# Hiding the weights – CBC black box algorithms with a guaranteed error bound

Alexander D. Gilbert, Frances Y. Kuo and Ian H. Sloan

October 9, 2018

### Abstract

The component-by-component (CBC) algorithm is a method for constructing good generating vectors for lattice rules for the efficient computation of high-dimensional integrals in the "weighted" function space setting introduced by Sloan and Woźniakowski. The "weights" that define such spaces are needed as inputs into the CBC algorithm, and so a natural question is, for a given problem how does one choose the weights? This paper introduces two new CBC algorithms which, given bounds on the mixed first derivatives of the integrand, produce a randomly shifted lattice rule with a guaranteed bound on the root-mean-square error. This alleviates the need for the user to specify the weights. We deal with "product weights" and "product and order dependent (POD) weights". Numerical tables compare the two algorithms under various assumed bounds on the mixed first derivatives, and provide rigorous upper bounds on the root-mean-square integration error.

## 1 Introduction

Our aim in the current work is the efficient and relatively painless numerical computation of high-dimensional integrals of the form

$$I_s f := \int_{[0,1]^s} f(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} \,,$$

where $[0,1]^s := [0,1] \times \cdots \times [0,1]$ denotes the $s$-dimensional unit cube. A quasi-Monte Carlo (QMC) approximation of the above integral is an equal-weight quadrature rule

$$Q_{n,s}(P_n) f := \frac{1}{n} \sum_{k=0}^{n-1} f(\boldsymbol{t}_k) \,,$$

where the quadrature points, $P_n := \{\boldsymbol{t}_k\}_{k=0}^{n-1}$, are chosen deterministically from $[0,1]^s$. The setting for the error analysis of such QMC approximations, introduced by Sloan and Woźniakowski [14], assumes that the integrand $f$ belongs to some $s$-variate weighted function space $W_{s,\boldsymbol{\gamma}}$, where in the original formulation each variable has

1

an associated "weight" parameter $\gamma_i > 0$ whose size describes the importance of $x_i$. Weights of this kind $\boldsymbol{\gamma} = \{\gamma_i\}_{i=1}^{\infty}$ are incorporated into $W_{s,\boldsymbol{\gamma}}$ through the weighted norm $\|\cdot\|_{s,\boldsymbol{\gamma}}$ and are nowadays referred to as *product weights*, see Section 2 for more details. In Section 2 we also consider more general weights, that is, a parameter $\gamma_{\mathfrak{u}}$ is allowed for each subset $\mathfrak{u} \subseteq \{1, \ldots, s\}$, but in the Introduction we shall concentrate on product weights.

This paper introduces two new variants of the component-by-component (CBC) algorithm that, given a bound on the norm of the integrand, choose not only the QMC points but also the weights, with a view to minimising the error of the QMC approximation. Although still playing an important role in our constructions, the weight parameters no longer need to be chosen by the practitioner because this choice is handled automatically inside the algorithms.

More precisely, in such a weighted function space $W_{s,\boldsymbol{\gamma}}$, the *worst-case error* of $Q_{n,s}$ over the unit ball of $W_{s,\boldsymbol{\gamma}}$, is defined by

$$e_{n,s,\boldsymbol{\gamma}}(P_n) := \sup_{\|f\|_{s,\boldsymbol{\gamma}} \leq 1} |I_s f - Q_{n,s}(P_n)f| \,,$$

from which it follows by linearity that the error of a QMC approximation satisfies

$$|I_s f - Q_{n,s}(P_n)f| \leq e_{n,s,\boldsymbol{\gamma}}(P_n) \|f\|_{s,\boldsymbol{\gamma}} \,. \tag{1}$$

This error bound is convenient because of its separation into the product of two factors, one which depends only on the quadrature points and the other which depends only on the integrand. A key aspect of the current work is that both the worst-case error and the norm depend on the weights.

In this paper the user is assumed to have information about the norm $\|f\|_{s,\boldsymbol{\gamma}}$ (defined in (4) below) in the form of estimates of the parameters $B_\ell$ and $b_i$ in the following assumption:

**Standing Assumption:** *For two sequences of positive real numbers $(B_\ell)_{\ell=1}^{s}$ and $(b_i)_{i=1}^{s}$, we assume that the mixed first derivatives of the integrand satisfy the following family of upper bounds, for each $\mathfrak{u} \subseteq \{1 : s\}$:*

$$\int_{[0,1]^{|\mathfrak{u}|}} \left( \int_{[0,1]^{s-|\mathfrak{u}|}} \frac{\partial^{|\mathfrak{u}|}}{\partial \boldsymbol{x}_{\mathfrak{u}}} f(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}_{-\mathfrak{u}} \right)^2 \mathrm{d}\boldsymbol{x}_{\mathfrak{u}} \leq B_{|\mathfrak{u}|} \prod_{j \in \mathfrak{u}} b_j^2 \,. \tag{2}$$

Here $\{1 : s\}$ is shorthand for $\{1, 2, \ldots, s\}$, $\boldsymbol{x}_{\mathfrak{u}} = \{x_j\}_{j \in \mathfrak{u}}$ are the active variables, $\boldsymbol{x}_{-\mathfrak{u}} = \{x_j\}_{j \in \{1:s\} \setminus \mathfrak{u}}$ are the inactive variables, and $\partial^{|\mathfrak{u}|}/\partial \boldsymbol{x}_{\mathfrak{u}}$ is the first-order, mixed partial derivative with respect to $\boldsymbol{x}_{\mathfrak{u}}$.

Bounds of the form given in (2), together with explicit values of $B_\ell$ and $b_i$, have been found for a particular PDE problem in several recent papers, including [7].

The CBC algorithm, first invented by Korobov [4], and rediscovered in [13, 12], is an efficient method of constructing "good" QMC point sets such as lattice rules [8, 11]. The idea behind the construction is to work through each dimension $i = 1, 2, \ldots, s$ sequentially, choosing the $i$th component of the rule by minimising the

worst-case error in that dimension while all previous components remain fixed. Because the worst-case error depends explicitly on the weights, the CBC algorithm requires the weights as inputs. Thus, the weights are not only useful for theory, but also as a practical necessity. The relevant aspects of the CBC algorithm and general QMC theory will be presented in Section 2, however, for a more complete introduction to the concepts above see [2].

The CBC construction has the virtue that, as was shown in [1, 5], the worst-case error of the resulting QMC approximation converges to zero at a rate that, depending on the weights, can be arbitrarily close to $n^{-1}$, with a constant that can be independent of $s$. From [1, 5], the root-mean-square error of a CBC generated "randomly shifted lattice rule" approximation of $I_s f$ (in the special case of prime $n$ and product weights) is bounded above by

$$\left( \frac{1}{n-1} \prod_{i=1}^{s} \left( 1 + \gamma_i^\lambda \frac{2\zeta(2\lambda)}{(2\pi^2)^\lambda} \right) \right)^{\frac{1}{2\lambda}} \|f\|_{s,\gamma} \quad \text{for all } \lambda \in \left( \tfrac{1}{2}, 1 \right] , \tag{3}$$

where $\zeta(x) = \sum_{k=1}^{\infty} k^{-x}$ is the Riemann zeta function.

Until recently the choice of weights was generally *ad hoc*, but in the paper [7] a new principle was used to determine weights for a particular problem: having estimated an upper bound on the norm of the integrand (which like the worst-case error depends on the weights), those authors chose weights that minimise an upper bound on the error (3). (Note that [7] dealt with a specific problem of randomly shifted lattice rules applied to PDEs with random coefficients, however the strategy of that paper can easily be applied to other problems.) The result is a family of weight sequences indexed by the parameter $\lambda$, where $\lambda$ affects the theoretical rate of convergence. The fact that $\lambda$ must be chosen by the user is a major drawback of the strategy in [7]. One option would be to take $\lambda$ as close to $\frac{1}{2}$ as possible to ensure a good convergence rate, however, because of the occurrence of the zeta function $\zeta(2\lambda)$, the constant goes to $\infty$ as $\lambda \to \frac{1}{2}$. A good rate of convergence does not help for a fixed value of $n$ if the constant becomes too large. To obtain the best bound a delicate balance between the two factors in (3) is needed.

Another drawback of the method used in [7] is that the bound (3) is often a crude overestimate. The first algorithm we introduce in this paper, the **double CBC (DCBC) algorithm**, counters this while at the same time removing the need to choose $\lambda$ by dealing with the exact "shift-averaged" worst-case error (see Section 2), rather than the upper bound given by the first factor in (3). For the case of product weights, at step $i$ of the DCBC algorithm, after fixing the component of the lattice rule, the weight $\gamma_i$ is chosen so as to minimise a bound on the error in the current dimension. An advantage of this method is that the choice of weight in each dimension adds virtually no extra computational cost to the algorithm.

The second algorithm we propose begins with the upper bound (3), and hence the family of weights indexed by $\lambda$ obtained following the strategy in [7]. To choose the "best" $\lambda$, and in turn the "best" weights, an iteration of the CBC algorithm with respect to $\lambda$ is employed to minimise heuristically a bound on the approximation

error. Because of this iteration process it is called the **iterated CBC (ICBC) algorithm**.

The philosophy of both algorithms is to concentrate on reducing the guaranteed error bounds. This paradigm represents a shift away from the usual focus on the best rate of convergence. It is particularly useful when dealing with problems where function evaluations are highly expensive, such as those where QMC methods have been applied to PDEs with random coefficients [7], for which there is often a practical limit on the number of quadrature points $n$.

So far we have discussed only product weights, however, both algorithms can be extended to cover a more general form of weights called "POD weights" (see Section 2).

The structure of the paper is as follows. Section 2 provides a brief review of the relevant aspects of QMC theory. Details on our two new algorithms are given in Section 3. In Section 4 we give tables for the guaranteed error bounds resulting from the two algorithms, under various assumptions on the parameters $B_\ell$ and $b_i$ in (2). The examples show that there are different situations where each of the algorithms outperforms the other, thus it is not possible to say that one algorithm always outperforms the other.

## 2    Relevant quasi-Monte Carlo theory

Here we fix notation and briefly outline the relevant aspects of QMC theory, including weighted function spaces for error analysis, randomly shifted lattice rules and the CBC construction. For a more comprehensive overview the reader is referred to the review papers [2, 6] or the book [3].

### 2.1    Weighted Sobolev spaces and randomisation

Given a collection of positive real numbers, $\boldsymbol{\gamma} = \{\gamma_{\mathfrak{u}}\}$ where $\mathfrak{u}$ denotes a finite subset of $\mathbb{N} = \{1, 2, \ldots, \}$, let $W_{s,\boldsymbol{\gamma}}$ be the $s$-dimensional weighted Sobolev space with unanchored norm

$$\|f\|_{s,\boldsymbol{\gamma}}^2 = \sum_{\mathfrak{u} \subseteq \{1:s\}} \frac{1}{\gamma_{\mathfrak{u}}} \int_{[0,1]^{|\mathfrak{u}|}} \left( \int_{[0,1]^{s-|\mathfrak{u}|}} \frac{\partial^{|\mathfrak{u}|}}{\partial \boldsymbol{x}_{\mathfrak{u}}} f(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}_{-\mathfrak{u}} \right)^2 \mathrm{d}\boldsymbol{x}_{\mathfrak{u}} . \qquad (4)$$

In practice it is difficult to work with general weights $\gamma_{\mathfrak{u}}$ and so often weights with some inherent structure are used. The three most common forms are *product weights* where $\gamma_{\mathfrak{u}} = \prod_{j \in \mathfrak{u}} \gamma_j$ for some sequence $1 \geq \gamma_1 \geq \gamma_2 \geq \cdots > 0$; *order dependent weights* where each weight depends only on the cardinality of the set, $\gamma_{\mathfrak{u}} = \Gamma_{|\mathfrak{u}|}$, for a sequence of positive real numbers $\Gamma_0 := 1, \Gamma_1, \Gamma_2, \ldots$; and *product and order dependent (POD) weights* which are a hybrid of the previous two, with $\gamma_{\mathfrak{u}} = \Gamma_{|\mathfrak{u}|} \prod_{j \in \mathfrak{u}} \gamma_j$.

Given a random shift $\boldsymbol{\Delta}$ uniformly distributed on $[0,1]^s$, and point set $P_n = \{\boldsymbol{t}_k\}_{k=0}^{n-1}$, the randomly shifted point set $(P_n; \boldsymbol{\Delta}) = \{\tilde{\boldsymbol{t}}_k\}_{k=0}^{n-1}$ is obtained by taking

$\tilde{\boldsymbol{t}}_k = \{\boldsymbol{t}_k + \boldsymbol{\Delta}\}$, where the braces indicate that we take the fractional part of each component in the vector to ensure it still belongs to $[0,1]^s$. We will write the shifted QMC approximation as $Q_{n,s}^{\mathrm{sh}}(P_n; \boldsymbol{\Delta})$.

In this setting, the *shift-averaged worst-case error* is used as a measure of the quality of a point set. It is simply the worst-case error of the shifted point set, averaged in the root-mean-square sense over all possible shifts, that is, $e_{n,s,\boldsymbol{\gamma}}^{\mathrm{sh}}(P_n) :=$ $(\int_{[0,1]^s} e_{n,s,\boldsymbol{\gamma}}^2(P_n; \boldsymbol{\Delta}) \, \mathrm{d}\boldsymbol{\Delta})^{1/2}$. It then follows from the error bound (1) that the root-mean-square error of a shifted QMC approximation (where the expected value is taken with respect to the shift $\boldsymbol{\Delta}$) satisfies

$$\sqrt{\mathbb{E}\left(\left|I_s f - Q_{n,s}^{\mathrm{sh}}(P_n; \cdot) f\right|^2\right)} \leq e_{n,s,\boldsymbol{\gamma}}^{\mathrm{sh}}(P_n) \, \|f\|_{s,\boldsymbol{\gamma}} \, . \tag{5}$$

Again, the usefulness of this error bound lies in the fact that the right hand side splits into two factors. Note that both factors depend on the weights.

In practice we use a small number of independent and identically distributed random shifts to estimate the error of approximation, see e.g., [2].

## 2.2 Randomly shifted lattice rules

A *rank-1 lattice rule* is a QMC rule for which the quadrature points are generated by a single integer vector $\boldsymbol{z}$ called the *generating vector*. Each component $z_i$ belongs to $\mathbb{U}_n := \{z \in \mathbb{N} : z < n, \gcd(z,n) = 1\}$, the multiplicative group of integers modulo $n$, and $\boldsymbol{z} \in \mathbb{U}_n^s = \mathbb{U}_n \times \cdots \times \mathbb{U}_n$. The number of positive integers less than and co-prime to $n$ is given by the Euler totient function $\varphi(n) = |\mathbb{U}_n|$. So, for an $n$-point lattice rule in $s$ dimensions there are $(\varphi(n))^s$ possible generating vectors.

As mentioned in the previous section, incorporating randomness into the QMC rule is practically beneficial, and for lattice rules this is best done by randomly shifting the points. Given some random shift $\boldsymbol{\Delta} \sim U([0,1]^s)$, and generating vector $\boldsymbol{z}$, the $\boldsymbol{\Delta}$-shifted rank-1 lattice rule has points

$$\tilde{\boldsymbol{t}}_k = \left\{ \frac{k\boldsymbol{z}}{n} + \boldsymbol{\Delta} \right\}$$

for $k = 0, 1, \ldots, n-1$. The shift-averaged worst-case error of a randomly shifted lattice rule in the space $\mathcal{W}_{s,\boldsymbol{\gamma}}$ with general weights, is given explicitly by (see [2, Eq. (5.12)])

$$e_{n,s,\boldsymbol{\gamma}}^{\mathrm{sh}}(\boldsymbol{z}) = \sqrt{\sum_{\emptyset \neq \mathfrak{u} \subseteq \{1:s\}} \gamma_{\mathfrak{u}} \left( \frac{1}{n} \sum_{k=0}^{n-1} \prod_{j \in \mathfrak{u}} \mathscr{B}_2\left( \left\{ \frac{kz_j}{n} \right\} \right) \right)}, \tag{6}$$

where $\mathscr{B}_2(x) = x^2 - x + \frac{1}{6}$ is the Bernoulli polynomial of degree 2.

## 2.3 The component-by-component construction

The CBC algorithm is a method of constructing generating vectors that results in "good" lattice rules in the context of minimising the shift-averaged worst-case error (6). The CBC construction is a greedy algorithm that works through each component of the generating vector sequentially, choosing $z_i$ to *minimise the shift-averaged worst-case error in that dimension while all previous components remain fixed.*

**Algorithm 1 (The CBC algorithm)**
Given $n$ and $s$ and a sequence of weights $\boldsymbol{\gamma} = \{\gamma_{\mathfrak{u}}\}_{\mathfrak{u} \subseteq \{1:s\}}$.

1. Set $z_1$ to 1.

2. For $i = 2, \ldots, s$ choose $z_i \in \mathbb{U}_n$ so as to minimise $e_{n,i,\boldsymbol{\gamma}}^{\mathrm{sh}}(z_1, \ldots, z_{i-1}, z_i)$ given that all of the previous components $z_1, \ldots, z_{i-1}$ remain fixed.

Setting $z_1$ to be 1 is done by convention, since in the first dimension every choice results in an equivalent quadrature rule. Using structured weights (product, order dependent or POD form) simplifies the formula for the shift-averaged worst-case error (6) even further, allowing the calculation of $e_{n,i,\boldsymbol{\gamma}}^{\mathrm{sh}}(z_1, \ldots, z_{i-1}, z_i)$ for all $z_i \in \mathbb{U}_n$ together to be performed as one matrix-vector product. In general a naive implementation of this algorithm costs $\mathcal{O}(s\,n^2)$ operations, however a fast construction performs the matrix-vector product using a fast Fourier transform (FFT) reduces this to $\mathcal{O}(s\,n\log n)$ in the case of product weights and $\mathcal{O}(s\,n\log n + s^2 n)$ for order dependent or POD weights. For full details on the *Fast CBC construction* see [2, 9, 10].

The shift-averaged worst-case error of a CBC generated lattice rule satisfies the following upper bound:

$$e_{n,s,\boldsymbol{\gamma}}^{\mathrm{sh}}(\boldsymbol{z}) \leq \left( \frac{1}{\varphi(n)} \sum_{\emptyset \neq \mathfrak{u} \subseteq \{1:s\}} \gamma_{\mathfrak{u}}^{\lambda} \left( \frac{2\zeta(2\lambda)}{(2\pi^2)^{\lambda}} \right)^{|\mathfrak{u}|} \right)^{\frac{1}{2\lambda}} \quad \text{for all } \lambda \in \left( \tfrac{1}{2}, 1 \right]. \quad (7)$$

This result was proved in [2] for general weights and in [1, 5] for product weights.

## 3 The CBC black box algorithms

The two new algorithms introduced in this section aim to choose weights so as to make the bound (5) on the root-mean-square error of the QMC approximation as small as possible. Hence we will require a bound on the norm of the integrand $f \in \mathcal{W}_{s,\boldsymbol{\gamma}}$ to be known and of the specific form given in the Assumption (2) in the Introduction.

In this way the norm of $f$ in $W_{s,\boldsymbol{\gamma}}$, see (4), with some, as yet unspecified, weights $\boldsymbol{\gamma}$ will be bounded by

$$\|f\|_{s,\boldsymbol{\gamma}}^2 \leq \sum_{\mathfrak{u} \subseteq \{1:s\}} \frac{1}{\gamma_{\mathfrak{u}}} B_{|\mathfrak{u}|} \prod_{j \in \mathfrak{u}} b_j^2 =: M_{s,\boldsymbol{\gamma}}, \tag{8}$$

and in turn from (5) the mean-square error of a lattice rule approximation will be bounded by

$$\mathbb{E}\left( \left| I_s f - Q_{n,s}^{\mathrm{sh}}(\boldsymbol{z};\cdot)f \right|^2 \right) \leq \left( e_{n,s,\boldsymbol{\gamma}}^{\mathrm{sh}}(\boldsymbol{z}) \right)^2 M_{s,\boldsymbol{\gamma}}. \tag{9}$$

The first new algorithm is named the **double CBC (DCBC) algorithm** since at each step two parameters are chosen: the component of the generating vector and the weight. We assume the weights are of product or POD form. In the case of POD weights we assume that the order dependent weight factors $\{\Gamma_\ell\}_{\ell=0}^s$ are given. Starting with the error bound (9), in each dimension, with all previous parameters remaining fixed, we choose the component of $\boldsymbol{z}$ to minimise $e_{n,s,\boldsymbol{\gamma}}^{\mathrm{sh}}$ and then the product component of the weight to minimise the entire bound.

## 3.1   The double CBC algorithm for product weights

In the case of product weights, the squared shift-averaged worst-case error (see (6)) of a lattice rule with generating vector $\boldsymbol{z}$ is

$$\left( e_{n,s,\boldsymbol{\gamma}}^{\mathrm{sh}}(z_1, \ldots, z_s) \right)^2 = -1 + \frac{1}{n} \sum_{k=0}^{n-1} \prod_{j=1}^{s} \left( 1 + \gamma_j \mathscr{B}_2\left( \left\{ \frac{kz_j}{n} \right\} \right) \right)$$

$$= \left( e_{n,s-1,\boldsymbol{\gamma}}^{\mathrm{sh}}(z_1, \ldots, z_{s-1}) \right)^2 + \gamma_s G_s(z_1, \ldots, z_s),$$

in which the first term is independent of $\gamma_s$, and

$$G_s(z_1, \ldots, z_s) := \frac{1}{n} \sum_{k=0}^{n-1} \mathscr{B}_2\left( \left\{ \frac{kz_s}{n} \right\} \right) \prod_{j=1}^{s-1} \left( 1 + \gamma_j \mathscr{B}_2\left( \left\{ \frac{kz_j}{n} \right\} \right) \right).$$

For product weights it is natural to assume that the bound on the norm is also of product form, that is, $B_\ell = 1$ for all $\ell = 1, 2, \ldots, s$. It follows that this bound (8) can also be written recursively as

$$M_{s,\boldsymbol{\gamma}} = \sum_{\mathfrak{u} \subseteq \{1:s\}} \prod_{j \in \mathfrak{u}} \frac{b_j^2}{\gamma_j} = \prod_{j=1}^{s} \left( 1 + \frac{b_j^2}{\gamma_j} \right) = \left( 1 + \frac{b_s^2}{\gamma_s} \right) M_{s-1,\boldsymbol{\gamma}}. \tag{10}$$

In this situation, the bound (9) on the mean-square error can be written as

$$\left( \left( e_{n,s-1,\boldsymbol{\gamma}}^{\mathrm{sh}}(z_1, \ldots, z_{s-1}) \right)^2 + \gamma_s G_s(z_1, \ldots, z_s) \right) \left( 1 + \frac{b_s^2}{\gamma_s} \right) M_{s-1,\boldsymbol{\gamma}}. \tag{11}$$

Treating (11) as a function of $\gamma_s$ and $z_s$, and noting that $z_s$ is only present in $G_s$, at each step of the algorithm we can first choose $z_s$ to minimise $G_s$ and then choose $\gamma_s$ to minimise the entire error bound. For future reference, the minimiser of expressions of this form is given by the following Lemma.

**Lemma 1** *Suppose that $a, b, c, d$ are positive real numbers. Then the function $h :$ $(0, \infty) \to (0, \infty)$ given by $h(x) = (a + bx)(c + \frac{d}{x})$ is minimised by $x^* = \sqrt{\frac{ad}{bc}}$.*

**Proof.** The first two derivatives of $h$ with respect to $x$ are $h'(x) = bc - ad/x^2$ and $h''(x) = 2ad/x^3 > 0$ for $x > 0$, so $h$ is convex. Solving $h'(x) = 0$ yields the formula for the stationary point $x^*$, which is the global minimum. $\qquad\square$

Consequently, the choice of weight that minimises the bound on the mean-square error (11) is given by, with $s$ replaced by $i$,

$$\gamma_i = \sqrt{\frac{\left(e^{\text{sh}}_{n,i-1,\boldsymbol{\gamma}}(z_1, \ldots, z_{i-1})\right)^2 b_i^2}{G_i(z_1, \ldots, z_i)}}. \tag{12}$$

Note that in the first dimension the upper bound on the mean-square error (11) becomes $G_1\left(\gamma_1 + b_1^2\right)$, which attains its minimum when $\gamma_1 = 0$. Since 0 is not a sensible choice of weight our algorithm requires that $\gamma_1$ be given.

**Algorithm 2 (The double CBC algorithm for product weights)**
Given $n$ and $s$, and bounds of the form (2) with $B_\ell = 1$ for all $\ell$, and the weight in the first dimension $\gamma_1$, set $z_1$ to 1. Then for each $i = 2, \ldots, s$,

1. Choose $z_i \in \mathbb{U}_n$ to minimise $G_i(z_1, \ldots, z_{i-1}, z_i)$.

2. Set $\gamma_i$ as in (12) and update the mean-square error bound (11).

At each step of the algorithm, the process of choosing $z_i$ to minimise $G_i$ is the same as in the original CBC algorithm. Thus, the methods used in the fast CBC construction can also be applied in this algorithm.

## 3.2 The double CBC algorithm for POD weights

For weights of POD form, given a sequence of order dependent weight factors $\{\Gamma_\ell\}$ and a bound on the norm $M_{s,\boldsymbol{\gamma}}$ the algorithm chooses the product component of the weights $\gamma_i$ in each dimension. Note that, for this case we no longer assume all $B_\ell = 1$. As before, the first step is to obtain a recursive formula for the bound on the norm of the integrand in each dimension. By splitting the sum in (8) according

to whether or not $s$ belongs to the set $\mathfrak{u}$, we have

$$M_{s,\gamma} = \sum_{\ell=0}^{s} \frac{B_\ell}{\Gamma_\ell} \sum_{\substack{\mathfrak{u}\subseteq\{1:s\}\\|\mathfrak{u}|=\ell}} \prod_{j\in\mathfrak{u}} \frac{b_j^2}{\gamma_j} = \sum_{\ell=0}^{s} \frac{B_\ell}{\Gamma_\ell} \left( \sum_{\substack{\mathfrak{u}\subseteq\{1:s-1\}\\|\mathfrak{u}|=\ell}} \prod_{j\in\mathfrak{u}} \frac{b_j^2}{\gamma_j} + \sum_{\substack{s\in\mathfrak{u}\subseteq\{1:s\}\\|\mathfrak{u}|=\ell}} \prod_{j\in\mathfrak{u}} \frac{b_j^2}{\gamma_j} \right)$$

$$= M_{s-1,\gamma} + \frac{b_s^2}{\gamma_s} \sum_{\ell=1}^{s} \frac{B_\ell}{\Gamma_\ell} \underbrace{\sum_{\substack{\mathfrak{u}\subseteq\{1:s-1\}\\|\mathfrak{u}|=\ell-1}} \prod_{j\in\mathfrak{u}} \frac{b_j^2}{\gamma_j}}_{H_{s-1,\ell-1}} = M_{s-1,\gamma} + \frac{b_s^2}{\gamma_s} \sum_{\ell=0}^{s-1} H_{s-1,\ell}\,, \qquad (13)$$

where we have introduced the terms $H_{i,\ell}$ to simplify the notation. Applying a similar method of splitting the sum, a recursive formula for $H_{s,\ell}$ is obtained

$$H_{s,\ell} := \frac{B_{\ell+1}}{\Gamma_{\ell+1}} \sum_{\substack{\mathfrak{u}\subseteq\{1:s\}\\|\mathfrak{u}|=\ell}} \prod_{j\in\mathfrak{u}} \frac{b_j^2}{\gamma_j} = \frac{B_{\ell+1}}{\Gamma_{\ell+1}} \sum_{\substack{\mathfrak{u}\subseteq\{1:s-1\}\\|\mathfrak{u}|=\ell}} \prod_{j\in\mathfrak{u}} \frac{b_j^2}{\gamma_j} + \frac{B_{\ell+1}}{\Gamma_{\ell+1}} \sum_{\substack{s\in\mathfrak{u}\subseteq\{1:s\}\\|\mathfrak{u}|=\ell}} \prod_{j\in\mathfrak{u}} \frac{b_j^2}{\gamma_j}$$

$$= H_{s-1,\ell} + \frac{b_s^2 B_{\ell+1}}{\gamma_s \Gamma_{\ell+1}} \sum_{\substack{\mathfrak{u}\subseteq\{1:s-1\}\\|\mathfrak{u}|=\ell-1}} \prod_{j\in\mathfrak{u}} \frac{b_j^2}{\gamma_j}$$

$$= H_{s-1,\ell} + \frac{b_s^2}{\gamma_s} \frac{B_{\ell+1}}{B_\ell} \frac{\Gamma_\ell}{\Gamma_{\ell+1}} H_{s-1,\ell-1}\,, \qquad (14)$$

with $H_{i,0} = \frac{B_1}{\Gamma_1}$ for all $i = 1, 2, \ldots, s$ and $H_{i,\ell} = 0$ for all $\ell > i$.

It follows that for POD weights the upper bound (9) on the mean-square error of the QMC approximation can be written recursively as

$$\mathbb{E}\left( \left| I_s f - Q_{n,s}^{\text{sh}}(z_1, \ldots, z_s; \cdot) f \right|^2 \right) \qquad (15)$$

$$\leq \left( \left( e_{n,s-1,\gamma}^{\text{sh}}(z_1, \ldots, z_s) \right)^2 + \gamma_s G_s(z_1, \ldots, z_s) \right) \left( M_{s-1,\gamma} + \frac{b_s^2}{\gamma_s} \sum_{\ell=0}^{s-1} H_{s-1,\ell} \right),$$

where $G_s(z_1, \ldots, z_s)$ is now given by

$$G_s(z_1, \ldots, z_s)$$

$$= \frac{1}{n} \sum_{k=0}^{n-1} \left( \underbrace{\mathscr{B}_2\left( \left\{ \frac{kz_s}{n} \right\} \right)}_{\Omega_n(z_s,k)} \sum_{\ell=1}^{s} \frac{\Gamma_\ell}{\Gamma_{\ell-1}} \underbrace{\sum_{\substack{\mathfrak{u}\subseteq\{1:s-1\}\\|\mathfrak{u}|=\ell-1}} \left( \prod_{i=1}^{\ell-1} \frac{\Gamma_i}{\Gamma_{i-1}} \right) \prod_{j\in\mathfrak{u}} \left( \gamma_j \mathscr{B}_2\left( \left\{ \frac{kz_j}{n} \right\} \right) \right)}_{p_{s-1,\ell-1}(k)} \right)$$

$$= \frac{1}{n} \sum_{k=0}^{n-1} \Omega_n(z_s, k) \sum_{\ell=1}^{s} \frac{\Gamma_\ell}{\Gamma_{\ell-1}} p_{s-1,\ell-1}(k)\,. \qquad (16)$$

We have introduced the terms $\Omega_n$, $p_{s-1,\ell-1}$ to simplify notation, and we have arranged to deal only with the ratios $\Gamma_\ell/\Gamma_{\ell-1}$ to improve numerical stability. Again by Lemma 1, the product component of the weight that minimizes the bound on (15) is given by, with $s$ replaced by $i$,

$$\gamma_i = \sqrt{\frac{\left(e_{n,i,\boldsymbol{\gamma}}^{\mathrm{sh}}(z_1,\ldots,z_{i-1})\right)^2 b_i^2 \sum_{\ell=0}^{i-1} H_{i-1,\ell}}{M_{i-1,\boldsymbol{\gamma}}\, G_i(z_1,\ldots,z_i)}}. \tag{17}$$

Calculating $G_s$ for all $z_s \in \mathbb{U}_n$ by summing over all $\mathfrak{u} \subseteq \{1 : s-1\}$ as in (16) would cost $\mathcal{O}(n^2 2^{s-1})$ operations and is infeasible for even moderate $s$. The cost can be reduced by storing $\Omega_n$ and constructing $p_{s-1,\ell-1}$ recursively. Letting $\boldsymbol{G}_s = [G_s(z_1,\ldots,z_{s-1},z_s)]_{z_s \in \mathbb{U}_n}$, the calculation of $G_s$ for all $z_s \in \mathbb{U}_n$ can be performed by the matrix-vector product

$$\boldsymbol{G}_s = \frac{1}{n}\boldsymbol{\Omega}_n \sum_{\ell=1}^{s} \frac{\Gamma_\ell}{\Gamma_{\ell-1}} \boldsymbol{p}_{s-1,\ell-1}, \tag{18}$$

where

$$\boldsymbol{\Omega}_n := \left[\mathscr{B}_2\left(\left\{\frac{kz}{n}\right\}\right)\right]_{z\in\mathbb{U}_n,\,k=0,1,\ldots,n-1}.$$

At each step the vectors $p_{s-1,\ell-1}$ can be constructed recursively as follows

$$\boldsymbol{p}_{s,\ell} = \boldsymbol{p}_{s-1,\ell} + \frac{\Gamma_\ell}{\Gamma_{\ell-1}} \gamma_s \boldsymbol{\Omega}_n(z_s,:) .* \boldsymbol{p}_{s-1,\ell-1}, \tag{19}$$

where $\boldsymbol{\Omega}_n(z_s,:)$ is the row corresponding the new component of the generating vector $z_s$, and $.*$ denotes component-wise multiplication, and with $\boldsymbol{p}_{s,0} = \mathbf{1}$, $\boldsymbol{p}_{s,\ell} = \mathbf{0}$ for all $\ell > s$. Note that (19) is obtained by splitting the sum according to whether or not $s \in \mathfrak{u}$, as in (13) and (14). Since the cost of updating $\boldsymbol{p}_{s,\ell}$ is $\mathcal{O}(s\,n)$ the total cost of calculating $G_s$ in each dimension has been reduced to $\mathcal{O}(n^2 + sn)$ operations. Additionally, using the concepts from the Fast CBC algorithm [9, 10] this product can be performed more efficiently using the FFT which would further reduce the cost to $\mathcal{O}(n \log n + sn)$. The total cost of the algorithm is $\mathcal{O}(s\,n \log n + s^2 n)$ operations.

**Algorithm 3 (The double CBC algorithm for POD weights)**
Given $n$ and $s$, bounds of the form (2), order dependent weight factors $\{\Gamma_\ell\}_{\ell=0}^{s}$, and the weight in the first dimension $\gamma_1$, set $z_1 = 1$, $H_{0,0} = B_1/\Gamma_1$, $\boldsymbol{p}_{0,0} = \mathbf{1}$. Then for each $i = 2,\ldots,s$,

1. For $\ell = 0,\ldots,i-1$, update $H_{i-1,\ell}$ using (14) and $\boldsymbol{p}_{i-1,\ell}$ using (19).

2. Calculate $\boldsymbol{G}_i$ using (18) and FFT.

3. Choose $z_i \in \mathbb{U}_n$ to minimise $G_i(z_1,\ldots,z_{i-1},z_i)$.

4. Set $\gamma_i$ as in (17) and update the mean-square error bound (15).

We have so far neglected the question of how to choose the order dependent weight factors $\Gamma_\ell$. Three possible choices are:

- $\Gamma_\ell$ given *a priori*, such as by the common choice $\Gamma_\ell = \ell!$.

- $\Gamma_\ell = \Gamma_\ell(\lambda)$, that is, the order dependent weight factors of the weights $\gamma_{\mathfrak{u}}(\lambda)$ from the formula (20) below. In this case we are still left with the predicament of how to choose $\lambda$, a choice which this algorithm aimed to circumvent.

- $\Gamma_\ell = B_\ell$. Here the recursion for the bound on the norm (13) is the same as the product weight case (10), that is, the terms $H_{i,\ell}$ are no longer required. Further, since there is some inherent connection between the form of the bound on the norm and the weights this choice seems more natural than the other two.

## 3.3 The iterated CBC algorithm

Combining (9) with the upper bound on shift-averaged worst-case error (7) we have that the mean-square error of a CBC constructed lattice rule approximation is bounded above by

$$\mathbb{E}\left(\left|I_s f - Q_{n,s}^{\mathrm{sh}}(\boldsymbol{z};\cdot)f\right|^2\right) \leq \left(\frac{1}{\varphi(n)} \sum_{\emptyset \neq \mathfrak{u} \subseteq \{1:s\}} \gamma_{\mathfrak{u}}^\lambda \left(\frac{2\zeta(2\lambda)}{(2\pi^2)^\lambda}\right)^{|\mathfrak{u}|}\right)^{\frac{1}{\lambda}}$$
$$\times \left(\sum_{\mathfrak{u} \subseteq \{1:s\}} \frac{1}{\gamma_{\mathfrak{u}}} B_{|\mathfrak{u}|} \prod_{j \in \mathfrak{u}} b_j^2\right) \quad \text{for all } \lambda \in \left(\frac{1}{2},1\right].$$

From [7, Lemma 6.2], the weights that minimise this error bound for each $\lambda \in \left(\frac{1}{2},1\right]$ are of POD form

$$\gamma_{\mathfrak{u}}(\lambda) = \Gamma_{|\mathfrak{u}|}(\lambda) \prod_{j \in \mathfrak{u}} \gamma_j(\lambda) = \left(B_{|\mathfrak{u}|} \prod_{j \in \mathfrak{u}} \frac{(2\pi^2)^\lambda b_j^2}{2\zeta(2\lambda)}\right)^{\frac{1}{1+\lambda}}. \tag{20}$$

For each $\lambda \in \left(\frac{1}{2},1\right]$ the corresponding weights $\boldsymbol{\gamma}(\lambda) = \{\gamma_{\mathfrak{u}}(\lambda)\}_{\mathfrak{u} \subseteq \{1:s\}}$ can be taken as input into the CBC algorithm to construct a lattice rule generating vector which, through the weights, depends on $\lambda$: $\boldsymbol{z}(\lambda)$. Now that we know the weights $\boldsymbol{\gamma}(\lambda)$ and generating vector $\boldsymbol{z}(\lambda)$ explicitly, from (9), the mean-square error of the resulting QMC approximation is bounded by

$$\mathbb{E}\left(\left|I_s f - Q_{s,n}^{\mathrm{sh}}(\boldsymbol{z};\cdot)f\right|^2\right) \leq \left(e_{n,s,\boldsymbol{\gamma}(\lambda)}^{\mathrm{sh}}(\boldsymbol{z}(\lambda))\right)^2 M_{s,\boldsymbol{\gamma}(\lambda)} =: E_{n,s,\boldsymbol{z}(\lambda)}(\lambda). \tag{21}$$

11

The goal of the **iterated CBC (ICBC) algorithm** is to carry out iterations of the original CBC algorithm to choose a $\lambda$ that minimises the right hand side of (21). However, since each component $z_j$ is obtained by a minimisation over a set of integers and because this minimisation depends on the weights (and hence $\lambda$), when treated as a function of $\lambda$ the shift-averaged worst-case error is discontinuous. Hence, we cannot guarantee that a minimum exists and as such our algorithm heuristically searches for a "good" value of $\lambda$. As mentioned in the introduction, the choice of $\lambda$ is non-trivial since one needs to balance the size of the constant and the theoretical convergence rate.

Suppose that in the upper bound (21) the generating vector $\boldsymbol{z}$ remains fixed, then the upper bound, $E_{n,s,\boldsymbol{z}}(\lambda)$, is a continuous function of the single variable $\lambda$ and can be minimised numerically.

The idea behind this algorithm is at each step of the iteration to use $E_{n,s,\boldsymbol{z}^{(k)}}(\lambda)$ as an approximation to the right hand side of the upper bound (21). In this way the next iterate $\lambda_{k+1}$ is taken to be the minimiser of $E_{n,s,\boldsymbol{z}^{(k)}}(\lambda)$, which can be found numerically using a quasi-Newton method.

**Algorithm 4 (The iterated CBC algorithm)**
Given $n$, $s$, bounds of the form (2), an initial $\lambda_0 \in \left(\frac{1}{2}, 1\right]$, a tolerance $\tau$ and a maximum number of iterations $k_{\max}$. For $k = 0, 1, 2, \ldots, k_{\max}$:

1. Generate the weights $\gamma_{\mathfrak{u}}(\lambda_k)$ using (20).

2. Construct the generating vector $\boldsymbol{z}^{(k)}$ from the original CBC algorithm with weights $\gamma_{\mathfrak{u}}(\lambda_k)$.

3. If $\left| \dfrac{\mathrm{d}}{\mathrm{d}\lambda} E_{n,s,\boldsymbol{z}^{(k)}}(\lambda_k) \right| < \tau$ then end the algorithm.

4. Otherwise, choose $\lambda_{k+1}$ to be the minimiser of $E_{n,s,\boldsymbol{z}^{(k)}}(\lambda)$, found numerically using a quasi-Newton algorithm.

**Remark 1** For the quasi-Newton algorithm in Step 4 we require the derivative of $E_{n,s,\boldsymbol{z}}$, for fixed $\boldsymbol{z}$, with respect to $\lambda$

$$
\frac{\mathrm{d}E_{n,s,\boldsymbol{z}}}{\mathrm{d}\lambda} = \left( \sum_{\emptyset \neq \mathfrak{u} \subseteq \{1:s\}} \gamma'_{\mathfrak{u}}(\lambda) \left( \frac{1}{n} \sum_{k=0}^{n-1} \prod_{j \in \mathfrak{u}} \mathscr{B}_2 \left( \left\{ \frac{kz_j}{n} \right\} \right) \right) \right) \left( \sum_{\mathfrak{u} \subseteq \{1:s\}} \frac{B_{|\mathfrak{u}|} \prod_{j \in \mathfrak{u}} b_j^2}{\gamma_{\mathfrak{u}}(\lambda)} \right)
$$
$$
- \left( \sum_{\emptyset \neq \mathfrak{u} \subseteq \{1:s\}} \gamma_{\mathfrak{u}}(\lambda) \left( \frac{1}{n} \sum_{k=0}^{n-1} \prod_{j \in \mathfrak{u}} \mathscr{B}_2 \left( \left\{ \frac{kz_j}{n} \right\} \right) \right) \right) \left( \sum_{\mathfrak{u} \subseteq \{1:s\}} \frac{\gamma'_{\mathfrak{u}}(\lambda)}{\gamma_{\mathfrak{u}}^2(\lambda)} B_{|\mathfrak{u}|} \prod_{j \in \mathfrak{u}} b_j^2 \right),
$$

where the derivative of each weight with respect $\lambda$ is

$$
\gamma'_{\mathfrak{u}}(\lambda) = \gamma_{\mathfrak{u}}(\lambda) \left( \frac{-\log \left( B_{|\mathfrak{u}|} \prod_{j \in \mathfrak{u}} \frac{(2\pi^2)^\lambda b_j^2}{2\zeta(2\lambda)} \right)}{(1+\lambda)^2} + \frac{|\mathfrak{u}| \left( \log(2\pi^2) - \frac{2\zeta'(2\lambda)}{\zeta(2\lambda)} \right)}{(1+\lambda)} \right).
$$

# 4 Numerical results

For the numerical results we look at how each new method performs for different types of bounds (2), that is, for different sequences $\boldsymbol{b} := (b_i)_{i=1}^s$ and $\boldsymbol{B} := (B_\ell)_{\ell=1}^s$. As a figure of merit we will use the upper bound on the root-mean-square (RMS) error (cf. (9)), which we denote by

$$E_{n,s,\boldsymbol{b},\boldsymbol{B}}(\boldsymbol{\gamma}, \boldsymbol{z}) := e_{n,s,\boldsymbol{\gamma}}^{\mathrm{sh}}(\boldsymbol{z}) \sqrt{M_{s,\boldsymbol{\gamma}}}.$$

Here we have specifically used this notation to indicate the dependence on the sequences $\boldsymbol{b}$, $\boldsymbol{B}$ but also to emphasise that this upper bound is primarily a function of $\boldsymbol{\gamma}$ and $\boldsymbol{z}$, the outputs of our algorithms. Note also that in Tables 1–8 the results given for our algorithms are *guaranteed* error bounds (in the RMS sense) for integrands which satisfy the appropriate bounds.

In the examples let the maximum dimension be $s = 100$ and the number of points $n$ be prime and ranging up to 32,003. Here we choose $n$ to be prime because it makes the "fast" aspects of the implementation simpler, but note that $n$ prime is not a requirement of either algorithm. Also, we use the notation "e" for the base-10 exponent.

## 4.1 The case $B_\ell = 1$

As a start, let $B_\ell = 1$ for all $\ell = 1, \ldots, s$, and consider the cases $b_i = i^{-2}$, $b_i = 0.5^i$ and $b_i = 0.8^i$. With $B_\ell = 1$ it is natural to restrict attention to product weights. The results for this case are given in Tables 1, 2, 3, respectively. These tables compare results for $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ from the DCBC, ICBC algorithms with the original CBC algorithm using common choices of product weights. The choices of common weights are $\gamma_i = i^{-1.1}$, $\gamma_i = i^{-2}$ and $\gamma_i(\lambda)$ as in (20) with $\lambda = 0.6, 1$. The row labelled "rate" gives the exponent ($x$) for a least-squares fit of the result $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ to a power law ($n^{-x}$).

Comparing results for the two new algorithms, note that for $b_i = i^{-2}$ (Table 1) the results of the DCBC algorithm are better, while for $b_i = 0.5^i$, $0.8^i$ (Tables 2, 3) the ICBC algorithm produces better bounds, and so it is not the case that one algorithm is always better than the other. However, in all cases the ICBC algorithm performs as well as or better than the original CBC algorithm with common choices of weights.

Table 4 gives the final value of $\lambda$, denoted $\lambda^*$, resulting from the ICBC algorithm for our three choices of $\boldsymbol{b}$, along with the resulting RMS error bound. Notice that, as expected the value of $\lambda^*$ found by the algorithm appears to approach 0.5 as $n$ increases, albeit very slowly.

Figures 1a and 1b compare the weight $\gamma_i$ in each dimension for the DCBC, ICBC algorithms, with $\gamma_i = i^{-2}$ as a reference, for $b_i = i^{-2}$ and $b_i = 0.5^i$, respectively. Here $n = 1999$.

Figure 2 compares the RMS error bound in each dimension for the DCBC and ICBC algorithms, for $b_i = 0.8^i$ with $n = 1999$. Notice that due to the greedy nature

|  | Variants | | CBC with common weights | | | |
|---|---|---|---|---|---|---|
| $n$ | **DCBC** | ICBC | $\gamma_i = i^{-1.1}$ | $\gamma_i = i^{-2}$ | $\gamma_i(\lambda = 0.6)$ | $\gamma_i(\lambda = 1)$ |
| 251 | **6.8e-3** | 7.0e-3 | 3.5e-2 | 7.5e-3 | 8.2e-3 | 1.3e-2 |
| 499 | **3.5e-3** | 3.6e-3 | 2.1e-2 | 4.0e-3 | 4.2e-3 | 7.6e-3 |
| 997 | **1.8e-3** | 1.9e-3 | 1.3e-2 | 2.2e-3 | 2.2e-3 | 4.3e-3 |
| 1999 | **9.7e-4** | 1.0e-3 | 7.8e-3 | 1.2e-3 | 1.1e-3 | 2.4e-3 |
| 4001 | **5.1e-4** | 5.2e-4 | 4.8e-3 | 6.3e-4 | 5.8e-4 | 1.4e-3 |
| 7993 | **2.7e-4** | 2.7e-4 | 2.9e-3 | 3.4e-4 | 2.9e-4 | 7.8e-4 |
| 16001 | **1.4e-4** | 1.4e-4 | 1.8e-3 | 1.9e-4 | 1.5e-4 | 4.4e-4 |
| 32003 | **7.4e-5** | 7.5e-5 | 1.1e-3 | 1.0e-4 | 7.9e-5 | 2.5e-4 |
| rate | **0.93** | 0.93 | 0.71 | 0.88 | 0.95 | 0.82 |

Table 1: Results for the root-mean-square error bound $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ for $b_i = i^{-2}$: DCBC, ICBC and CBC results for common choices of weights.

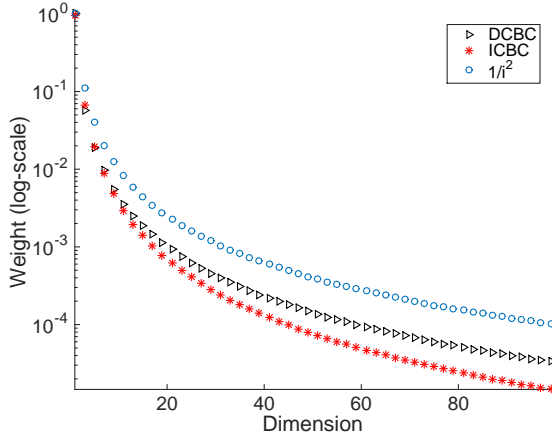|  | Variants | | CBC with common weights | | | |
|---|---|---|---|---|---|---|
| $n$ | DCBC | **ICBC** | $\gamma_i = i^{-1.1}$ | $\gamma_i = i^{-2}$ | $\gamma_i(\lambda = 0.6)$ | $\gamma_i(\lambda = 1)$ |
| 251 | 4.1e-3 | **3.3e-3** | 2.8e-2 | 5.5e-3 | 3.3e-3 | 6.7e-3 |
| 499 | 2.1e-3 | **1.7e-3** | 1.7e-2 | 2.9e-3 | 1.7e-3 | 3.6e-3 |
| 997 | 1.1e-3 | **8.6e-4** | 1.0e-2 | 1.6e-3 | 8.6e-4 | 2.0e-3 |
| 1999 | 5.6e-4 | **4.4e-4** | 6.2e-3 | 8.6e-4 | 4.4e-4 | 1.1e-3 |
| 4001 | 2.9e-4 | **2.2e-4** | 3.8e-3 | 4.6e-4 | 2.2e-4 | 5.8e-4 |
| 7993 | 1.5e-4 | **1.1e-4** | 2.3e-3 | 2.5e-4 | 1.1e-4 | 3.1e-4 |
| 16001 | 7.6e-5 | **5.9e-5** | 1.4e-3 | 1.4e-4 | 5.9e-5 | 1.7e-4 |
| 32003 | 3.9e-5 | **3.0e-5** | 8.7e-4 | 7.5e-5 | 3.0e-5 | 9.3e-5 |
| rate | 0.96 | **0.96** | 0.71 | 0.88 | 0.97 | 0.88 |

Table 2: Results for the root-mean-square error bound $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ for $b_i = 0.5^i$: DCBC, ICBC and CBC results for common choices of weights.

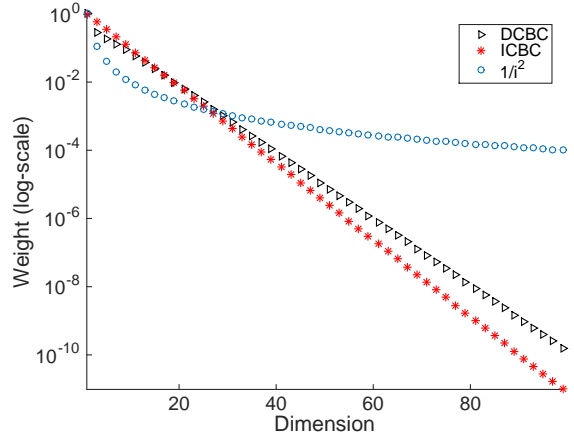|  | Variants | | CBC with common weights | | | |
|---|---|---|---|---|---|---|
| $n$ | DCBC | **ICBC** | $\gamma_i = i^{-1.1}$ | $\gamma_i = i^{-2}$ | $\gamma_i(\lambda = 0.6)$ | $\gamma_i(\lambda = 1)$ |
| 251 | 9.9e-2 | **8.3e-2** | 2.0e-1 | 2.8 | 1.6e-1 | 1.2e-1 |
| 499 | 5.7e-2 | **5.0e-2** | 1.2e-1 | 1.5 | 8.9e-2 | 7.2e-2 |
| 997 | 3.5e-2 | **2.9e-2** | 7.5e-2 | 8.2e-1 | 5.1e-2 | 4.5e-2 |
| 1999 | 2.1e-2 | **1.7e-2** | 4.6e-2 | 4.4e-1 | 2.8e-2 | 2.8e-2 |
| 4001 | 1.2e-2 | **1.0e-2** | 2.8e-2 | 2.4e-1 | 1.6e-2 | 1.8e-2 |
| 7993 | 7.3e-3 | **5.9e-3** | 1.7e-2 | 1.3e-1 | 9.1e-3 | 1.1e-2 |
| 16001 | 4.3e-3 | **3.5e-3** | 1.0e-2 | 7.1e-2 | 5.0e-3 | 6.7e-3 |
| 32003 | 2.5e-3 | **2.0e-3** | 6.4e-3 | 3.9e-2 | 2.9e-3 | 4.2e-3 |
| rate | 0.75 | **0.75** | 0.71 | 0.88 | 0.82 | 0.69 |

Table 3: Results for the root-mean-square error bound $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ for $b_i = 0.8^i$: DCBC, ICBC and CBC results for common choices of weights.

| $n$ | $b_i = i^{-2}$ | $b_i = 0.5^i$ | $b_i = 0.8^i$ |
|---|---|---|---|
| 251 | 0.672 | 0.616 | 0.756 |
| 499 | 0.668 | 0.615 | 0.744 |
| 997 | 0.661 | 0.610 | 0.735 |
| 1999 | 0.657 | 0.607 | 0.725 |
| 4001 | 0.652 | 0.604 | 0.715 |
| 7993 | 0.645 | 0.601 | 0.711 |
| 16001 | 0.642 | 0.597 | 0.700 |
| 32003 | 0.637 | 0.594 | 0.696 |

Table 4: Value of $\lambda^*$ from ICBC for product weights with $b_i = i^{-2}$, $0.5^i$ and $0.8^i$.



(a) $b_i = i^{-2}$.　　　　　　　　　　　　　(b) $b_i = 0.5^i$.

Figure 1: Weight in each dimension found from DCBC and ICBC, with $\gamma_i = i^{-2}$ for comparison ($n = 1999$).
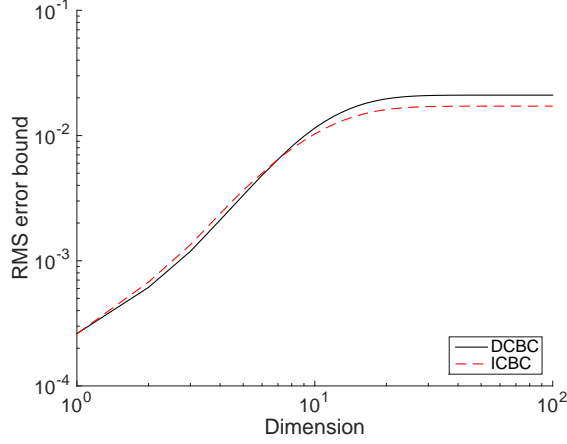
Figure 2: $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ in each dimension for DCBC and ICBC with $b_i = 0.8^i$ (loglog scale).

of the DCBC algorithm, it performs better in the earlier dimensions, however, the ICBC algorithm produces weights with a better bound overall.

## 4.2 POD weights

Now, we look at the performance of each algorithm for combinations of $B_\ell = \ell$ and $B_\ell = \ell!$ with $b_i = i^{-2}$ and $b_i = 0.5^i$, the results are given in Tables 5–8. Each table presents the RMS error bound $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ obtained from the DCBC algorithm for different choices of order dependent weights, and those obtained from the ICBC algorithm along with the output $\lambda^*$. With regards to the strategy for choosing $\Gamma_\ell$ in the DCBC algorithm for POD weights, in three out of the four cases the best choice was to let $\Gamma_\ell = B_\ell$ (see Tables 5–7). However, in all cases the results are similar and indicate that the choice of $\Gamma_\ell$ does not greatly effect the final bound $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$.

|  | DCBC | | ICBC | |
|---|---|---|---|---|
| $n$ | $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ $(\Gamma_\ell = B_\ell)$ | $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ $(\Gamma_\ell = \ell!)$ | $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ | $\lambda^*$ |
| 251 | 8.6e-3 | 8.5e-3 | **8.7e-3** | 0.680 |
| 499 | 4.6e-3 | 4.5e-3 | **4.6e-3** | 0.673 |
| 997 | 2.5e-3 | 2.5e-3 | **2.5e-3** | 0.666 |
| 1999 | 1.3e-3 | 1.3e-3 | **1.3e-3** | 0.659 |
| 4001 | 6.9e-4 | 7.0e-4 | **6.8e-4** | 0.655 |
| 7993 | 3.7e-4 | 3.7e-4 | **3.6e-4** | 0.650 |
| 16001 | 1.9e-4 | 2.0e-4 | **1.9e-4** | 0.645 |
| 32003 | 1.0e-4 | 1.1e-4 | **1.0e-4** | 0.640 |
| rate | 0.91 | 0.90 | **0.92** | |

Table 5: POD weight results with $b_i = i^{-2}$ and $B_\ell = \ell$: $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ from DCBC for different choices of $\Gamma_\ell$, $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ from ICBC and $\lambda^*$ from ICBC.

| | DCBC | | ICBC | |
|---|---|---|---|---|
| $n$ | $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ ($\Gamma_\ell = B_\ell$) | $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ ($\Gamma_\ell = \ell$) | $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ | $\lambda^*$ |
| 251 | **9.2e-3** | 1.1e-2 | 9.7e-3 | 0.692 |
| 499 | **5.0e-3** | 5.8e-3 | 5.1e-3 | 0.685 |
| 997 | **2.7e-3** | 3.2e-3 | 2.8e-3 | 0.679 |
| 1999 | **1.5e-3** | 1.7e-3 | 1.5e-3 | 0.673 |
| 4001 | **7.9e-4** | 9.6e-4 | 8.0e-4 | 0.667 |
| 7993 | **4.2e-4** | 5.2e-4 | 4.3e-4 | 0.661 |
| 16001 | **2.3e-4** | 2.8e-4 | 2.3e-4 | 0.656 |
| 32003 | **1.2e-4** | 1.6e-4 | 1.3e-4 | 0.651 |
| rate | **0.89** | 0.87 | 0.89 | |

Table 6: POD weight results with $b_i = i^{-2}$ and $B_\ell = \ell!$: $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ from DCBC for different choices of $\Gamma_\ell$, $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ from ICBC and $\lambda^*$ from ICBC.

| | DCBC | | ICBC | |
|---|---|---|---|---|
| $n$ | $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ ($\Gamma_\ell = B_\ell$) | $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ ($\Gamma_\ell = \ell!$) | $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ | $\lambda^*$ |
| 251 | 4.9e-3 | 5.0e-3 | **3.8e-3** | 0.619 |
| 499 | 2.5e-3 | 2.6e-3 | **2.0e-3** | 0.617 |
| 997 | 1.3e-3 | 1.4e-3 | **1.0e-3** | 0.612 |
| 1999 | 6.9e-4 | 7.2e-4 | **5.3e-4** | 0.608 |
| 4001 | 3.6e-4 | 3.8e-4 | **2.7e-4** | 0.605 |
| 7993 | 1.9e-4 | 2.0e-4 | **1.4e-4** | 0.602 |
| 16001 | 9.8e-5 | 1.0e-4 | **7.2e-5** | 0.597 |
| 32003 | 5.1e-5 | 5.3e-5 | **3.7e-5** | 0.595 |
| rate | 0.94 | 0.93 | **0.95** | |

Table 7: POD weight results with $b_i = 0.5^i$ and $B_\ell = \ell$: $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ from DCBC for different choices of $\Gamma_\ell$, $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ from ICBC and $\lambda^*$ from ICBC.

| | DCBC | | ICBC | |
|---|---|---|---|---|
| $n$ | $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ ($\Gamma_\ell = B_\ell$) | $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ ($\Gamma_\ell = \ell$) | $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ | $\lambda^*$ |
| 251 | 5.1e-3 | 5.1e-3 | **4.0e-3** | 0.625 |
| 499 | 2.6e-3 | 2.6e-3 | **2.1e-3** | 0.622 |
| 997 | 1.4e-3 | 1.4e-3 | **1.1e-3** | 0.618 |
| 1999 | 7.3e-4 | 7.3e-4 | **5.6e-4** | 0.614 |
| 4001 | 3.9e-4 | 3.8e-4 | **2.9e-4** | 0.608 |
| 7993 | 2.0e-4 | 2.0e-4 | **1.5e-4** | 0.604 |
| 16001 | 1.1e-4 | 1.0e-4 | **7.9e-5** | 0.602 |
| 32003 | 5.6e-5 | 5.5e-5 | **4.1e-5** | 0.599 |
| rate | 0.93 | 0.93 | **0.95** | |

Table 8: POD weight results with $b_i = 0.5^i$ and $B_\ell = \ell!$: $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ from DCBC for different choices of $\Gamma_\ell$, $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ from ICBC and $\lambda^*$ from ICBC.

# 5 Concluding remark

We introduced two new CBC algorithms, the double CBC (DCBC) algorithm and the iterated CBC (ICBC) algorithm, which only require parameters specified by the problem to determine the point set for QMC integration, and provide guaranteed error bounds by also choosing "good" weight parameters. The numerical results show different examples where each algorithm performs better than the other. Both algorithms generally outperform the original CBC algorithm with common choices of weights. In all cases the entries $E_{n,s,\boldsymbol{b},\boldsymbol{B}}$ provide guaranteed upper bounds on the root-mean-square error for the randomly shifted integration rules, under the indicated assumptions on the bound parameters $B_\ell$ and $b_i$.

# References

[1] J. Dick, On the convergence rate of the component-by-component construction of good lattice rules, J. Complexity 20 (2004) 493–522.

[2] J. Dick, F. Y. Kuo, I. H. Sloan, High-dimensional integration: The quasi-Monte Carlo way, Acta Numerica 22 (2013) 133–288.

[3] J. Dick, F. Pillichshammer, Digital Nets and Sequences: Discrepancy Theory and Quasi-Monte Carlo Integration, Cambridge University Press, New York, NY, USA, 2010.

[4] N. M. Korobov, Approximate evaluation of repeated integrals, Dokl. Akad. Nauk SSSR 124 (1959) 1207–1210.

[5] F. Y. Kuo, Component-by-component constructions achieve the optimal rate of convergence for multivariate integration in weighted Korobov and Sobolev spaces, J. Complexity 19 (2003) 301–320.

[6] F. Y. Kuo, C. Schwab, I. H. Sloan, Quasi-Monte Carlo methods for high-dimensional integration: the standard (weighted Hilbert space) setting and beyond, ANZIAM Journal 53 (2012a) 1–37.

[7] F. Y. Kuo, C. Schwab, I. H. Sloan, Quasi-Monte Carlo finite element methods for a class of elliptic partial differential equations with random coefficients, SIAM J. Numer. Anal. 50 (2012b) 3351–3374.

[8] H. Niederreiter, Random number generation and quasi-Monte Carlo methods, Regional Conference Series in Applied Mathematics, SIAM, 1992.

[9] D. Nuyens, R. Cools, Fast algorithms for component-by-component construction of rank-1 lattice rules in shift-invariant reproducing kernel Hilbert spaces, Math. Comp. 75 (2006a) 903–920.

[10] D. Nuyens, R. Cools, Fast component-by-component construction of rank-1 lattice rules with a non-prime number of points, J. Complexity 22 (2006b) 4–28.

[11] I. H. Sloan and S. Joe, Lattice Methods for Multiple Integration, Oxford University Press, Oxford, 1994.

[12] I. H. Sloan, F. Y. Kuo and S. Joe, Constructing randomly shifted lattice rules in weighted Sobolev spaces, SIAM J. Numer. Anal. 40 (2002) 1650–1665.

[13] I. H. Sloan and A. V. Reztsov, Component-by-component construction of good lattice rules, Math. Comp. 71 (2001) 263–273.

[14] I. H. Sloan and H. Woźniakowski, When are quasi-Monte Carlo algorithms efficient for high dimensional integrals?, J. Complexity 14 (1998) 1–33.

Alexander D. Gilbert
alexander.gilbert@unsw.edu.au
School of Mathematics and Statistics, University of New South Wales, Sydney NSW 2052, Australia

Frances Y. Kuo
f.kuo@unsw.edu.au
School of Mathematics and Statistics, University of New South Wales, Sydney NSW 2052, Australia

Ian H. Sloan
i.sloan@unsw.edu.au
School of Mathematics and Statistics, University of New South Wales, Sydney NSW 2052, Australia