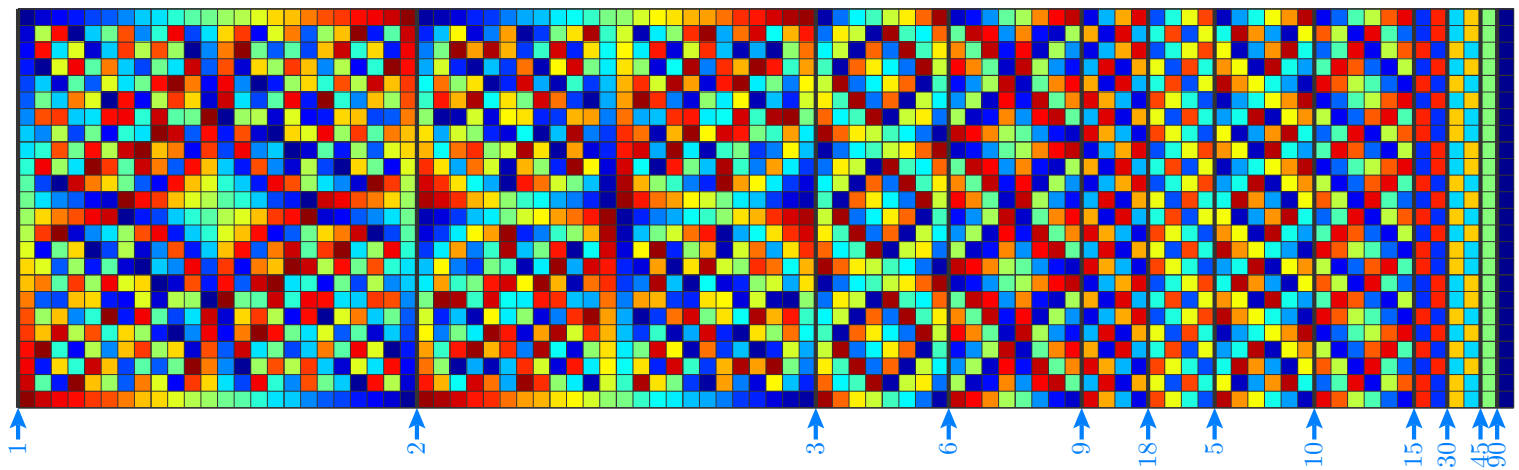
Lattice sequences: From fast construction to rapid application

Dirk Nuyens*, Ronald Cools
⟨(dirk.nuyens|ronald.cools)@cs.kuleuven.be⟩

Department of Computer Science, K.U.Leuven



Second workshop on high-dimensional approximation
Australian National University, Canberra, February 19–22, 2007.

Outline

1. High-dimensional integration
2. Good lattice rules
3. Fast constructions
4. Software issues
5. A periodizing application

Integration over $[0,1)^s$

Approximate the s-dimensional integral

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

by a simple *cubature rule*

$$Q(f; P_n) := \sum_{k=0}^{n-1} w_k\, f(\boldsymbol{x}_k)$$

using n points $\boldsymbol{x}_k \in P_n$ and associated weights w_k.

$\Rightarrow$ What rule to use? There are many choices...

$\Rightarrow$ It is probably important to have an *open* method that allows us to add more points if needed...

Classical product rules look like

$$\int_0^1 \cdots \int_0^1 f(\boldsymbol{x})\, \mathrm{d}(\boldsymbol{x}) \approx \sum_{k_1=0}^{m-1} w_{k_1}^{(1)} \cdots \sum_{k_s=0}^{m-1} w_{k_s}^{(s)} f(x_{k_1}^{(1)}, \ldots, x_{k_s}^{(s)}).$$

- By construction the total number of points is exponential in the number of dimensions, e.g., $n = m^s$.
- The weights slightly complicate matters.

A possible solution is *Monte Carlo type integration*:

$$\int_0^1 \cdots \int_0^1 f(\boldsymbol{x})\, \mathrm{d}(\boldsymbol{x}) \approx \frac{1}{n} \sum_{k=0}^{n-1} f(\boldsymbol{x}_k)$$

with the sample points $\boldsymbol{x}_k \in P_n$ uniformly distributed over $[0, 1)^s$ and all weights equal to $1/n$.

Point sets to choose from

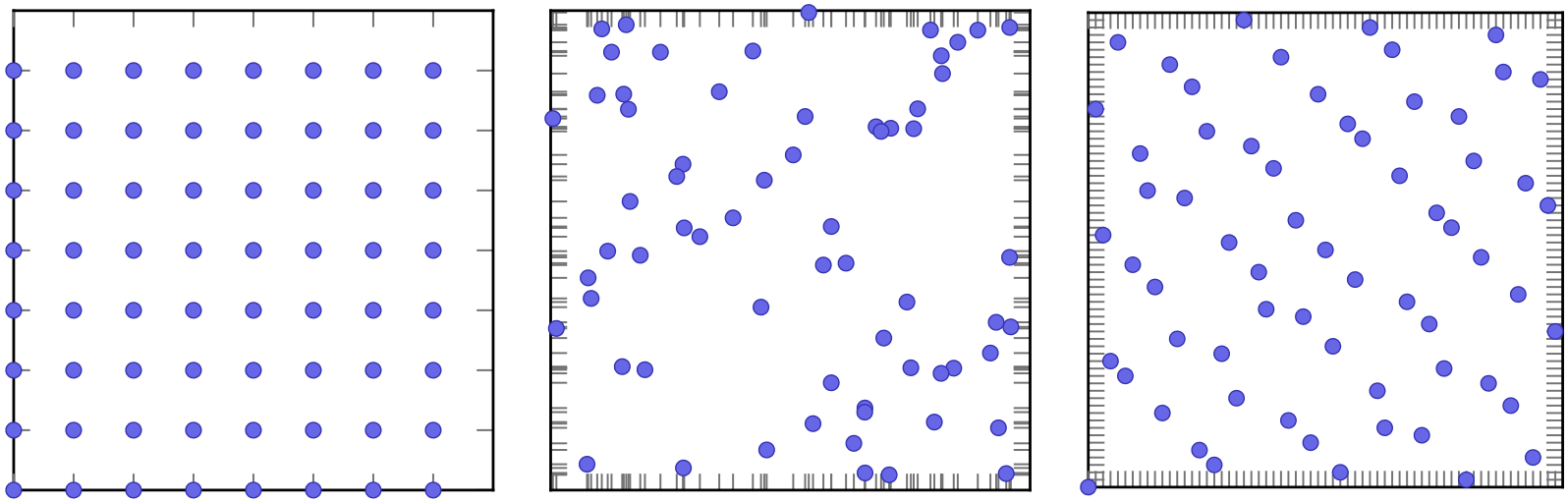


Figure: Three point sets with each 64 samples in the unit square.

1. The product left-rectangle rule. → Classical product rule
2. Pseudo-random numbers. → Monte Carlo
   (Fig: mt19937, the Mersenne Twister, with default initial state.)
3. *Low-discrepancy points*. → Quasi-Monte Carlo
   (Fig: Sobol' with direction numbers from Joe and Kuo.)

What discrepancy?

The *discrepancy* of a point set is a measure for its deviation from the uniform distribution.

1. The *local discrepancy* over a subset J of the unit cube $[0,1)^s$ is defined as

$$\operatorname{discr}(J, P_n) := \frac{|P_n \cap J|}{n} - \operatorname{vol}(J).$$

2. The *discrepancy* is then defined as an aggregate measure over local discrepancies.

Best known discrepancy is the (weighted) star discrepancy.

The *star discrepancy* considers only rectangular boxes $J \subseteq [0,1)^s$ anchored at $(0,0,\ldots,0)$.

The *weighted star discrepancy* attaches different weights $\gamma_{\mathfrak{u}}$, i.e., different importance, to different projections over coordinates in $\mathfrak{u}$.

For $1 \leq p \leq \infty$, $\mathcal{D}_s = \{1,\ldots,s\}$, (Hickernell; Sloan & Woźniakowski):

$$D_p^\star(P_n) := \left(\sum_{\emptyset \neq \mathfrak{u} \subseteq \mathcal{D}_s} \gamma_{\mathfrak{u}} \int_{[0,1)^{|\mathfrak{u}|}} \left| \mathrm{discr}([\mathbf{0}, \boldsymbol{x}_{\mathfrak{u}}), P_n(\mathfrak{u})) \right|^p \, \mathrm{d}\boldsymbol{x}_{\mathfrak{u}} \right)^{1/p},$$

$$D_\infty^\star(P_n) := \max_{\emptyset \neq \mathfrak{u} \subseteq \mathcal{D}_s} \gamma_{\mathfrak{u}} \sup_{\boldsymbol{x}_{\mathfrak{u}} \in [0,1)^{|\mathfrak{u}|}} \left| \mathrm{discr}\left([\mathbf{0}, \boldsymbol{x}_{\mathfrak{u}}), P_n(\mathfrak{u})\right) \right|.$$

The L_2 variant of the star discrepancy is equal to the worst-case error in an anchored Sobolev space.

Using the technique of reproducing kernel Hilbert spaces we can concentrate on the worst-case error.

Low-discrepancy points

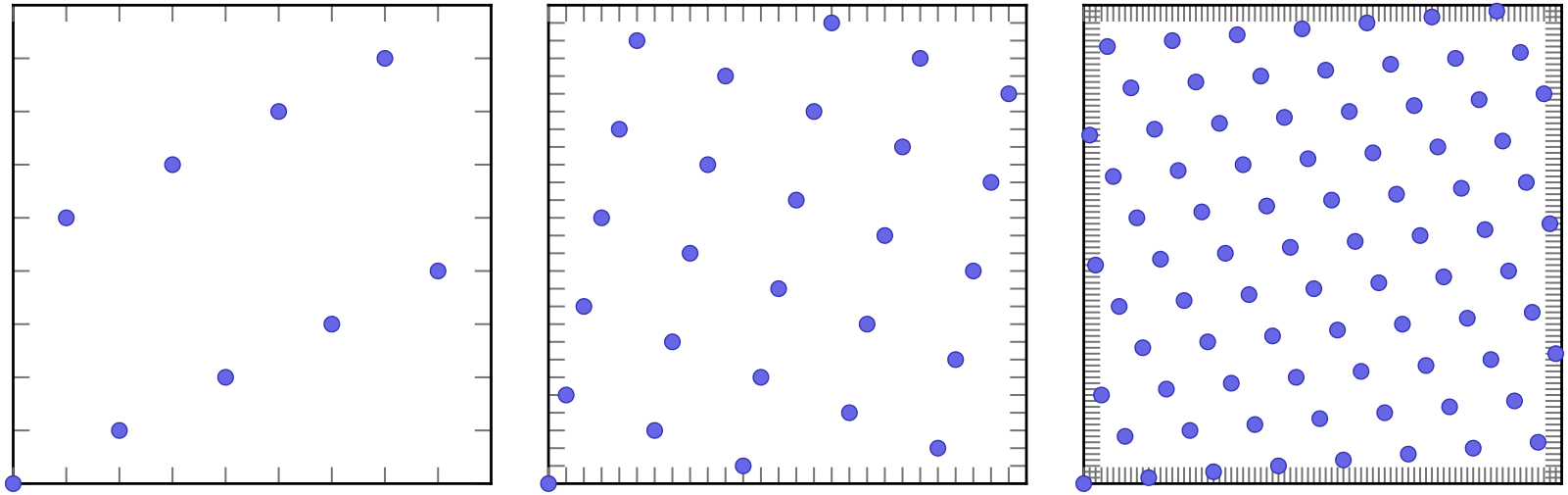
Two important types of low-discrepancy points are

1. *Lattice rule point sets (and sequences)*:
   A lattice rule has a point set "which forms an additive abelian subgroup of $\mathbb{R}^s/\mathbb{Z}^s$ under the operation of componentwise addition modulo 1."
   $\rightarrow$ Any integer combination of points gives a point in the set, or, if you have a basis, then all linear integer combinations are the points in the set.
2. *Digital nets (and sequences)*:
   A digital net or (t, m, s)-net in base b with b^m points has exactly b^t points in each elementary interval with volume b^{t-m} which is anchored and spans powers of the base b. (Smaller t is better.)

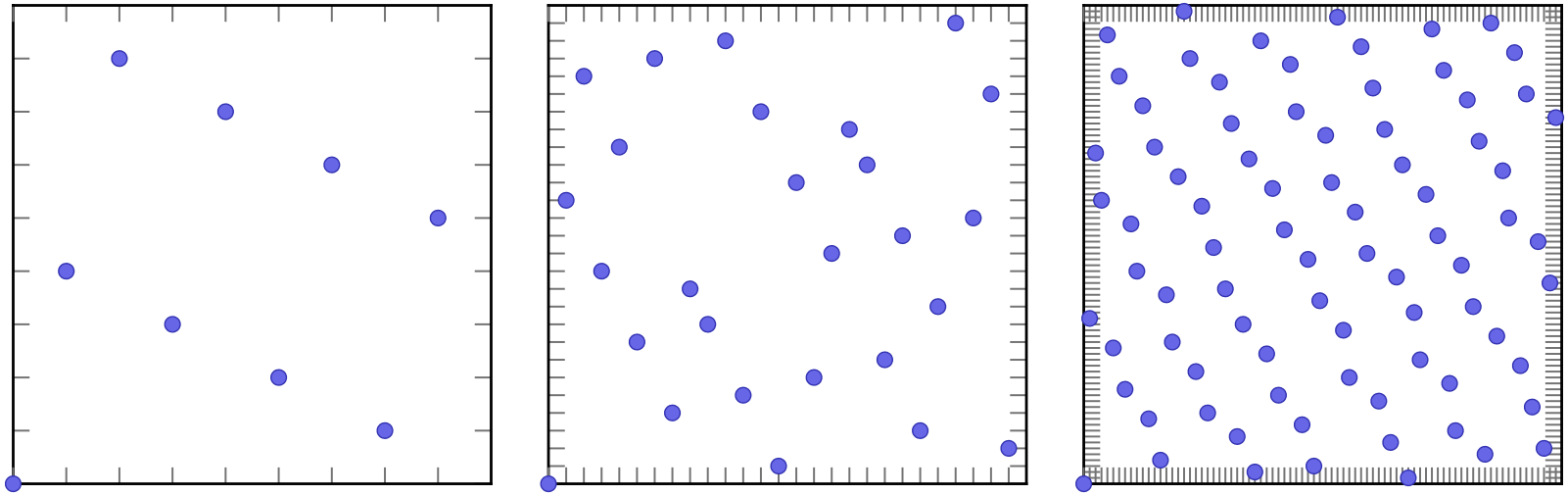
In both cases a *sequence* consists of specific subsequences having the right properties (anchored at and spanning powers of the base).

Point sets to choose from

1. A lattice sequence in base 3 with 9, 27 and 81 points:



2. A (t, s)-sequence (here Faure) in base 3 with 9, 27 and 81 points:



What are “good” lattice rules?

$\Rightarrow$ This talk is about lattice rules (but the same holds for other rules).

- A good lattice rule Q fulfills a good (e.g., optimal) bound on the worst-case error in a given function space $\mathcal{F}$.
- The *worst-case error* is defined as

$$e(Q,\mathcal{F}) := \sup_{\substack{f\in\mathcal{F}\\ \|f\|_{\mathcal{F}}\leq 1}} |I(f)-Q(f)|.$$

Calculating the worst-case error

- In a *reproducing kernel Hilbert space* $\mathcal{H}(K)$ the reproducing kernel $K(\boldsymbol{x},\boldsymbol{y})$ is able to reproduce $f \in \mathcal{H}(K)$:

$$f(\boldsymbol{x}) = \langle f(\cdot), K(\cdot,\boldsymbol{x})\rangle_{\mathcal{H}(K)}.$$

- The squared worst-case error in $\mathcal{H}(K)$ has the very nice form:

$$\begin{aligned} e^2(Q,K) &= \langle I(K) - Q(K), I(K) - Q(K)\rangle_{\mathcal{H}(K)} \\ &= I(I(K)) - 2Q(I(K)) + Q(Q(K)). \end{aligned}$$

- An important property (computationally and practically) is the *shift-invariant* property:

$$K(\{\boldsymbol{x}+\boldsymbol{\Delta}\},\{\boldsymbol{y}+\boldsymbol{\Delta}\}) = K(\boldsymbol{x},\boldsymbol{y}),$$
$$\text{since then} \quad K(\boldsymbol{x},\boldsymbol{y}) = K(\{\boldsymbol{x}-\boldsymbol{y}\},\boldsymbol{0}).$$

($\{x\}$ is a shorthand for the fractional part.)

Computationally it is interesting when

1. the kernel K is shift-invariant:
   $K(\boldsymbol{x},\boldsymbol{y}) = K(\{\boldsymbol{x}-\boldsymbol{y}\},\boldsymbol{0}) = K(\{\boldsymbol{x}-\boldsymbol{y}\})$,
2. the points form an abelian group, (e.g., Q is a lattice rule):
   $\{\boldsymbol{x}-\boldsymbol{y}\} \in P_n \qquad \forall \boldsymbol{x},\boldsymbol{y} \in P_n$,
3. (for simplicity) the operator norm of integration is 1:
   $\|I\|^2 = I(I(K)) = 1$,
4. (for simplicity) the s-dimensional space is a tensor-product space:
   $K(\boldsymbol{x}) = \prod_{j=1}^{s} \eta_j(x_j)$,

then the squared worst-case error has the very interesting form

$$e^2(Q,K) = -1 + \frac{1}{n}\sum_{k=0}^{n-1}\prod_{j=1}^{s}\eta_j(x_j^{(k)})$$

Function space settings

But first some function spaces. Two interesting settings:

1. Functions as Fourier-series (Korobov space):

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{h}\in\mathbb{Z}^s} \hat{f}(\boldsymbol{h})\ \exp(2\pi \mathrm{i}\ \boldsymbol{h}\cdot\boldsymbol{x}).$$

   $\rightarrow$ Periodic functions, natural setting for lattice rules.

2. Functions as Walsh-series (in base b) (Walsh space):

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{h}\in\mathbb{Z}_*^s} \hat{f}_b(\boldsymbol{h})\,\mathrm{wal}_{b,\boldsymbol{h}}(\boldsymbol{x}).$$

   In base 2 this is just a Haar-wavelet decomposition.
   $\rightarrow$ Walsh-series are used for *polynomial lattice rules* which are in fact digital nets (Dick & Pillichshammer).

Function space settings

If $\mathcal{H}(K)$ is a separable space the reproducing kernel is given by

$$K(x,y) = \sum_h \varphi_h(x)\ \overline{\varphi_h(y)},$$

for $\{\varphi_h\}_h$ an orthonormal basis for $\mathcal{H}(K)$.

The sums must be finite: weigh influence of basis functions by reducing higher "frequencies".

1. Kernel for Korobov space with smoothness $\alpha > 1$:

$$K(x,y) = 1 + \sum_{h \in \mathbb{Z}\setminus\{0\}} \frac{\exp(2\pi \mathrm{i}\ hx)}{|h|^{\alpha/2}}\ \frac{\exp(-2\pi \mathrm{i}\ hy)}{|h|^{\alpha/2}}.$$

2. Kernel for Walsh space with smoothness $\alpha > 1$:

$$K(x,y) = 1 + \sum_{\substack{h=1 \\ t=\lfloor \log_b(h) \rfloor}}^{\infty} \frac{\mathrm{wal}_{b,h}(x)}{(b^t)^{\alpha/2}}\ \frac{\overline{\mathrm{wal}_{b,h}(y)}}{(b^t)^{\alpha/2}}.$$

Function space settings

If $\mathcal{H}(K)$ is a separable space the reproducing kernel is given by

$$K(x,y) = \sum_h \varphi_h(x)\ \overline{\varphi_h(y)},$$

for $\{\varphi_h\}_h$ an orthonormal basis for $\mathcal{H}(K)$.

The sums must be finite: weigh influence of basis functions by reducing higher "frequencies".

1. Kernel for Korobov space with smoothness $\alpha > 1$:

$$K(x,y) = 1 + \sum_{h\in\mathbb{Z}\setminus\{0\}} \frac{\exp(2\pi \mathrm{i}\, h(x-y))}{|h|^\alpha} = K(\{x-y\}).$$

2. Kernel for Walsh space with smoothness $\alpha > 1$:

$$K(x,y) = 1 + \sum_{\substack{h=1 \\ t=\lfloor \log_b(h)\rfloor}}^{\infty} \frac{\mathrm{wal}_{b,h}(x)}{(b^t)^{\alpha/2}}\ \frac{\overline{\mathrm{wal}_{b,h}(y)}}{(b^t)^{\alpha/2}}.$$

Function space settings

If $\mathcal{H}(K)$ is a separable space the reproducing kernel is given by

$$K(x,y) = \sum_h \varphi_h(x)\ \overline{\varphi_h(y)},$$

for $\{\varphi_h\}_h$ an orthonormal basis for $\mathcal{H}(K)$.

The sums must be finite: weigh influence of basis functions by reducing higher "frequencies".

1. Kernel for Korobov space with smoothness $\alpha > 1$:

$$K(x,y) = 1 + \sum_{h\in\mathbb{Z}\setminus\{0\}} \frac{\exp(2\pi \mathrm{i}\, h(x-y))}{|h|^\alpha} = K(\{x-y\}).$$

2. Kernel for Walsh space with smoothness $\alpha > 1$:

$$K(x,y) = 1 + \sum_{\substack{h=1 \\ t=\lfloor \log_b(h)\rfloor}}^{\infty} \frac{\mathrm{wal}_{b,h}(x \ominus_b y)}{(b^t)^\alpha} = K(x \ominus_b y).$$

Function space settings

If $\mathcal{H}(K)$ is a separable space the reproducing kernel is given by

$$K(x,y) = \sum_h \varphi_h(x)\ \overline{\varphi_h(y)},$$

for $\{\varphi_h\}_h$ an orthonormal basis for $\mathcal{H}(K)$.

The sums must be finite: weigh influence of basis functions by reducing higher "frequencies".

1. Kernel for Korobov space with smoothness $\alpha > 1$: *shift-invariant*

$$K(x,y) = 1 + \sum_{h\in\mathbb{Z}\setminus\{0\}} \frac{\exp(2\pi \mathrm{i}\, h(x-y))}{|h|^\alpha} = K(\{x-y\}).$$

2. Kernel for Walsh space with smoothness $\alpha > 1$:

$$K(x,y) = 1 + \sum_{\substack{h=1 \\ t=\lfloor \log_b(h)\rfloor}}^{\infty} \frac{\mathrm{wal}_{b,h}(x \ominus_b y)}{(b^t)^\alpha} = K(x \ominus_b y).$$

Function space settings

If $\mathcal{H}(K)$ is a separable space the reproducing kernel is given by

$$K(x,y) = \sum_h \varphi_h(x)\ \overline{\varphi_h(y)},$$

for $\{\varphi_h\}_h$ an orthonormal basis for $\mathcal{H}(K)$.

The sums must be finite: weigh influence of basis functions by reducing higher "frequencies".

1. Kernel for Korobov space with smoothness $\alpha > 1$: *shift-invariant*

$$K(x,y) = 1 + \sum_{h\in\mathbb{Z}\setminus\{0\}} \frac{\exp(2\pi \mathrm{i}\, h(x-y))}{|h|^\alpha} = K(\{x-y\}).$$

2. Kernel for Walsh space with smoothness $\alpha > 1$: *digital shift-invariant*

$$K(x,y) = 1 + \sum_{\substack{h=1 \\ t=\lfloor \log_b(h)\rfloor}}^{\infty} \frac{\mathrm{wal}_{b,h}(x \ominus_b y)}{(b^t)^\alpha} = K(x \ominus_b y).$$

Sobolev spaces

There are two popular Sobolev spaces in use in the literature:

1. The *anchored Sobolev space*, where constant parts in the norm are evaluated at a fixed anchor.
2. The *unanchored Sobolev space*, where the constant parts in the norm are integrated out.

Both can be written in terms of the Korobov kernel when made shift-invariant:

$$K^{\text{shinv}}(\{\boldsymbol{x}-\boldsymbol{y}\}) = \int_{[0,1)^s} K(\{\boldsymbol{x}+\boldsymbol{\Delta}\},\{\boldsymbol{y}+\boldsymbol{\Delta}\})\,\mathrm{d}\boldsymbol{\Delta}.$$

And both can be written in terms of the Walsh kernel when made digital shift-invariant:

$$K^{\text{ds}}(\boldsymbol{x} \ominus_b \boldsymbol{y}) = \int_{[0,1)^s} K(\boldsymbol{x} \oplus_b \boldsymbol{\Delta}, \boldsymbol{y} \oplus_b \boldsymbol{\Delta})\,\mathrm{d}\boldsymbol{\Delta}.$$

Why shift-invariant?

Why this interest in shift-invariant spaces?

1. Computationally this will be advantageous as we see in a minute.
   $\rightarrow$ Fast construction.
2. We like to use a small number of randomly shifted point sets to obtain a statistical error estimate.
   $\rightarrow$ Ease of use.
3. The expected value of the squared worst-case error in a space K is the squared worst-case error in the associated shift-invariant space K^{shinv}.

Weighted function spaces

Consider weighted shift-invariant kernels of the form

$$K(\boldsymbol{x},\boldsymbol{y}) = K(\{\boldsymbol{x}-\boldsymbol{y}\}) = \sum_{u\subseteq\mathcal{D}_s} \gamma_u \prod_{j\in u} \omega(\{x_j - y_j\}),$$

matching an ANOVA decomposition of a function

$$f(\boldsymbol{x}) = \sum_{u\subseteq\mathcal{D}_s} f_u(\boldsymbol{x}_u) = f_\emptyset + \sum_{j=1}^{s} f_{\{j\}}(x_j) + \sum_{j=1}^{s}\sum_{i=j+1}^{s} f_{\{j,i\}}(x_j,x_i) + \cdots$$

The weights γ_u model our idea of importance of the different parts.

(Dick, Sloan, Wang & Woźniakowski; Hickernell)

Two important weight settings are:
(assuming shift-invariant kernel, Q a lattice rule and $\|I\| = 1$)

1. *Product weights*: $\gamma_{\mathfrak{u}} = \prod_{j\in\mathfrak{u}} \gamma_{\{j\}}$ with squared worst-case error:

$$e^2(Q,K) = -1 + \frac{1}{n}\sum_{k=0}^{n-1}\prod_{j=1}^{s}(1+\gamma_{\{j\}}\,\omega(x_j^{(k)}))$$

2. *Order-dependent weights*: $\gamma_{\mathfrak{u}} = \Gamma_{|\mathfrak{u}|}$ with squared worst-case error:

$$e^2(Q,K) = \frac{1}{n}\sum_{k=0}^{n-1}\sum_{\ell=1}^{s}\Gamma_\ell \sum_{\substack{\mathfrak{u}\subseteq\mathcal{D}_s\\|\mathfrak{u}|=\ell}}\prod_{j\in\mathfrak{u}}\omega(x_j^{(k)}).$$

This is called of *finite order* $q^\star$ if $\Gamma_\ell = 0$ for $\ell > q^\star$.

Component-by-component construction of good lattice rules

The two expressions for the squared worst-case error on the previous slide will now be in the spot light.
First note that they can be written iteratively in the number of dimensions. For product weights:

$$e_s^2 = -1 + \frac{1}{n}\sum_{k=0}^{n-1}(1+\gamma_{\{s\}}\,\omega(x_s^{(k)}))\,p_{s-1}(k)$$

and (less obvious) for order-dependent weights:

$$e_s^2 = e_{s-1}^2 + \frac{1}{n}\sum_{k=0}^{n-1}\omega(x_s^{(k)})\left(\sum_{\ell=1}^{\min(q^\star,s)}\Gamma_\ell\,p_{s-1,\ell-1}(k)\right)$$

using $O(n)$ and $O(nq^\star)$ memory respectively.

The point set of a rank-1 lattice generated by the integer vector $\boldsymbol{z}$ is given by

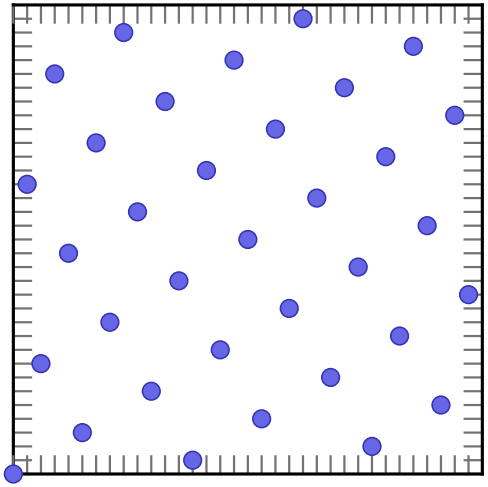
$$P_n(\boldsymbol{z}) := \left\{ \left\{ \frac{k\,\boldsymbol{z}}{n} \right\} : k \in \mathbb{Z}_n \right\} = \left\{ \frac{k \cdot \boldsymbol{z} \bmod n}{n} : k \in \mathbb{Z}_n \right\}$$

and the components of $\boldsymbol{z}$ are integers from the set

$$U_n := \{v \in \mathbb{Z}_n : \gcd(v, n) = 1\} = \mathbb{Z}_n^{\times}.$$

Every one-dimensional projection is the equally spaced set $\{k/n : k = 0, \ldots, n-1\}$ and the components of $\boldsymbol{z}$ can be interpreted as *permutations* of this set.

$\rightarrow$ Search the z_j "component by component."
(Sloan, Joe, Kuo, ...)



Component-by-component construction first finds z_1, minimizing the worst-case error, then z_2, minimizing the worst-case error, and so on...

Component-by-component construction (Sloan & Reztsov)

- has proven to be a very valuable technique;
- achieves rules with optimal rate of convergence in weighted Korobov space, i.e., $O(n^{-\alpha/2+\delta})$, $\delta > 0$; (Kuo; Dick)
- achieves rules with optimal rate of convergence in weighted Sobolev space, i.e., $O(n^{-1+\delta})$, $\delta > 0$; (Kuo; Dick)
- ... (see previous talk) ...

Moreover, the cost of component-by-component construction is only $O(s(n\log(n) + tn))$ using $O(tn)$ memory, where $t = 1$ for product weights and $t = q^\star$ for order-dependent weights of order $q^\star$.
(N. & Cools)

Prime number of points

Let's look back at the formula

$$e_s^2(z_s) = -1 + \frac{1}{n}\sum_{k=0}^{n-1}\left(1 + \gamma_{\{s\}}\,\omega\left(\frac{k\cdot z_s \bmod n}{n}\right)\right) p_{s-1}(k).$$

The interesting part is

$$y(z_s) = \sum_{k=0}^{n-1}\omega\left(\frac{k\cdot z_s \bmod n}{n}\right) p_{s-1}(k)$$

which needs to be calculated for all $z_s \in U_n$.

$\Rightarrow$ This is a matrix-vector product:

$$y = \Omega_n\, p_{s-1}.$$

Prime number of points

Let's look back at the formula

$$e_s^2(z_s) = -1 + \frac{1}{n}\sum_{k=0}^{n-1}\left(1 + \gamma_{\{s\}}\,\omega\left(\frac{k\cdot z_s \bmod n}{n}\right)\right) p_{s-1}(k).$$

The interesting part is

$$y(z_s) = \sum_{k=0}^{n-1}\omega\left(\frac{k\cdot z_s \bmod n}{n}\right) p_{s-1}(k)$$

which needs to be calculated for all $z_s \in U_n$.

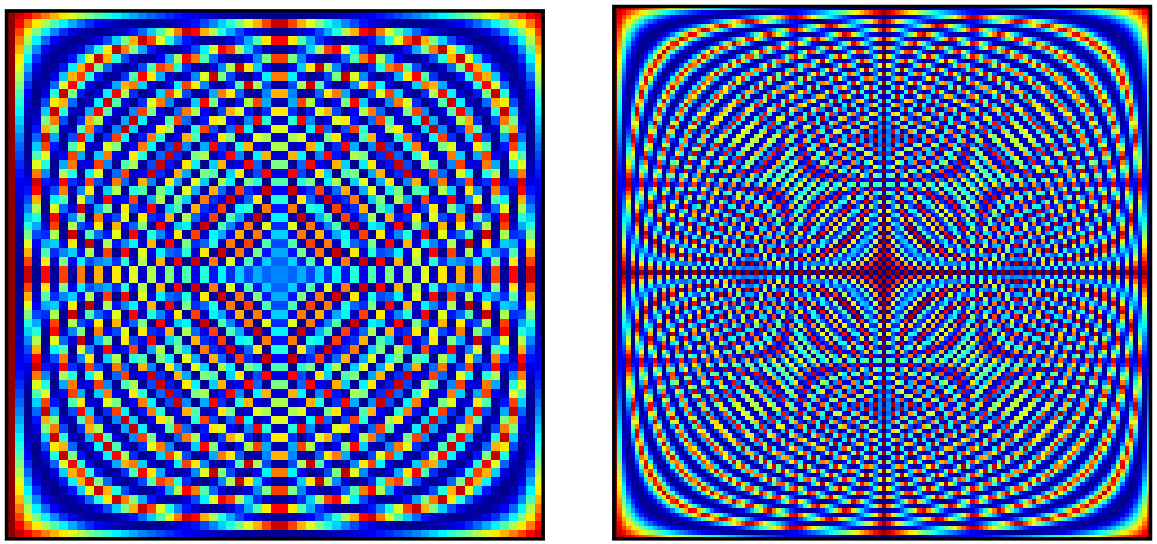
$\Rightarrow$ This is a matrix-vector product:

$$\boldsymbol{y} = \boldsymbol{\Omega}_n\,\boldsymbol{p}_{s-1}.$$

Define the matrix

$$\boldsymbol{\Omega}_n := \left[\omega\left(\left\{\frac{k\,z}{n}\right\}\right)\right]_{\substack{z\in U_n\\ k\in\mathbb{Z}_n}} = \left[\omega\left(\frac{k\cdot z \bmod n}{n}\right)\right]_{\substack{z\in U_n\\ k\in\mathbb{Z}_n}}$$

which looks like the image on the left (for $n = 61$):



(The image on the right is the matrix for $n = 122$, but z taken from just $\mathbb{Z}_n \setminus \{0\}$ instead of U_n.)

Remember

$$U_n := \{v \in \mathbb{Z}_n : \gcd(v, n) = 1\} = \mathbb{Z}_n^\times, \qquad |U_n| = \varphi(n).$$

When n is prime we can find a generator g for U_n such that

$$\begin{aligned}\langle g \rangle &= \{g^k \bmod n : k \in \mathbb{Z}\} \\ &= \{g^0, g^1, \ldots, g^{\varphi(n)-1}\} \bmod n.\end{aligned}$$

And we could of course also take negative powers of g

$$\begin{aligned}&g^0, g^{-1}, g^{-2}, \ldots, g^1 \\ &\quad \equiv g^0, g^{\varphi(n)-1}, g^{\varphi(n)-2}, \ldots, g^1 \pmod n.\end{aligned}$$

Prime number of points

When n is prime $\mathbb{Z}_n = U_n \cup \{0\}$ ... and $\mathbf{\Omega}_n$ is basically just homomorph to the multiplication table of the group U_n.
E.g., for $n = 7$:

$\cdot$	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	1	3	5
3	3	6	2	5	1	4
4	4	1	5	2	6	3
5	5	3	1	6	4	2
6	6	5	4	3	2	1

$\mapsto$

$\cdot$	1	5	4	6	2	3
1	1	5	4	6	2	3
3	3	1	5	4	6	2
2	2	3	1	5	4	6
6	6	2	3	1	5	4
4	4	6	2	3	1	5
5	5	4	6	2	3	1

Multiplication modulo n is so much easier using a representation in powers of a generator g...

$$g^{\alpha} \cdot g^{\beta} \equiv g^{\alpha+\beta \pmod{\varphi(n)}} \pmod{n}.$$

$\Rightarrow$ The multiplication table now takes the form of a circulant matrix.

Prime number of points

When n is prime $\mathbb{Z}_n = U_n \cup \{0\}$ ... and $\mathbf{\Omega}_n$ is basically just homomorph to the multiplication table of the group U_n.
E.g., for $n = 7$:

$\cdot$	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	1	3	5
3	3	6	2	5	1	4
4	4	1	5	2	6	3
5	5	3	1	6	4	2
6	6	5	4	3	2	1

$\mapsto$

$\cdot$	1	5	4	6	2	3
1	1	5	4	6	2	3
3	3	1	5	4	6	2
2	2	3	1	5	4	6
6	6	2	3	1	5	4
4	4	6	2	3	1	5
5	5	4	6	2	3	1

Multiplication modulo n is so much easier using a representation in powers of a generator g...

$$g^{\alpha} \cdot g^{\beta} \equiv g^{\alpha+\beta \pmod{\varphi(n)}} \pmod{n}.$$

$\Rightarrow$ The multiplication table now takes the form of a circulant matrix.

When n is prime $\mathbb{Z}_n = U_n \cup \{0\}$ ... and $\mathbf{\Omega}_n$ is basically just homomorph to the multiplication table of the group U_n.
E.g., for $n = 7$:

$\cdot$	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	1	3	5
3	3	6	2	5	1	4
4	4	1	5	2	6	3
5	5	3	1	6	4	2
6	6	5	4	3	2	1

$\mapsto$

$\cdot$	1	5	4	6	2	3
1	1	5	4	6	2	3
3	3	1	5	4	6	2
2	2	3	1	5	4	6
6	6	2	3	1	5	4
4	4	6	2	3	1	5
5	5	4	6	2	3	1

Multiplication modulo n is so much easier using a representation in powers of a generator g...

$$g^{\alpha} \cdot g^{\beta} \equiv g^{\alpha+\beta \pmod{\varphi(n)}} \pmod{n}.$$

$\Rightarrow$ The multiplication table now takes the form of a circulant matrix.

When n is prime $\mathbb{Z}_n = U_n \cup \{0\}$ ... and $\mathbf{\Omega}_n$ is basically just homomorph to the multiplication table of the group U_n.
E.g., for $n = 7$:

$\cdot$	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	1	3	5
3	3	6	2	5	1	4
4	4	1	5	2	6	3
5	5	3	1	6	4	2
6	6	5	4	3	2	1

$\mapsto$

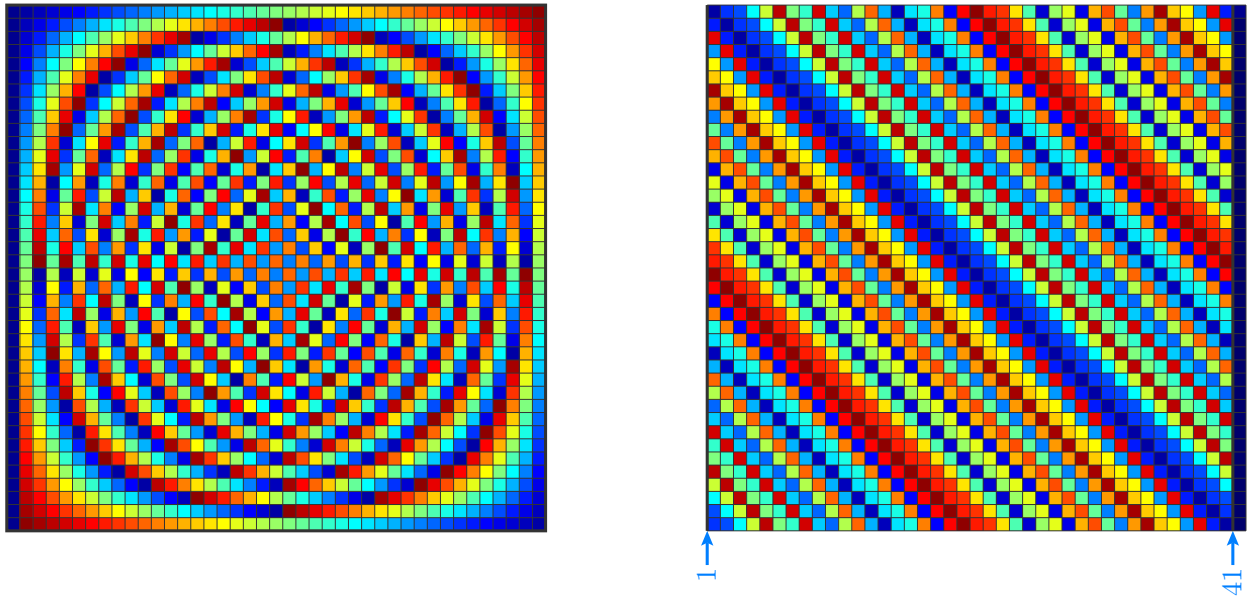
$\cdot$	1	5	4	6	2	3
1	1	5	4	6	2	3
3	3	1	5	4	6	2
2	2	3	1	5	4	6
6	6	2	3	1	5	4
4	4	6	2	3	1	5
5	5	4	6	2	3	1

Multiplication modulo n is so much easier using a representation in powers of a generator g...

$$g^{\alpha} \cdot g^{\beta} \equiv g^{\alpha+\beta \pmod{\varphi(n)}} \pmod{n}.$$

$\Rightarrow$ The multiplication table now takes the form of a circulant matrix.

This is just a permutation on the rows and columns:



- The CBC algorithm multiplies with this matrix in each step.
- Matrix-vector multiplication with a circulant matrix $\boldsymbol{C}_m$ of size $m \times m$ takes $O(m \log(m))$ (instead of $O(m^2)$):

$$\boldsymbol{C}_m \, \boldsymbol{x} = \mathrm{IFFT}(\mathrm{FFT}(\boldsymbol{c}) .* \mathrm{FFT}(\boldsymbol{x})).$$

$\Rightarrow$ Fast construction of good lattice rule in time $O(sn \log(n))$.

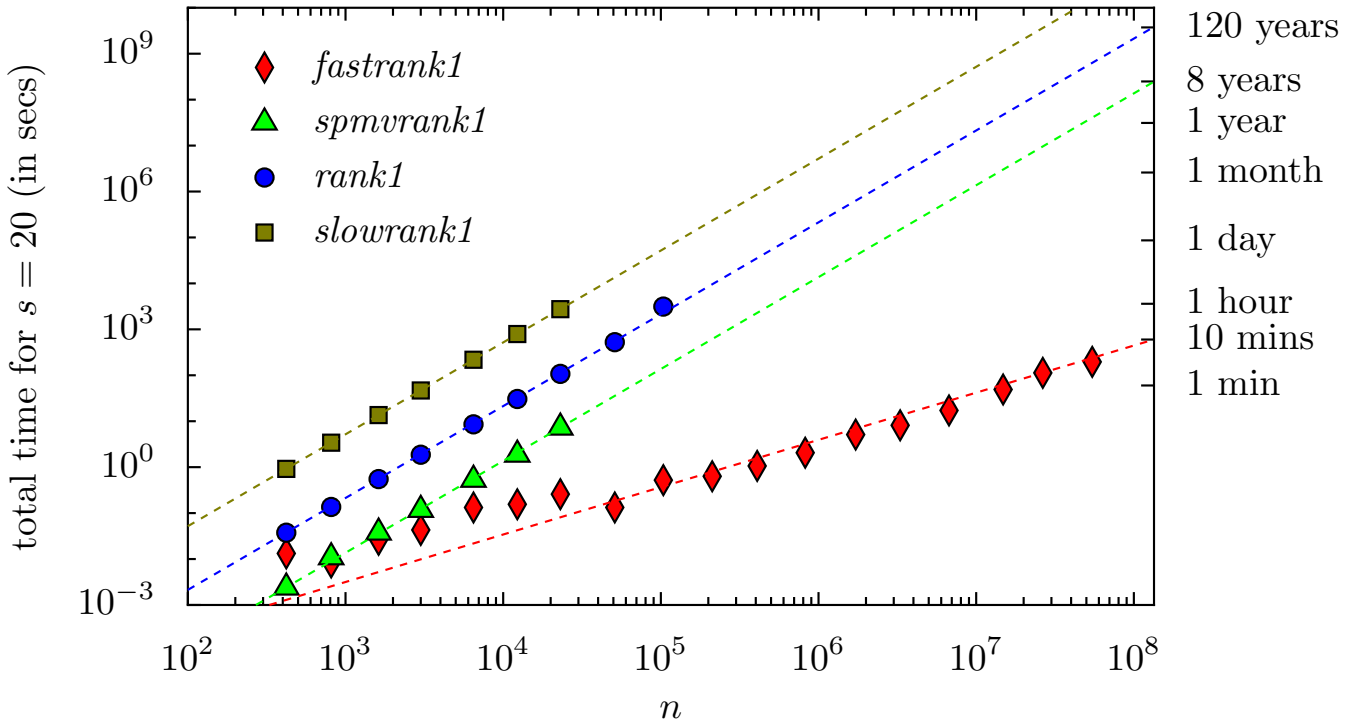


Figure: Timings generated on a P4 2.4GHz ht, 2GB RAM for 20 dimensions

Notes:

- This is known for prime number FFTs as *Rader factorization*.
- Normal FFT tricks do not work here since in general $\omega(ab) \neq \omega(a)\omega(b)$.
- Also works for polynomial lattice rules modulo an irreducible polynomial ($\simeq$ prime number $\rightarrow$ field).

Non-prime number of points

The technique for prime n makes use of a generator for U_n, so...

- The multiplicative group $U_n = \{z \in \mathbb{Z}_n : \gcd(z, n) = 1\} = \mathbb{Z}_n^\times$, $|U_n| = \varphi(n)$, is cyclic whenever

$$n = 2,\ 4,\ p^k \text{ or } 2p^k,$$

  with p an odd prime. A generator for the cyclic group U_n is called a *primitive root* modulo n.

- Every group U_n, $n = n_1 n_2 \cdots n_r$, all relatively prime, can be written as

$$U_n \simeq \begin{cases} U_{n_1} \oplus U_{n_2} \oplus \cdots \oplus U_{n_r} & \text{if } 8 \nmid n, \\ (-\langle 5 \rangle_{2^k} \oplus \langle 5 \rangle_{2^k}) \oplus U_{n_2} \oplus \cdots \oplus U_{n_r} & \text{if } 8 \mid n,\ n_1 = 2^k,\ k \geq 3, \end{cases}$$

  where every subgroup is a cyclic multiplicative group.

- A "generator" for U_n is now obtained by using the Chinese remainder theorem to resolve the isomorphism:

$$v \bmod n \simeq (g_1^{\alpha_1} \bmod n_1, g_2^{\alpha_2} \bmod n_2, \ldots, g_r^{\alpha_r} \bmod n_r).$$

- Now remember:

$$\boldsymbol{\Omega}_n = \left[\omega \left(\frac{k \cdot z \bmod n}{n} \right) \right]_{\substack{z \in U_n \\ k \in \mathbb{Z}_n}}$$

  So a permutation on the columns to group the $v \in \mathbb{Z}_n$ that are also in U_n gives us a first "nested block circulant" block of size $\varphi(n) \times \varphi(n)$ (using a permutation based on the $g_1, g_2, \ldots, g_r$).
- What to do with the rest of $\mathbb{Z}_n$?

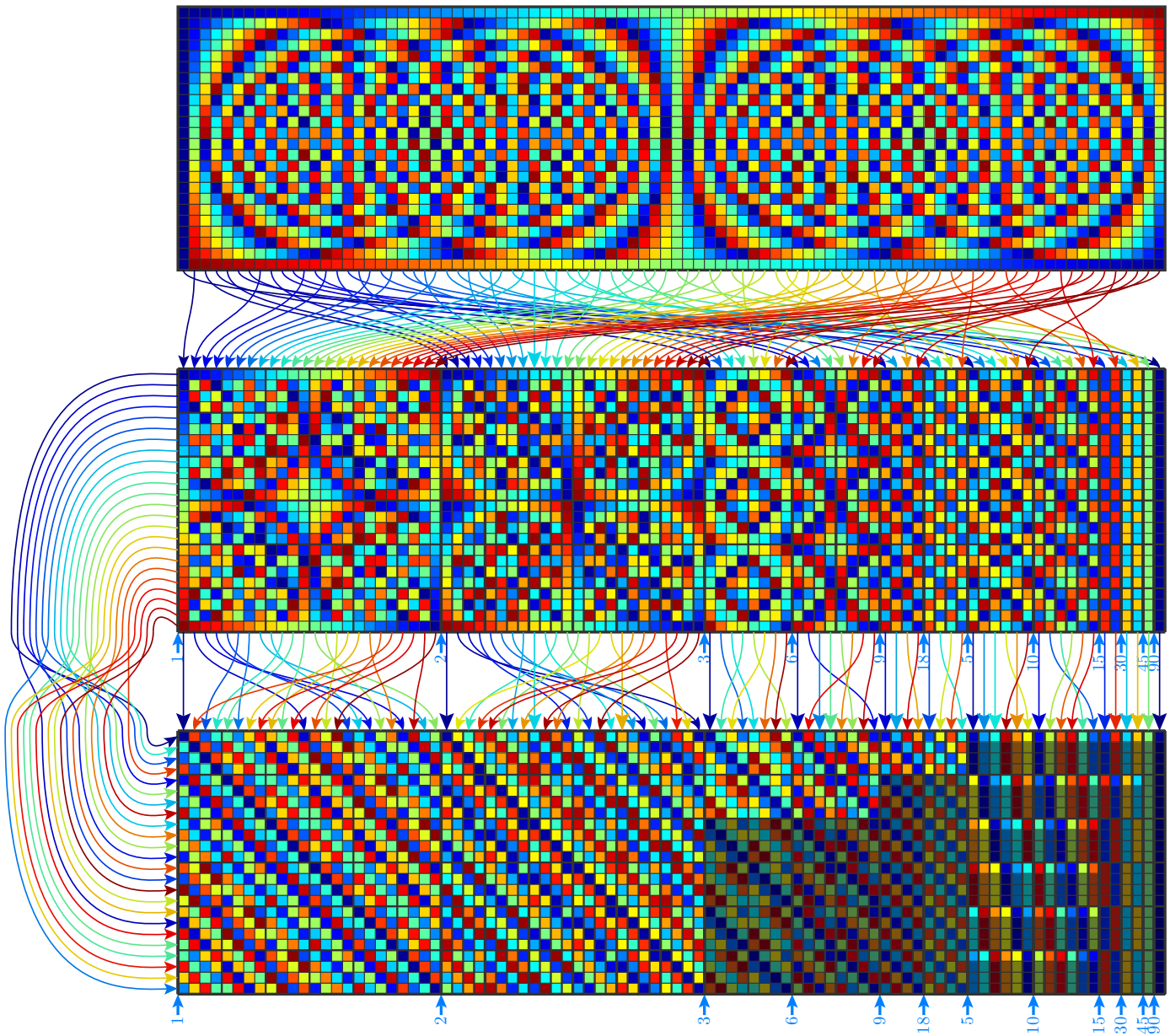
- Using the partition

$$\mathbb{Z}_n = \bigcup_{d|n} d\, U_n,$$

  we can do the trick from the previous slide for every divisor of n.
- It turns out that the complete matrix $\boldsymbol{\Omega}_n$ can be permuted into nested block circulant form for each divisor block with *the same permutation*.
- An example for $n = 90 = 2 \times 5 \times 3^2$ is on the next slide...
- A fast matrix-vector product is possible in $O(n \log(n))$ (although this is a little bit complicated).

$\Rightarrow$ Fast construction of a good lattice rule with n composite.

Non-prime number of points



Prime powers and sequences

Now take $n = p^{m_2}$, a prime power, and set

$$P_{p^m} = \left\{ \frac{k \cdot \boldsymbol{z} \bmod p^m}{p^m} : k \in \mathbb{Z}_{p^m} \right\}, \qquad 0 \le m \le m_2,$$

then the point set is embedded

$$P_0 \subset P_p \subset P_{p^2} \subset \cdots \subset P_{p^{m_2}}.$$

For a sequence we need:

1. each embedded point set, P_{p^m}, to be a good lattice rule;
2. every subsequence anchored at and spanning a power of the base,

$$\boldsymbol{x}_{k\,p^m}, \boldsymbol{x}_{k\,p^m+1}, \ldots, \boldsymbol{x}_{(k+1)\,p^m-1},$$

be a good lattice.

For property 1 (*every embedded rule is a good rule*) we just need to alter the CBC algorithm to minimize

$$X_{m_1,m_2,s}(\boldsymbol{z}) := \max_{m_1 \le m \le m_2} \frac{e_{p^m,s}(\boldsymbol{z})}{e_{p^m,s}(\boldsymbol{z}^{(m)})},$$

with $\boldsymbol{z}^{(m)}$ the optimal generating vector for $n = p^m$ according to the CBC algorithm (Cools, Kuo, N.).

Suppose r is a generator for U_p, then

$$g = \begin{cases} r + p & \text{if } r^{p-1} \equiv 1 \pmod{p^2}, \\ r & \text{otherwise,} \end{cases}$$

is a generator for all U_{p^ℓ}.

If looking for the smallest generator, case 1 happens for the first time at $p = 40487$.

$\rightarrow \boldsymbol{\Omega}_{p^{m_2}}$ is also embedded: calculating $e_{p^{m_2},s}$ reveals all $e_{p^m,s}$, $m \le m_2$.

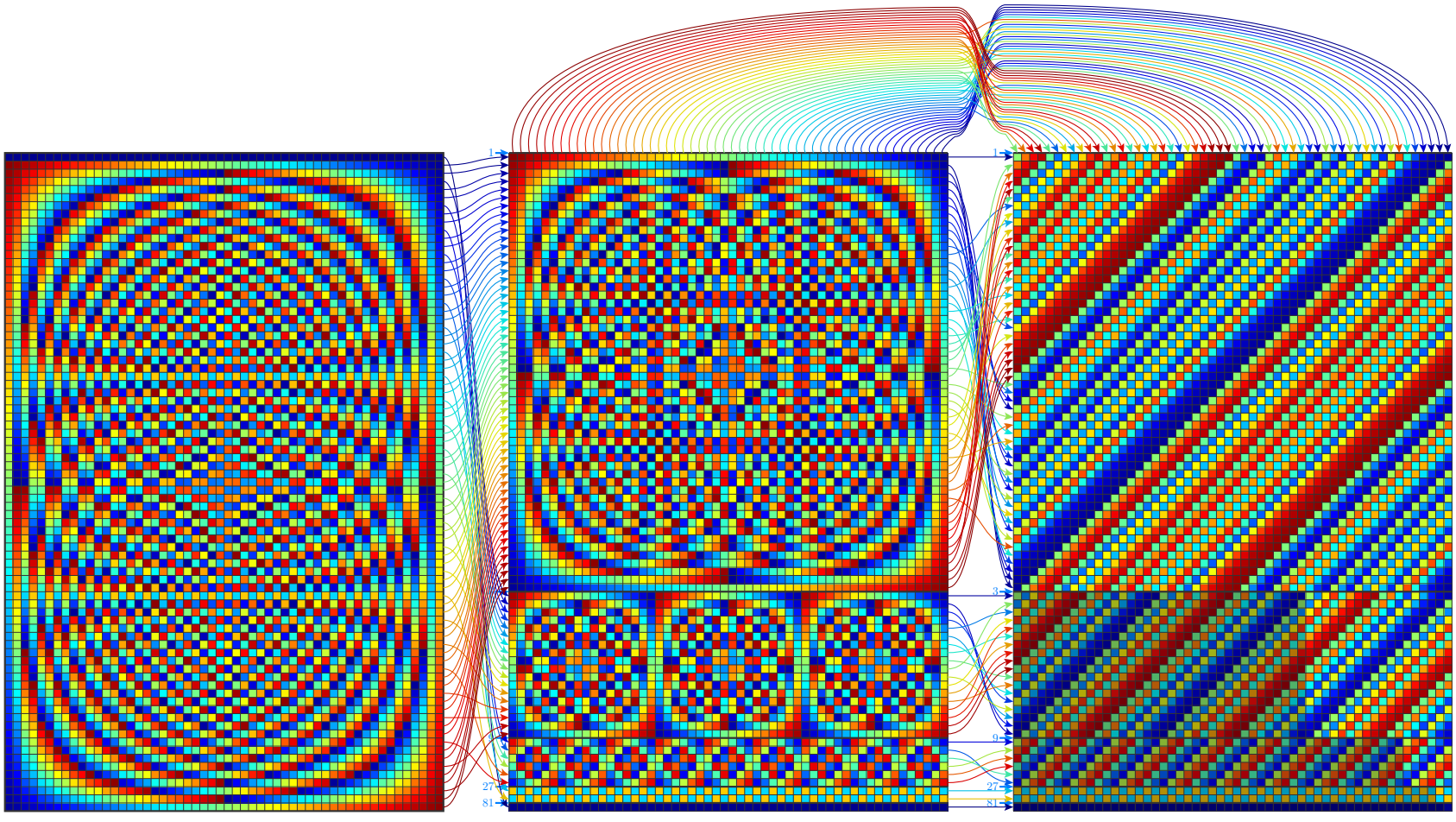
For property 2 (*every subsequence is a good rule*), which in effect means you can use the sequence *point by point*, we use

$$\boldsymbol{x}_k = \left\{ \frac{\varphi_p(k)\,\boldsymbol{z}}{p^{m_2}} \right\} \qquad \text{instead of} \qquad \boldsymbol{x}_k = \left\{ \frac{k\,\boldsymbol{z}}{p^{m_2}} \right\}.$$

- The φ_p is a permutation which keeps the embedding and has the effect that the sequence consists of smaller *shifted* rules.
- $\rightarrow$ These are good lattice rules if they have size $\geq p^{m_1}$: by construction and since we use a shift-invariant worst-case error.
- These permutations are well known for (t, s)-sequences, e.g., radical inverse and Gray ordering.

$\Rightarrow$ Fast construction of good lattice sequences.

Prime powers and sequences



Implementing lattice sequences

- The biggest advantage of lattice rules is its ease of implementation. Points are just generated by

$$\boldsymbol{x}_k = \left\{ \frac{k\boldsymbol{z}}{n} \right\}.$$

- This is no different for lattice sequences:

$$\boldsymbol{x}_k = \left\{ \frac{\varphi_p(k)\,\boldsymbol{z}}{p^{m_2}} \right\},$$

  where $\varphi_p(k)$ is easy for both radical inverse and Gray ordering.
- The speed to generate points from a lattice sequence is comparable with a fast and good pseudo random number generator (e.g., mt19937).

In contrast: implementing (t, s)-sequences or (t, m, s)-nets is more involved and they are generally a lot slower.

Fast constructions

- Implementing the fast construction algorithm can be tricky.
- Using a high level language (e.g., Matlab or Python) the code for the fast CBC algorithm fits easily on one page.
- Experimenting with the fast construction is fun!
  (You get results in fractions of seconds.)
- It is possible to do just in time construction.
- Hopefully this will learn us something about the weights...

→ Software available at my homepage.

A periodizing application

Consider the approximation of multivariate normal probabilities

$$N(\boldsymbol{a}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{|\boldsymbol{\Sigma}|(2\pi)^s}} \int_{-\infty}^{a_1} \cdots \int_{-\infty}^{a_s} \exp(-\tfrac{1}{2}\, \boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{x})\, \mathrm{d}\boldsymbol{x}.$$

- This is hard for dimensions > 3.
- A lot of work on this has been done by Alan Genz.
- Matlab contains code by Alan Genz using a low-discrepancy sequence. (A Kronecker sequences which is used as an "infinite lattice sequence".)
- We will use a classical trick for lattice rules: *periodization*...
  ... on lattice sequences.

Periodization

- Using a periodizing transform:

$$\int_{[0,1)^s} f(x_1, x_2, \ldots, x_s)\, \mathrm{d}\boldsymbol{x}$$
$$= \int_{[0,1)^s} f(\varphi(x_1), \varphi(x_2), \ldots, \varphi(x_s))\, \varphi'(x_1)\varphi'(x_2)\cdots\varphi'(x_s)\, \mathrm{d}\boldsymbol{x}.$$

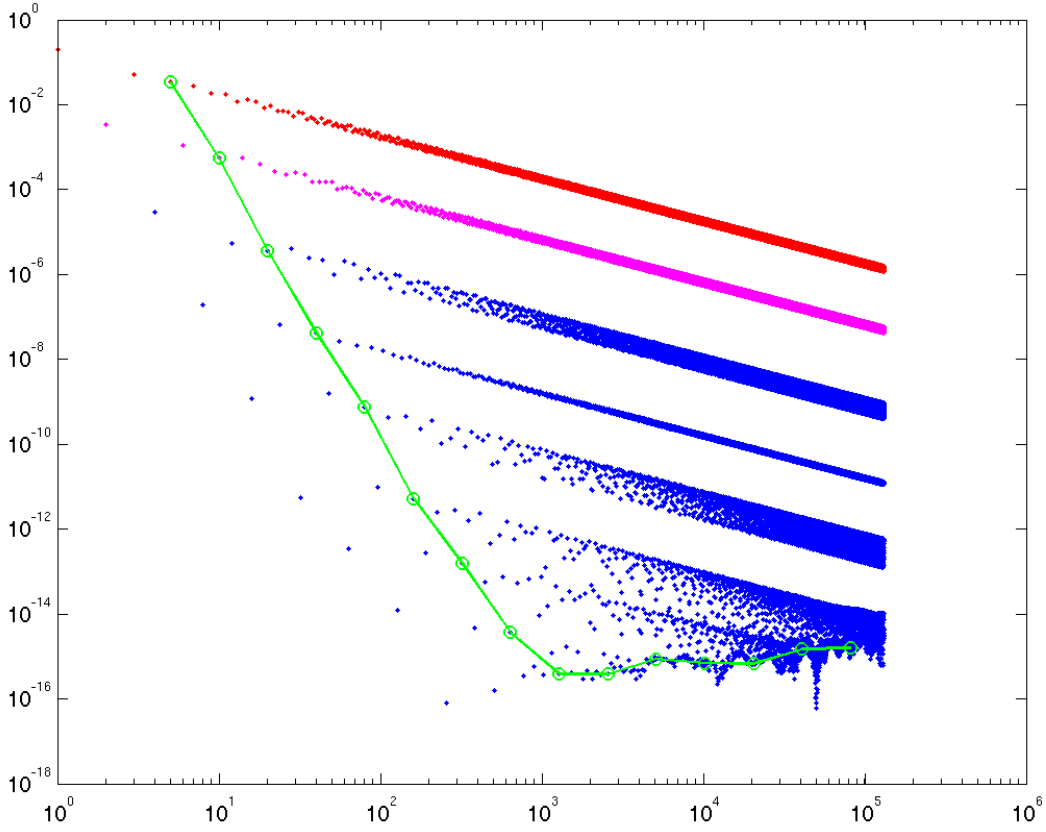
- However there is a warning: Using

$$|I(f) - Q(f)| = \langle f, \xi \rangle_{\mathcal{H}(K)} \leq \|f\|_{\mathcal{H}(K)}\, e(Q, K)$$

as an error bound, it can be shown that the worst-case error is better for a periodic function space but the norm of the periodized function can be much worse (Hickernell).

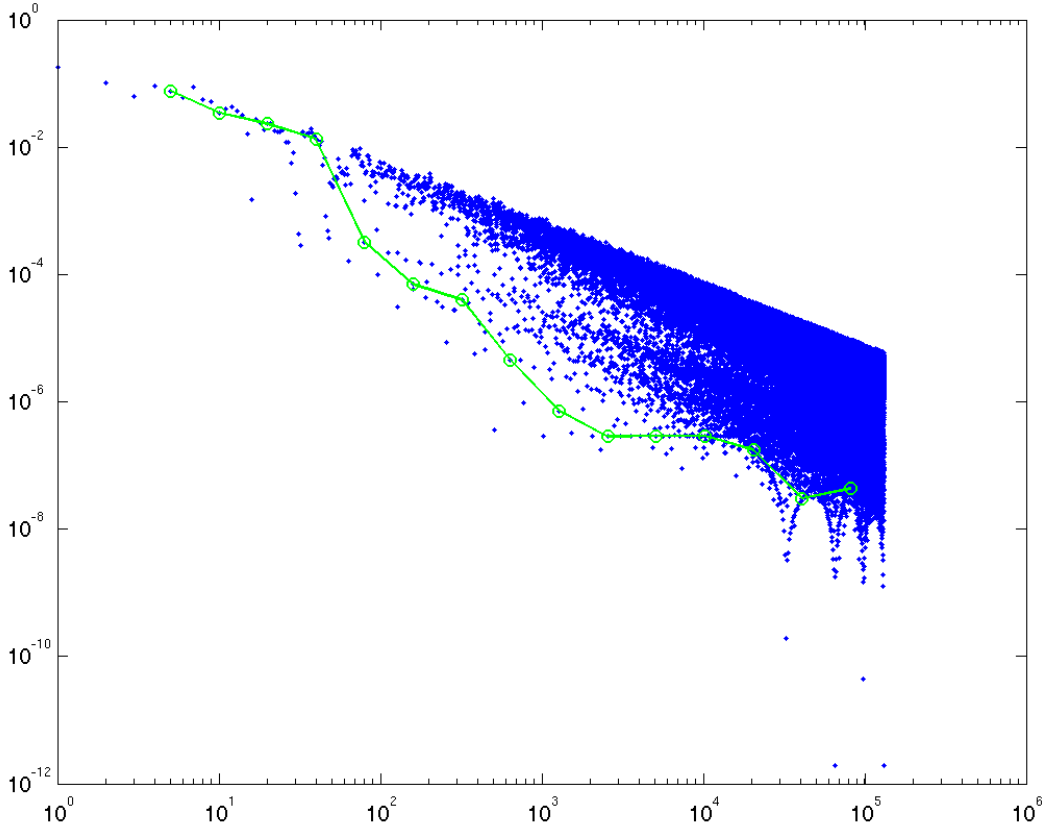
- Such a transformation is a smooth increasing function from $[0,1)$ to $[0,1)$ and all derivatives up to α are equal at the sides, $\varphi^{(k)}(0) = \varphi^{(k)}(1)$, $1 \le k \le \alpha$.
  E.g. one of the Sidi transforms:
$$\varphi_2(t) = \tfrac{1}{2\pi}(2\pi t - \sin(2\pi t)).$$
- We actually get a weighted cubature rule
$$Q_{\mathrm{per}}(f) := \frac{1}{n}\sum_{k=0}^{n-1}\left(\prod_{j=1}^{s}\varphi'(\boldsymbol{x})\right) f(\varphi(\boldsymbol{x})).$$
- For a lattice sequence the weights will be unbalanced. I.e., we want
$$\frac{1}{n}\sum_{k=0}^{n-1}\left(\prod_{j=1}^{s}\varphi'(\boldsymbol{x})\right) = 1.$$
  This will be true for the full lattices but not necessarily in between.

In 2 dimensions



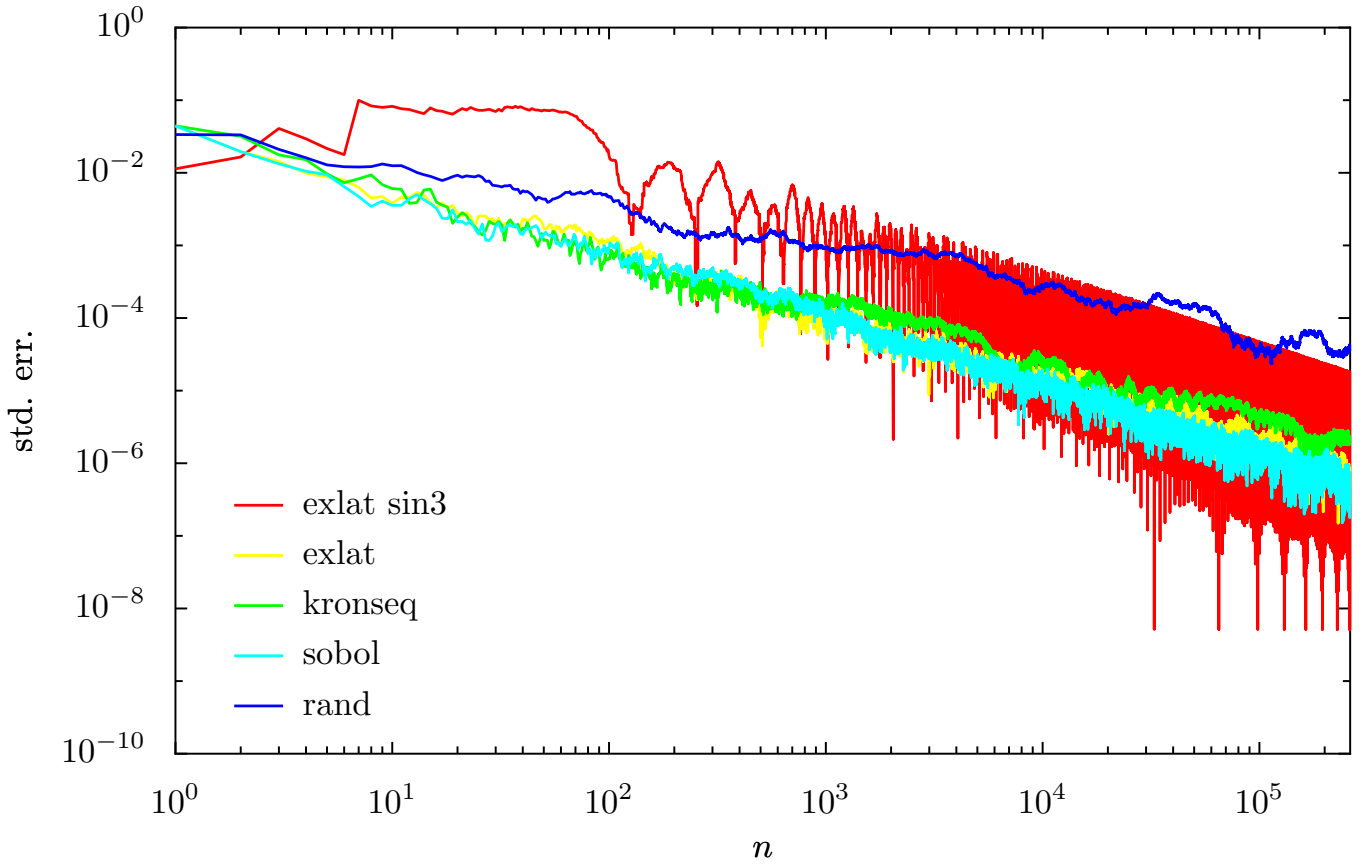
Number theoretic structure in the convergence graph! For lattice sequence in base 2: we get the red line for all odd number of points, the numbers in the magenta line all have 1 as the highest power of 2, and for the green line n has the form $5 \times 2^k \ldots$

In 4 dimensions



The structure gets distorted in higher dimensions, but the same observations hold!
Again the green line has n of the form 5×2^{k}.

In 5 dimensions



$\longrightarrow$ Lattice sequence and Sobol' are comparable, but using the standard error we can stop at good points and get better performance with periodization than without (only works for lattices).

To periodize or not to periodize?

- For low dimensions (≤ 5) it might be worthwhile trying periodization.
  (Low dimensions because periodization worsens the function and in higher dimensions we need more precision for the "weights".)
- Using an estimate for the error one can stop at the good points (even for this blind usage).
- The convergence graphs have structure: it might be possible to do better.

Conclusion

- In this talk we presented an overview on *fast component-by-component construction*.
- We showed that *lattice sequences* are a very interesting technology.
- Moreover, using lattice sequences is easy and fast.
- We showed lattice sequences perform well in applications.
- *Go use lattice sequences!*
- For software look at
  `http://www.cs.kuleuven.be/~dirkn/`.