in some selected subareas with highly localized behaviour of the integrated function. This might indicate,that such subareas are small, if they exist, and their number is not high.

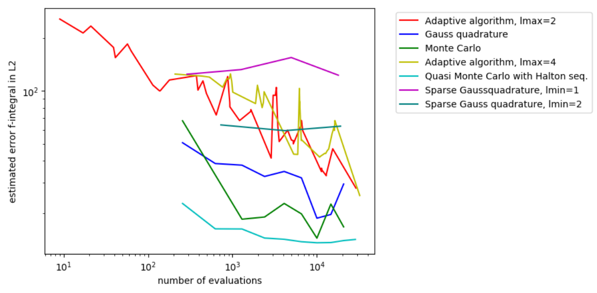
7.2.2 Refinement graph

The sparse-SpACE framework allows to plot the one dimensional refinement trees of spatially adaptive algorithm. In Fig 7.6 are depicted the refinement trees which the algorithm have made on the last refinement step before computing and returning the integral approximation. The trees were built by the algorithm with lmax=2 and lmax=4. According to the graphs the algorithm does not refine intensively along dimensions which correspond to the uncertain parameters with Normal distribution. It might indicate, that the estimated error along these dimensions is small compared to the dimensions with uniformly distributed parameters. In the x_1 and x_4 dimensions there are grouping of points near the boundary of the domain.

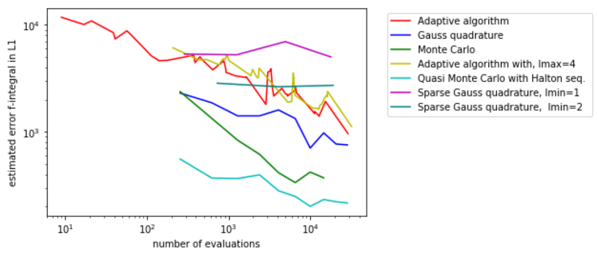
In the performed tests the boundary was set to zero. The inner part of the integration domain was according to the graphs also intensively refined. In order to investigate how much impact on placement of points the zero boundary has, there were performed tests with modified basis functions with lmax=2. On Fig 7.7 and 7.8 are shown error estimations for the computed results and on Fig 7.9 the refinement graph for a step of the algorithm with 16275 evaluated points. For error estimators with comparison values computed with Quasi Monte Carlo and Monte Carlo error estimations show deviations from the values computed for adaptive algorithm with lmax=2. On the refinement graph one might observe, that the points are still added to the graph on the areas which are close to the boundaries. In the dimension x_4 the interval [1.3,1.4] is intensively refined by the algorithm independently



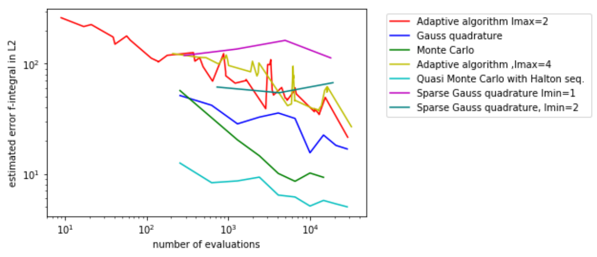
(a) Error estimator values for Gauss comparison value in L^1 norm



(b) Error estimator values for Gauss comparison value in in L^2 norm

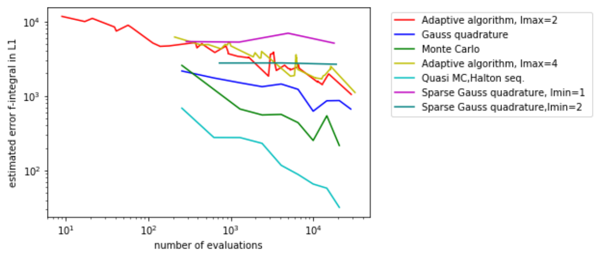


(c) Error estimator values for Monte Carlo comparison value in L^1 norm

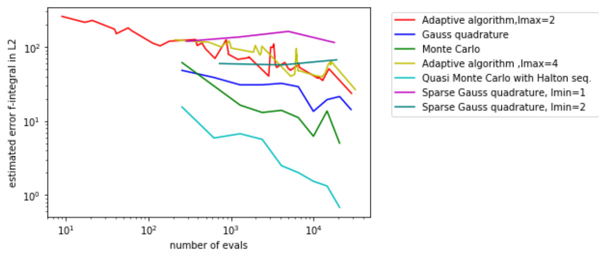


(d) Error estimator values for Monte Carlo comparison value in in L^2 norm

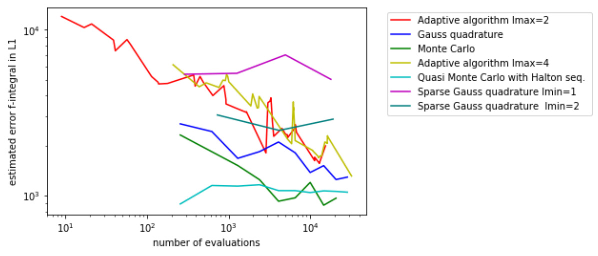
Figure 7.3



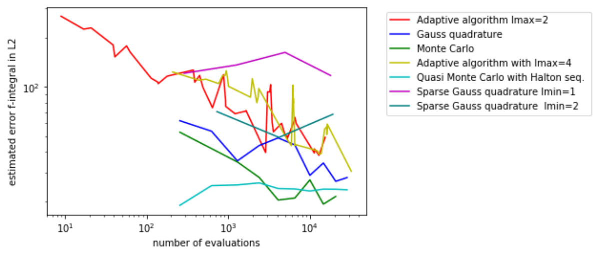
(a) Error estimator values for Quasi Monte Carlo comparison value in L^1 norm



(b) Error estimator values for Quasi Monte Carlo comparison value in in L^2 norm

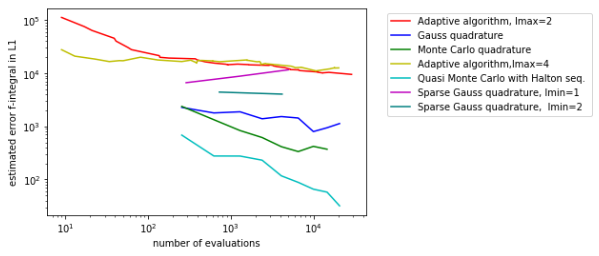


(c) Error estimator values for the adaptive algorithm with lmax=2 comparison value in L^1 norm

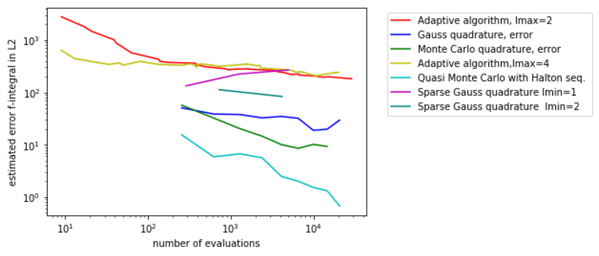


(d) Error estimator values for the adaptive algorithm with lmax=2 comparison value in L^2 norm

Figure 7.4



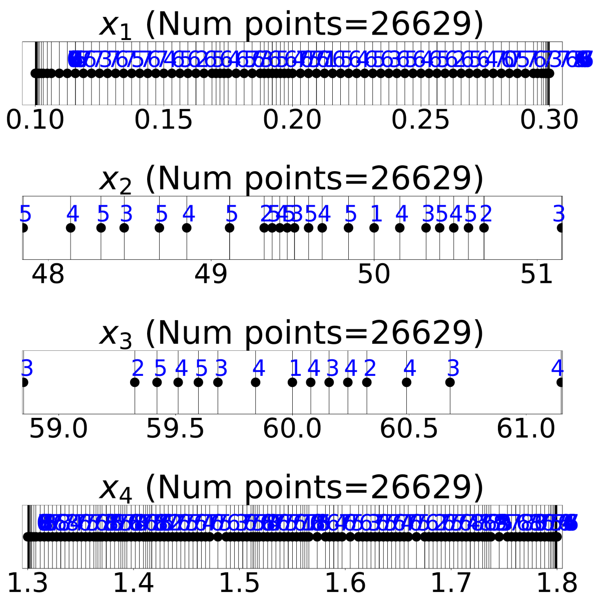
(a) Error estimator values with each quadrature rule using its value computed with the highest number of evaluation points as comparison value in L^1 norm



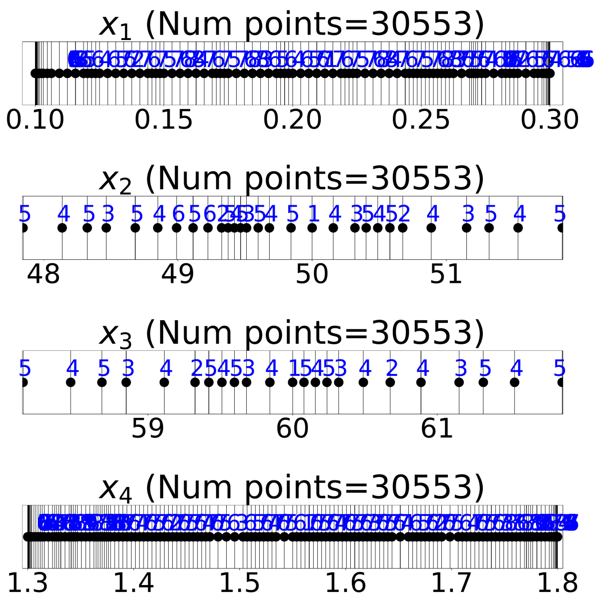
(b) Error estimator values with each quadrature rule using its value computed with the highest number of evaluation points as comparison value in L^2 norm

Figure 7.5

of using modified base to approximate the values of a function on the boundaries. This might have been caused by highly local behaviour of the function in that area so that using extrapolation to approximate the function values on borders and in the areas near the borders might result in large integration errors in the areas near the boundary which might come from strongly differing values of a function in that part of the domain. As a result, by spending more points near the boundaries one might obtain a more accurate approximation of the integral.

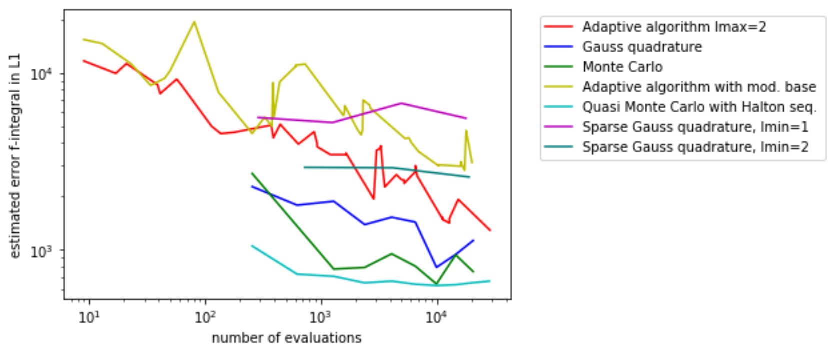


(a) Refinement graph for the adaptive algorithm with lmax=2

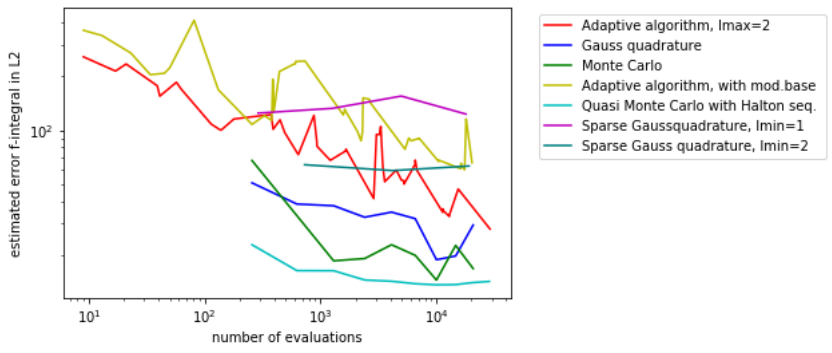


(b) Refinement graph for the adaptive algorithm with lmax=4

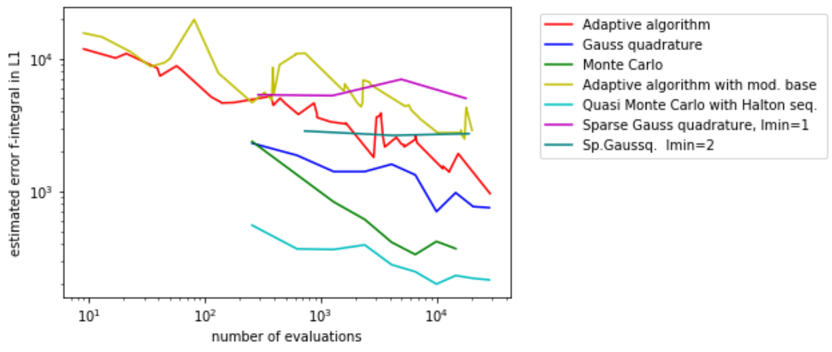
Figure 7.6



(a) Error estimator values for Gauss comparison value in L^1 norm for modified base

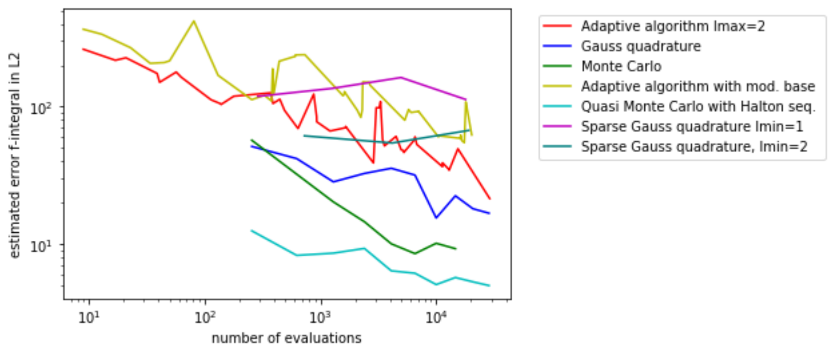


(b) Error estimator values for Gauss comparison value in in L^2 norm for modified base

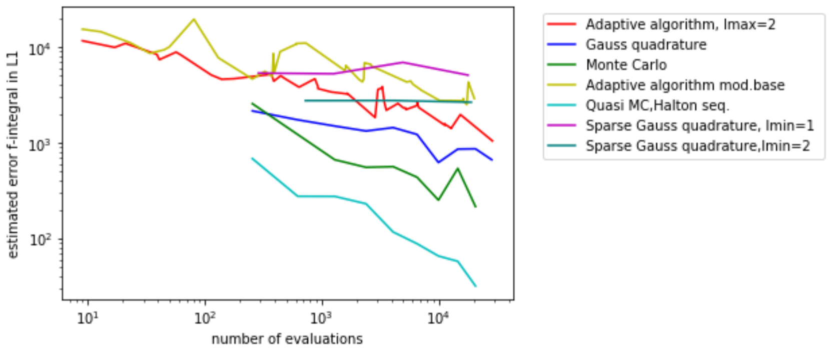


(c) Error estimator values for Monte Carlo comparison value in L^1 norm for modified base

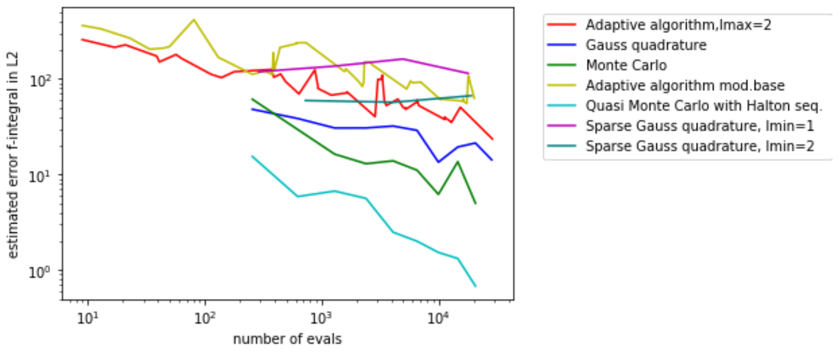
Figure 7.7



(a) Error estimator values for Monte Carlo comparison value in in L^2 norm for modified base

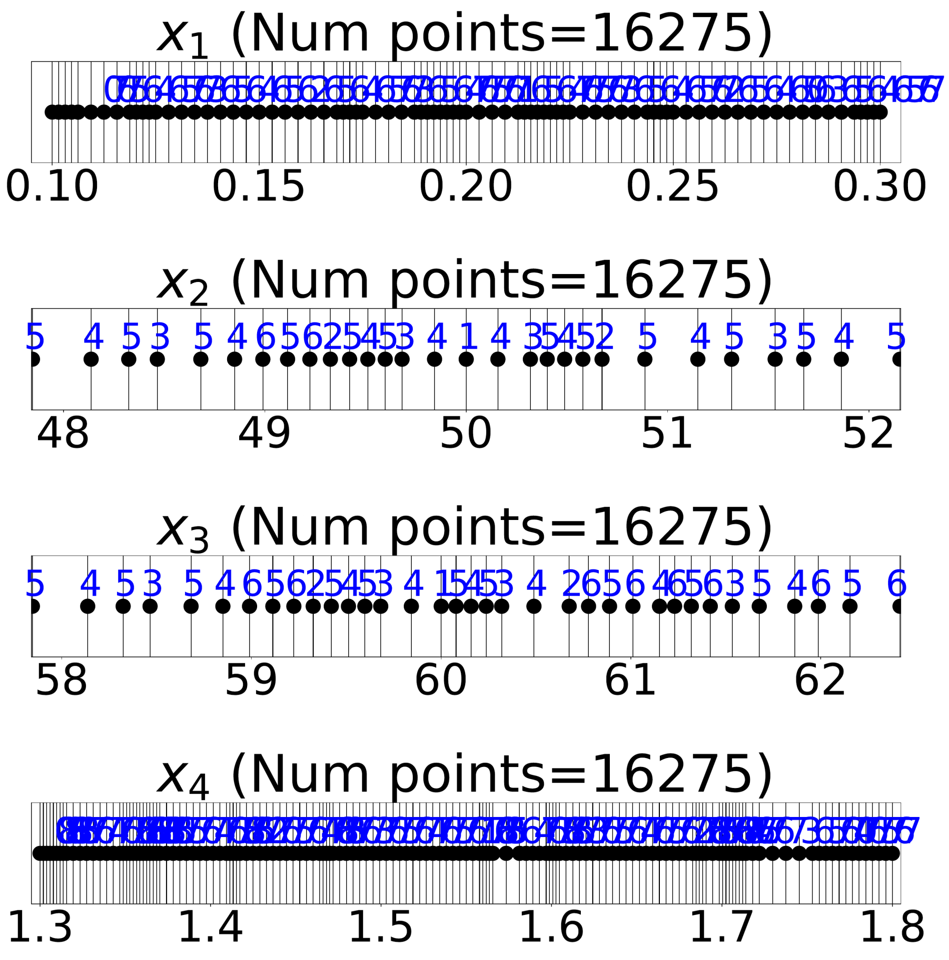


(b) Error estimator values for Quasi Monte Carlo comparison value in L^1 norm for modified base



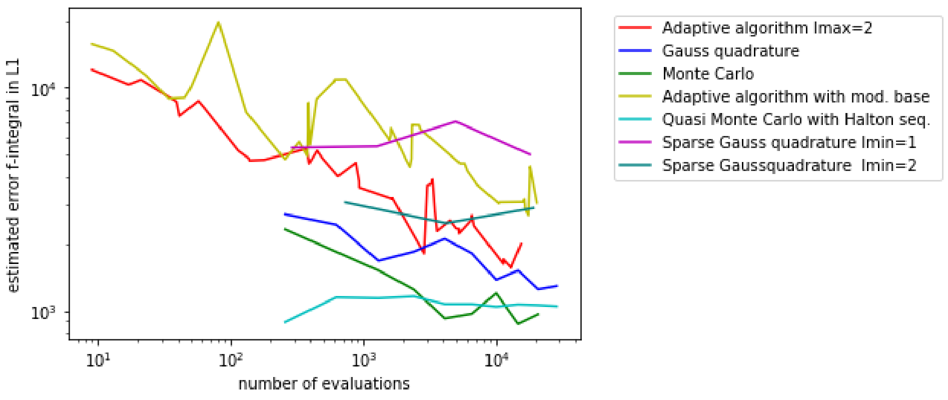
(c) Error estimator values for Quasi Monte Carlo comparison value in in L^2 norm for modified base

Figure 7.8

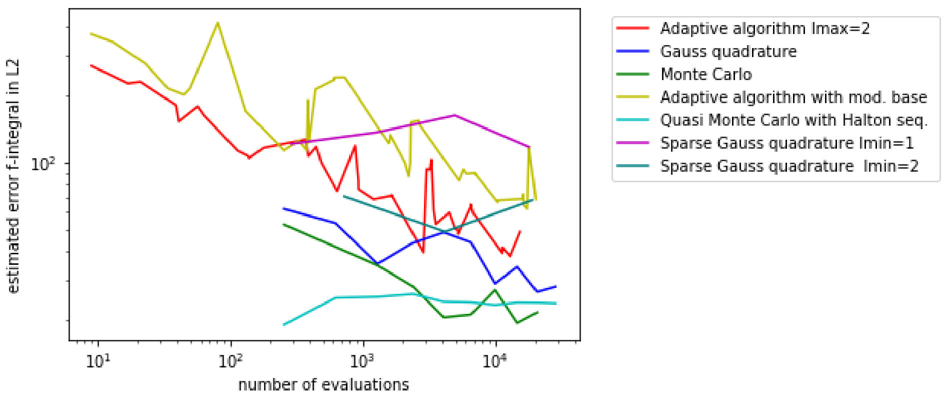


(a) Refinement graph for the adaptive algorithm with modified base and lmax=2

Figure 7.9



(a) Error estimator values for the adaptive algorithm with lmax=2 comparison value in L^1 norm for modified base



(b) Error estimator values for the adaptive algorithm with lmax=2 comparison value in L^2 norm for modified base

Figure 7.10

8 Conclusion

Based on the results of the performed tests, used quadrature rules and requirements for the rules to provide a good approximation of the integral one might make assumptions about properties of the integrated function, if the function is integrable. Monte Carlo and Quasi Monte Carlo have shown the best convergence rate among the used rules. Better results by using piecewise low order polynomials, such as constant function, for computing the integral-approximation compared to the results of Gauss quadrature might indicate, that the function is not sufficiently differentiable for Gauss quadrature. The results from the refinement graphs can be interpreted as a better approximation by hierarchical hat basis function across the dimensions x_2 and x_3 compared to the x_1 and x_4. Placement of points across the whole interval without choosing one particular subarea with the most of the used points might be a consequence of a smooth behaviour of the function in the integration domain with possible exception near the boundaries of the integration domain.
The adaptive algorithm has shown the decay of estimated errors for every considered error estimator with different comparison values including the values of Quasi Monte Carlo, which presumably has the lowest errors among the used methods. Its error decrease rate is more similar to the rate of Gauss quadrature rule, the estimated error values are bigger, than the values for Gauss, Monte Carlo and Quasi Monte Carlo methods. According to the error estimations with Monte Carlo and Quasi Monte Carlo the estimated errors of the adaptive algorithm are more similar to the estimated errors of Gauss quadrature, than to the values of Monte Carlo and Quasi Monte Carlo. This might indicate, that Gauss Quadrature and the adaptive algorithm achieve approximately the same quality in the approximation of the integral. The Sparse Gauss quadrature has similar or worse values of error estimations compared to the adaptive algorithm. Since the effectiveness of sparse grids with respect to L^2 and L^{max} norms might depend on the smallest upper bound for the second mixed derivative of a function and refinement graphs show the refinement near the boundary for some dimensions, it might be a sign, that the function does not have second mixed derivative defined on the integration domain or its maximal value is large. Oscillations in the values of error estimations might be caused by the used error estimator for selection of new points to refine.

List of Figures

Bibliography

[1] D. Xiu. “Fast numerical methods for stochastic computations: a review”. In: *Communications in computational physics* 5.2-4 (2009), pp. 242–272.

[2] P. Deuflhard and A. Hohmann. *Numerische Mathematik 1: eine algorithmisch orientierte Einführung*. Walter de Gruyter GmbH & Co KG, 2018.

[3] R. E. Caflisch. “Monte carlo and quasi-monte carlo methods”. In: *Acta numerica* 7 (1998), pp. 1–49.

[4] A. Sokol and A. Rønn-Nielsen. *Advanced probability*. `http://web.math.ku.dk/noter/filer/vidsand12.pdf`. 2013.

[5] L. Kuipers and H. Niederreiter. *Uniform distribution of sequences*. Courier Corporation, 2012.

[6] H.-J. Bungartz and M. Griebel. “Sparse grids”. In: *Acta numerica* 13 (2004), pp. 147–269.

[7] M. Obersteiner and H.-J. Bungartz. “A spatially adaptive sparse grid combination technique for numerical quadrature”. In: *Sparse Grids and Applications-Munich 2018*. 2019.

[8] D. M. Pflüger. “Spatially adaptive sparse grids for high-dimensional problems”. PhD thesis. Technische Universität München, 2010.

[9] T. Gerstner and M. Griebel. “Numerical integration using sparse grids”. In: *Numerical algorithms* 18.3-4 (1998), p. 209.

[10] M. Griebel, M. Schneider, and C. Zenger. “A combination technique for the solution of sparse grid problems”. In: (1990).

[11] M. Obersteiner and H.-J. Bungartz. “A generalized spatially adaptive sparse grid combination technique with dimension-wise refinement”. Unpublished Manuscript. 2020.

[12] B. Kleinmeier, B. Zönnchen, M. Gödel, and G. Köster. “Vadere: An open-source simulation framework to promote interdisciplinary understanding”. In: *arXiv preprint arXiv:1907.09520* (2019).

[13] F. Hofmeier. “Applying the Spatially Adaptive Combination Technique to Uncertainty Quantification”. In: (2019).

[14] J. A. Gubner. “Introduction to random vectors”. In: *Probability and Random Processes for Electrical and Computer Engineers*. Cambridge University Press, 2006, pp. 330–361. DOI: `10.1017/CBO9780511813610.010`.