

SPARSE POLYNOMIAL APPROXIMATIONS OF THE POISSON EQUATION OVER HYPERCUBES

CHRISTOPHE DE LUIGI, SERGE DUMONT, AND SYLVAIN MAIRE

ABSTRACT. We describe two ways to use reduced size polynomial approximations for the numerical solution of the Poisson equation over hypercubes. The first one is a sequential Monte Carlo algorithm. The second one is a non-standard Galerkin method which allows tests functions which do not verify the boundary conditions. Numerical examples are given in dimensions up to 5 in order to compare the two methods.

1. INTRODUCTION

Spectral methods [2,3] have been developed for a wide range of partial differential equations. They rely on tensor product approximations on several polynomial bases. They are especially efficient for smooth solutions on simple geometries. In the case of the Poisson equation in dimension Q over an hypercube $D = [-1, 1]^Q$, the most commonly used bases are the Legendre and the Tchebychef polynomial ones. Even in such favourable situations, the complexity of the spectral methods increases quickly with the dimension Q as the number of unknowns using the collocation method with tensor product approximations of degree N becomes $(N + 1)^Q$. Moreover one has to solve a plain linear symmetric system with a complexity of a $O((N + 1)^{2Q})$ using for example an iterative method like the conjuguate gradient method with or without preconditioning.

Hence it is worth considering approximations on different kinds of reduced size bases to solve these equations in order to attenuate the dimensional effect. We can for example mention the work from Von Petersdorf and Schwab [22] where sparse piecewise approximations are used to solve parabolic equations in high dimensions. Concerning the approximations of periodic smooth functions on Q -dimensional Fourier bases, Korobov spaces [12] have been introduced. They rely on a decay of the Fourier coefficients a_m as

$$|a_m| \leq \frac{C}{(\widetilde{m}_1 \widetilde{m}_2 \dots \widetilde{m}_Q)^\alpha}$$

where $\widetilde{m} = \max(1, |m|)$ and $\alpha > 1$ is linked to the smoothness of the integrand. The natural choice is to keep only the coefficients belonging to the set

$$\{m \in \mathbb{Z}^Q / (\widetilde{m}_1 \dots \widetilde{m}_Q) \leq d\}$$

where d is the level of approximations. Unfortunately one has to transform the original integrand using the periodisation method to achieve such a decay for non-periodic functions. This periodisation has a very bad effect on the constant C which grows very quickly with α . Another difficulty is the numerical computation of the coefficients which is handled using lattice rules in the purpose of numerical integration [11,23]. In the case of polynomial approximations, Novak and Ritter

[20] have developed a method to integrate polynomials such that their total degree is below a given value. Their method performs very well for dimensions $Q \geq 8$ and the integrand does not need to be periodic.

We have introduced in [15] polynomial spaces very similar to Korobov ones in a sense that the basis functions are chosen in a multidimensional Tchebychef polynomial basis according to a criterion based on the product of the degree of the polynomials of each variable. This basis has the same good properties than the previous Fourier basis but avoids the periodisation problems. Our criterion is also more selective compared to the one developed in Novak and Ritter and is directly linked to the regularity of the function to approximate. To compute the coefficients of an approximation on this basis, we have first used a sequential Monte Carlo algorithm [15] which was modified and improved by using quasi-Monte Carlo sequences [16]. The use of Tchebychef polynomials combined with random drawings associated to the Tchebychef weight were crucial in these works. We have finally replaced the algorithm by the least square problem of fitting our Tchebychef polynomial approximation model to some random, quasi-random points or quantified points [17]. This has led to quadrature formulas which will be used here to compute the approximations. In section 2, we make a better description of this method and give some numerical examples on various functions up to dimension 5 which will be the solutions of the Poisson equations of sections 3 and 4.

We describe in these two sections two numerical methods to use our basis for the numerical solution of the Poisson equation over an hypercube. The first one is a sequential Monte Carlo algorithm which has been introduced in [7] in the case of the Poisson equation. Its convergence properties and also its generalization to a large class of linear partial differential equations have been studied in [8]. The second one is a variational formulation introduced in [4] which allows test functions not verifying the boundary conditions. The main advantage of the method is that it reduces drastically the sizes of the matrices involved in the discretization of the variational formulation. We make some numerical comparisons between the two methods and we also compare approximations on our reduced basis to the standard approximations on tensor product Tchebychef polynomials.

2. THE APPROXIMATION METHOD

2.1. Description of the approximation. The Tchebychef polynomials $T_n(x) = \cos(n \arccos(x))$ are the orthogonal polynomials with respect to the inner product $\langle P, Q \rangle = \int_{-1}^1 \frac{P(x)Q(x)}{\sqrt{1-x^2}} dx$. They verify the differential equation

$$\frac{d}{dx}(\sqrt{1-x^2}T_n'(x)) + n^2 \frac{T_n(x)}{\sqrt{1-x^2}} = 0.$$

Using this equation, one can show that if $f \in C^{2L}([-1, 1])$ the coefficients b_n of its mean-square approximation on the Tchebychef polynomials verify $|b_n| \leq \frac{C}{n^{2L}}$, where C is a constant depending on f and L . The multidimensional interpolation polynomial $P_N(f)$ at the points $y_i = \cos(\frac{2i+1}{N+1} \frac{\pi}{2})$ of the Tchebychef grid writes

$$P_N(f) = \sum_{i_1=0}^N \sum_{i_2=0}^N \dots \sum_{i_Q=0}^N \alpha_{i_1, i_2, \dots, i_Q} T_{i_1}(x_1) T_{i_2}(x_2) \dots T_{i_Q}(x_Q)$$

where the $\alpha_{i_1, i_2, \dots, i_Q}$ are defined by

$$\alpha_{i_1, i_2, \dots, i_Q} = \frac{\pi^Q}{\prod_{j=1}^Q \|T_{i_j}\|_2^2 (N+1)^Q} \sum_{j_1=0}^N \dots \sum_{j_Q=0}^N f(y_{j_1}, \dots, y_{j_Q}) T_{i_1}(y_{j_1}) \dots T_{i_Q}(y_{j_Q}).$$

Furthermore, standard approximation results [2] show that $\|f - P_N(f)\|_2 \leq \frac{C}{N^{2L}}$ meaning that this approximation is really accurate especially when f is very smooth. However, this kind of approximation is very sensitive to the dimensional effect as its complexity is a $O(N^Q)$. Hence when the dimension Q increases one needs to consider other types of polynomial approximation which can also take advantage of the smoothness of the function f but with a smaller complexity. Letting $\widehat{m} = \max(1, m)$, we have proved in [13] that the coefficients b_m of the mean-square approximation in dimension Q verify

$$|b_m| \leq \frac{C_1}{(\widehat{m}_1 \widehat{m}_2 \dots \widehat{m}_Q)^{2L}}$$

still using the differential equation satisfied by the T_n for the Q integration variables. We can then give the approximation

$$f(x_1, x_2, \dots, x_Q) = \sum_{m \in W_{Q,d}} b_m T_{m_1}(x_1) T_{m_2}(x_2) \dots T_{m_Q}(x_Q) + r(t)$$

where the set

$$W_{Q,d} = \{m \in \mathbb{N}^Q / (\widehat{m}_1 \dots \widehat{m}_Q) \leq d\}$$

corresponds to a level d of approximation. We give in the following table some values of $L_{Q,d} = \text{card}(W_{Q,d})$ to give an idea of the complexity of the approximation.

d	$L_{2,d}$	$l_{3,d}$	$L_{4,d}$	$L_{5,d}$	$L_{6,d}$
1	4	8	16	32	64
2	8	20	48	112	256
3	12	32	80	192	448
5	21	62	168	432	1072
7	31	98	280	752	1936
10	48	165	504	1432	3872
15	76	276	880	2592	7232

Some theoretical results are described in [15] which can be summarized by the control on $\int_D r(t)^2 dt$. Under the previous assumptions, $\forall \varepsilon > 0$ there is a constant $C_{Q,\varepsilon}$ depending only on Q and ε such that

$$\|r\|_2 \leq \frac{C_{Q,\varepsilon}}{d^{2L-0.5-\varepsilon}}.$$

The approximation of the function f writes

$$f(x_1, x_2, \dots, x_Q) \simeq \sum_{m \in W_{Q,d}} \widetilde{b}_m T_{m_1}(x_1) T_{m_2}(x_2) \dots T_{m_Q}(x_Q).$$

We compute the $L_{Q,d}$ coefficients b_k belonging to $W_{Q,d}$ using a program which tests if $m_1 m_2 \dots m_Q \leq d$ and stores the values of these parameters in Q lists $l_1(k), l_2(k) \dots l_Q(k)$. We also define $c_1(k) = 1_{l_1(k) \geq 1}, \dots, c_Q(k) = 1_{l_Q(k) \geq 1}$ which occur in the following normalizations. As we create this lists, we also store in

another vector l the unique value of k corresponding to $l_1(k), l_2(k) \dots l_Q(k)$. This vector will be useful in section 3 when we need to compute second derivatives of these approximations. We now write

$$f(x_1, x_2, \dots, x_Q) \simeq \sum_{k=1}^{L_{Q,d}} \tilde{b}_k T_{l_1(k)}(x_1) T_{l_2(k)}(x_2) \dots T_{l_Q(k)}(x_Q).$$

2.2. The least-square problem. We describe quickly the least-square method developed in [17] to compute the coefficients b_k . For the sake of normalization, we use a least-square problem weighted by the norms of the basis functions

$$J_1 = \frac{1}{M} \sum_{i=1}^M \left(\sum_{k=1}^{L_{Q,d}} \tilde{s}_k \sqrt{2}^{\sum_{n=1}^Q c_n(k)} \prod_{n=1}^Q T_{l_n(k)}(X_n^{(i)}) - f(X_1^{(i)}, \dots, X_Q^{(i)}) \right)^2$$

with

$$\tilde{s}_k = \frac{\tilde{b}_k}{\sqrt{2}^{\sum_{n=1}^Q c_n(k)}}.$$

The minimization of J_1 is equivalent to the resolution of the system $B\tilde{s} = q$ with

$$B_{k,j} = \frac{\sqrt{2}^{\sum_{n=1}^Q c_n(k)+c_n(j)}}{M} \sum_{i=1}^M \prod_{n=1}^Q T_{l_n(k)}(X_n^{(i)}) T_{l_n(j)}(X_n^{(i)})$$

and

$$q_k = \frac{\sqrt{2}^{\sum_{n=1}^Q c_n(k)}}{M} \sum_{i=1}^M \prod_{n=1}^Q T_{l_n(k)}(X_n^{(i)}) f(X_1^{(i)}, \dots, X_Q^{(i)}).$$

The accuracy of the approximation depends on the condition number of the least square matrix B using the inequality $\|s - \tilde{s}\| \leq \|B\| \|B^{-1}\| \frac{\|r\|}{\|q\|} \|s\|$. The relative error in quadratic norm is

$$\frac{\|s - \tilde{s}\|_2}{\|s\|_2} \leq \|B\|_2 \|B^{-1}\|_2 \frac{\sqrt{C_{Q,\varepsilon}}}{\|q\|_2 d^{2L-0.5-\varepsilon}}.$$

If the data are random variables with density

$$w(x) = \prod_{i=1}^Q \frac{1}{\pi \sqrt{1-x_i^2}} 1_{[-1,1]}(x_i)$$

then the coefficients $B_{k,j}$ go to δ_{kj} with M because these coefficients are integrals computed by means of a Monte Carlo method. The speed of convergence toward the identity matrix is bounded by $\frac{C}{\sqrt{N}}$. The use of Tchebychef polynomials enables an uniform bound of this speed independent of d and Q that is $C \leq 1$. This crucial property is a straightforward consequence of the fact that these polynomials are uniformly bounded by 1. It has been observed for example in [15] that the constant C increases very quickly with d and Q when using Legendre polynomial basis. As mentioned in the introduction, it can be efficient to replace this Monte Carlo approximation by an approximation using Quasi-Monte Carlo sequences [12,18] as their rate of convergence for numerical integration is a $O(\frac{\ln(N)}{N}^{Q-1})$. Another similar point of view is to find the best way to represent with M points the density $w(x)$

according to some criterion. Using the competitive learning vector quantization algorithm [21], we have computed the M points in D minimizing the functional

$$J(M) = \min\left(\int_D \inf_{1 \leq i \leq M} |x - x_i|^2 w(x) dx : \{x_1, x_2, \dots, x_M \in D\}\right).$$

We have made some tests on pseudo-random linear generator, Halton sequences, Sobol sequences and points built from optimal quadratic quantization. On some basic examples in dimension 3, the quantization points appeared to be the most efficient just before the Halton sequences. However, these points are difficult to build in practice and one can also choose an hybrid strategy to build the quadrature points: make some steps of the competitive learning vector quantization algorithm initialized by Halton sequences.

2.3. Numerical integration and approximation. The first possibility to compute the coefficients \tilde{b}_k of the approximation

$$f(x_1, x_2, \dots, x_Q) \simeq \sum_{k=1}^{L_{Q,d}} \tilde{b}_k T_{l_1(k)}(x_1) T_{l_2(k)}(x_2) \dots T_{l_Q(k)}(x_Q)$$

is to solve the linear system $B\tilde{s} = q$ which has to be done for each different function. A cheaper way to do it is to store the Cholesky factorization of the matrix B once and for all. An even more efficient way is to build quadrature formulas for the numerical approximation of all the coefficients \tilde{b}_k and as for the numerical approximation $\tilde{I}(f)$ of

$$I(f) = \int_{[-1,1]^Q} f(x) dx.$$

We first compute numerically the inverse matrix B^{-1} and we then write $s = B^{-1}q$ to obtain

$$\tilde{s}_k = \sum_{j=1}^{L_{Q,d}} B_{kj}^{-1} q_j = \sum_{j=1}^{L_{Q,d}} B_{kj}^{-1} \frac{\sqrt{2}^{\sum_{n=1}^Q c_n(j)}}{M} \sum_{i=1}^M \prod_{n=1}^Q T_{l_n(k)}(X_n^{(i)}) f(X_1^{(i)}, \dots, X_Q^{(i)})$$

that is

$$\tilde{s}_k = \sum_{i=1}^M \left(\sum_{j=1}^{L_{Q,d}} B_{jk}^{-1} \frac{\sqrt{2}^{\sum_{n=1}^Q c_n(j)}}{M} \prod_{n=1}^Q T_{l_n(k)}(X_n^{(i)}) f(X_1^{(i)}, \dots, X_Q^{(i)}) \right).$$

Then, we have

$$\tilde{b}_k = \sum_{i=1}^M \lambda_{i,k} f(X_1^{(i)}, \dots, X_Q^{(i)})$$

with

$$\lambda_{i,k} = \sqrt{2}^{\sum_{n=1}^Q c_n(k)} \sum_{j=1}^{L_{Q,d}} B_{jk}^{-1} \frac{\sqrt{2}^{\sum_{n=1}^Q c_n(j)}}{M} \prod_{n=1}^Q T_{l_n(k)}(X_n^{(i)}).$$

If we are interested in numerical integration, we finally have

$$\tilde{I}(f) = \sum_{k=1}^{L_{Q,d}} \tilde{b}_k \prod_{n=1}^Q \int_{-1}^1 T_{l_n(k)}(x) dx = \sum_{i=1}^M \alpha_i f(X_1^{(i)}, \dots, X_Q^{(i)})$$

with

$$\alpha_i = \sum_{k=1}^{L_{Q,d}} \lambda_{i,k} \prod_{n=1}^Q \int_{-1}^1 T_{l_n(k)}(x) dx.$$

We have observed no significant difference between the two numerical integration procedures on some numerical tests. Hence we use the quadrature formulas for the computations of all the coefficients \tilde{b}_k of the approximation and for $\tilde{I}(f)$. This means that the coefficients $\lambda_{i,k}$ and α_i are computed and stored once and for all.

2.4. Numerical results. In this section, we give some numerical examples of approximations of functions in dimension 3 to 5 which will be the analytical solutions of the Poisson equations studied in section 3 and 4. This will show the accuracy of our approximation method and will also allow to check the efficiency of the two methods of resolution of the partial differential equations. As a first example, we use the functions

$$f_1(x) = \exp\left(\frac{\sum_{i=1}^Q x_i}{Q}\right)$$

which are obviously very smooth. As a second example, we build less regular functions having an exact degree 2 of smoothness. In dimension one, we use the cubic spline approximation of the function $\cos(\frac{x}{2})$ on $[-1, 1]$ at 7 equidistant points. In dimension Q , the functions f_2 are built using tensor products of the previous function. We now give some numerical examples from dimension 3 to 5 letting respectively the absolute errors $e_h(\frac{1}{2})$, $e_h(0)$, $e_h(I)$ for a function h at the reference points $(\frac{1}{2}, \dots, \frac{1}{2})$, $(0, \dots, 0)$ and on the integral over the domain $[-1, 1]^Q$. The $L_{Q,d}$ coefficients are computed using $[2.5 \times L_{3,d}]$ points built from Halton sequences.

d	$L_{3,d}$	$e_{f_1}(\frac{1}{2})$	$e_{f_1}(0)$	$e_{f_1}(I)$	$e_{f_2}(\frac{1}{2})$	$e_{f_2}(0)$	$e_{f_2}(I)$
3	32	2×10^{-4}	2×10^{-3}	1×10^{-3}	9×10^{-4}	1×10^{-2}	7×10^{-4}
5	62	8×10^{-5}	1×10^{-5}	3×10^{-5}	8×10^{-4}	2×10^{-5}	8×10^{-5}
7	98	1×10^{-5}	3×10^{-5}	5×10^{-6}	1×10^{-4}	9×10^{-4}	1×10^{-5}
10	165	2×10^{-7}	4×10^{-7}	1×10^{-8}	1×10^{-4}	6×10^{-5}	2×10^{-6}
15	276	2×10^{-7}	6×10^{-8}	2×10^{-8}	3×10^{-5}	2×10^{-5}	4×10^{-6}
30	700	1×10^{-10}	3×10^{-10}	4×10^{-11}	4×10^{-6}	6×10^{-7}	7×10^{-8}
60	1702	6×10^{-14}	2×10^{-13}	4×10^{-14}	6×10^{-7}	3×10^{-8}	3×10^{-9}

The approximations of both functions are really accurate. We achieve for example an accuracy of 10 digits on the approximation of f_1 and 6 digits on the approximation of f_2 when $d = 30$. The corresponding number of basis functions is 700. As a comparison, we also give the same kind of results using the approximation on the tensor product Tchebychef interpolation polynomials of degree N .

N	$(N + 1)^3$	$e_{f_1}(\frac{1}{2})$	$e_{f_1}(0)$	$e_{f_1}(I)$	$e_{f_2}(\frac{1}{2})$	$e_{f_2}(0)$	$e_{f_2}(I)$
3	64	1×10^{-4}	9×10^{-4}	5×10^{-4}	4×10^{-3}	3×10^{-2}	2×10^{-2}
5	216	3×10^{-7}	3×10^{-7}	2×10^{-7}	1×10^{-3}	2×10^{-3}	6×10^{-4}
7	512	1×10^{-10}	2×10^{-10}	1×10^{-10}	1×10^{-3}	1×10^{-3}	3×10^{-4}
10	1331	3×10^{-14}	5×10^{-14}	2×10^{-14}	5×10^{-4}	3×10^{-4}	7×10^{-5}

We observe that the approximation results are slightly more accurate for the function f_1 but really less accurate for the function f_2 . For example, when $N = 7$

which corresponds to 512 basis functions, we achieve an accuracy of 10 digits on the approximation of f_1 but only 3 digits on the approximation of f_2 . This means that the reduced basis is less sensitive to the smoothness of the functions than the usual tensor product Tchebychef interpolation. This has been already observed in [15] and as our basis is also a lot less sensitive to the dimensional effect, we only keep it for higher dimensions. We now turn out to dimension 4.

d	$L_{4,d}$	$e_{f_1}(\frac{1}{2})$	$e_{f_1}(0)$	$e_{f_1}(I)$	$e_{f_2}(\frac{1}{2})$	$e_{f_2}(0)$	$e_{f_2}(I)$
3	80	4×10^{-4}	1×10^{-3}	2×10^{-4}	1×10^{-2}	2×10^{-2}	1×10^{-3}
5	168	8×10^{-6}	4×10^{-6}	5×10^{-6}	8×10^{-4}	8×10^{-4}	6×10^{-4}
7	280	2×10^{-6}	1×10^{-5}	5×10^{-6}	1×10^{-3}	8×10^{-4}	3×10^{-4}
10	504	1×10^{-7}	1×10^{-7}	2×10^{-7}	2×10^{-4}	6×10^{-5}	2×10^{-5}
15	880	3×10^{-8}	4×10^{-8}	1×10^{-9}	5×10^{-5}	6×10^{-5}	1×10^{-5}
30	2453	2×10^{-10}	3×10^{-10}	1×10^{-12}	6×10^{-6}	3×10^{-6}	4×10^{-7}

Once again, we obtain a good accuracy on the approximations. The number of basis functions $L_{4,d}$ is about 3 times greater than $L_{3,d}$. Finally in dimension 5, we have the following table.

d	$L_{5,d}$	$e_{f_1}(\frac{1}{2})$	$e_{f_1}(0)$	$e_{f_1}(I)$	$e_{f_2}(\frac{1}{2})$	$e_{f_2}(0)$	$e_{f_2}(I)$
2	112	7×10^{-3}	4×10^{-3}	1×10^{-3}	2×10^{-2}	3×10^{-2}	4×10^{-2}
3	192	9×10^{-5}	4×10^{-4}	8×10^{-4}	5×10^{-3}	3×10^{-2}	1×10^{-2}
5	432	2×10^{-5}	9×10^{-6}	8×10^{-6}	5×10^{-3}	2×10^{-3}	2×10^{-3}
7	752	1×10^{-6}	5×10^{-6}	6×10^{-6}	2×10^{-3}	3×10^{-3}	7×10^{-4}
10	1432	6×10^{-8}	4×10^{-8}	5×10^{-9}	3×10^{-4}	4×10^{-5}	8×10^{-5}
15	2592	5×10^{-8}	1×10^{-8}	4×10^{-9}	7×10^{-5}	2×10^{-4}	2×10^{-5}

We still obtain a good accuracy on the approximations. The complexity of the approximation $L_{5,d}$ is about 3 times greater than $L_{4,d}$.

3. SEQUENTIAL MONTE CARLO APPROXIMATION

3.1. General description.

3.1.1. *Feynman-Kac representation.* We make a short description of the sequential control variates method developed in previous works focusing on the Poisson equation. The Feynman-Kac representation [5] of the pointwise solution of the Poisson equation

$$-\frac{1}{2}\Delta u = f$$

in a domain $D \subset \mathbb{R}^d$ with boundary Dirichlet conditions $u = g$ on ∂D is given by

$$u(x) = E_x(g(B_{\tau_D}) + \int_0^{\tau_D} f(B_s) ds)$$

where B_s is a d -dimensional Brownian motion and τ_D is the exit time of this process from the domain D . The Monte Carlo computation of this pointwise solution [14] usually leads to two kinds of numerical errors. The first one comes from the simulation error of the Brownian motion. If we call Δt the discretization step, the relative error on the approximate solution using the Euler scheme is a $O(\sqrt{\Delta t})$

which can be reduced to a $O(\Delta t)$ when using a more sophisticated approach to deal with the boundary conditions [6]. Some other schemes are also available like for example the modified walk on spheres method [10] which is really efficient in the case of constant diffusions. The second kind of error is the standard Monte Carlo error using M simulations that is $\frac{\sigma}{\sqrt{M}}$ where σ is the variance of

$$g(X_{\tau_D}^x) + \int_0^{\tau_D} f(X_s^x) ds$$

or of its discretized version. These errors are quite different from the usual approximation errors involved in the deterministic numerical methods. We now describe the sequential Monte Carlo algorithm developed in [7] which error and behaviour are quite similar to the ones of a deterministic iterative method.

3.1.2. Description of the algorithm. We first compute approximate values $u_i^{(1)}$ of the solution of this equation at N points x_i using a Monte Carlo method. Using this information, we can then build a global and regular linear approximation $u^{(1)}(x)$ of the Poisson equation. This can be achieved for example by a simple polynomial interpolation or by fitting the data to a polynomial model. We now use the control variate method with $u^{(1)}$ as an approximation of u . We put $\forall x \in D$

$$y(x) = u(x) - u^{(1)}(x)$$

and we now have to solve $\forall x \in D$ the Poisson equation

$$-\frac{1}{2}\Delta y = -\frac{1}{2}\Delta(u - u^{(1)}) = f + \frac{1}{2}\Delta u^{(1)}$$

with boundary conditions

$$y = g - u^{(1)}.$$

A global solution $y^{(1)}$ is computed using the same method that we have used to compute $u^{(1)}$. Finally we approximate the solution of the initial equation by

$$u^{(2)}(x) = u^{(1)}(x) + y^{(1)}(x),$$

we can expect that this solution is more accurate than the previous one. We can iterate this method to achieve an approximation $u^{(n)}(x)$ at the n th step of the relevant algorithm.

3.1.3. Convergence and approximation properties. We now recall the main hypotheses and convergence results of the algorithm which are described with more details in [8]. We assume that the approximation of the solution u can be written in a linear form

$$Pu(x) = \sum_{j=1}^N u(x_j)\Psi_j(x)$$

for some functions $\Psi_j(x)$. In our particular case, this form can be obtained easily because the approximation Pu of u is built via a discrete least-square problem (see also [8]). Some weak assumptions are also required on the simulation scheme of the Brownian motion with a discretization parameter Δ . The algorithm is stochastic and biased due to the simulation scheme. Hence we define the quantities

$$m_n = \max_{1 \leq i \leq N} |E(u_n(x_i) - u(x_i))|, v_n = \max_{1 \leq i \leq N} Var(x_i)$$

to study its convergence. To study the influence of the simulation scheme, we consider the difference of the solution of the Poisson equations, with g as both

source term and boundary condition, between respectively the discretized and the continuous one. Then $e(g, \Delta, x)$ and $V(g, \Delta, x)$ are respectively the mean value and the variance of the previous quantity. We first state the convergence result for the bias.

Theorem 3.1. *For any $n \geq 1$, we have*

$$m_n \leq \rho_m m_{n-1} + \max_{1 \leq i \leq N} |[P(u) - u](x_i) + P[e(u - Pu, \Delta, \cdot)](x_i)|$$

where $\rho_m = \max_{1 \leq i \leq N} [\sum_{j=1}^N |P[e(\Psi_j(\cdot, \Delta, \cdot))](x_i)|]$. If Δ is small enough, then $\rho_m < 1$ and m_n converges at a geometric rate up to a threshold equal to

$$\limsup m_n \leq \frac{1}{1 - \rho_m} \max_{1 \leq i \leq N} |[P(u) - u](x_i) + P[e(u - Pu, \Delta, \cdot)](x_i)|.$$

This theorem shows that even if the simulations are biased the upper limit on the bias depends mainly on the quality of the approximation $P(u) - u$. We now state the convergence of the variance v_n .

Theorem 3.2. *Setting*

$$C(\Delta, N) = 2 \max_{1 \leq i \leq N} \sum_{j=1}^N \Psi_j^2(x_i) \left[\sum_{k=1}^N \sqrt{V(\Psi_k, \Delta, x_j)} \right]^2,$$

$$\rho_v = \max_{1 \leq i \leq N} \left(\sum_{j=1}^N |P[e(\Psi_j(\cdot, \Delta, \cdot))](x_i)| \right)^2 + \frac{C(\Delta, N)}{M},$$

then one has for any $n \geq 1$

$$v_n \leq \rho_v v_{n-1} + \frac{1}{M} 2 \max_{1 \leq i \leq N} \sum_{j=1}^N \Psi_j^2(x_i) V(u - Pu, \Delta, x_j) + C(\Delta, N) m_{n-1}.$$

If Δ is small enough and M large enough, then $\rho_v < 1$ and v_n converges at a geometric rate up to a threshold equal to

$$\limsup m_n \leq \left(\frac{1}{1 - \rho_v} M \right) \left(2 \max_{1 \leq i \leq N} \sum_{j=1}^N \Psi_j^2(x_i) V(u - Pu, \Delta, x_j) + C(\Delta, N) \limsup m_n^2 \right).$$

Note that when $\rho_v < 1$ and $\rho_m < 1$ (which is always true for Δ small enough and M large enough) the convergence holds for both the bias and the variance. We now describe the algorithm based on spectral approximations especially how the new source term is at each step of the algorithm and also what is the accuracy on the solution.

3.2. Spectral formulation and approximation. We now describe the algorithm based on our spectral approximations especially how the new source term is at each step of the algorithm and also what is the accuracy on the solution.

3.2.1. *New source term.* For the sake of simplicity, we only describe how to obtain the formulas for the new source term in dimension 2. The extension to higher dimensions is straightforward. We recall [3] that the expression of the second derivative of a polynomial approximation $P_N(u) = \sum_{n=0}^N \alpha_n T_n$ of a function u writes $(P_N(u))'' = \sum_{n=0}^{N-2} \beta_n T_n$ with

$$\beta_k = \frac{1}{c_k} \sum_{p=k+2, p+k \text{ even}}^N p(p^2 - k^2) \alpha_p$$

where the normalization coefficient c_k is such that $c_0 = 2$ and $c_k = 1$ if $k \neq 0$. Then if the bidimensional approximation writes

$$u_N = \sum_{n=0}^N \sum_{m=0}^N \alpha_{n,m} T_n T_m,$$

we have

$$\Delta u_N = \sum_{n=0}^N \sum_{m=0}^N \alpha_{n,m}^{(2)} T_n T_m$$

with

$$\alpha_{n,m}^{(2)} = \frac{1}{c_n} \sum_{p=n+2, p+n \text{ even}}^N p(p^2 - n^2) \alpha_{p,m} + \frac{1}{c_m} \sum_{p=m+2, p+m \text{ even}}^N p(p^2 - m^2) \alpha_{n,p}.$$

If we now consider the bidimensional approximation on the reduced basis

$$u_d = \sum_{k=1}^{L_{2,d}} b_k T_{l_1(k)} T_{l_2(k)},$$

we have

$$\Delta u_d = \sum_{k=1}^{L_{2,d}} b_k T_{l_1(k)}'' T_{l_2(k)} + \sum_{k=1}^{L_{2,d}} b_k T_{l_1(k)} T_{l_2(k)}''.$$

Then, we can write

$$\sum_{k=1}^{L_{2,d}} b_k T_{l_1(k)}'' T_{l_2(k)} = \sum_{k=1}^{L_{2,d}} \sum_{m=0}^{l_1(k)-2} \frac{b_k}{c_m} l_1(k) (l_1(k)^2 - m^2) 1_{l_1(k)+m \text{ even}} T_m T_{l_2(k)}$$

and also

$$\sum_{k=1}^{L_{2,d}} b_k T_{l_1(k)} T_{l_2(k)}'' = \sum_{k=1}^{L_{2,d}} \sum_{m=0}^{l_2(k)-2} \frac{b_k}{c_m} l_2(k) (l_2(k)^2 - m^2) 1_{l_2(k)+m \text{ even}} T_{l_1(k)} T_m.$$

The function Δu_d can be obviously approximated on the same space than u_d . To compute the coefficients $b_i^{(2)}$ of its approximation

$$\Delta u_d = \sum_{i=1}^{L_{2,d}} b_i^{(2)} T_{l_1(i)} T_{l_2(i)}$$

on this space, we need to know to which two basis functions the terms $T_m T_{l_2(k)}$ and $T_{l_1(k)} T_m$ correspond to. This can be done easily by using a vector l , created along with the lists l_1 and l_2 , which tells which indices i_1 and i_2 correspond to the pairs $(m, l_2(k))$ and $(l_1(k), m)$, that is $l(m, l_2(k)) = i_1$ and $l(l_1(k), m) = i_2$.

3.2.2. *Approximations results.* The quality of the solution depends mainly on the approximation $u - Pu$. We have seen described in section 2 the two kinds of approximation results if we either consider the tensor product approximation or the approximation on the reduced size basis. We are likely to obtain numerical approximations of the solutions which are of the same magnitude than these approximations.

3.3. **Numerical results.** We only take in this section the first example of section 2, that is we solve Poisson equations in dimensions 3 to 5 which solutions are the functions $f_1(x) = \exp(\frac{\sum_{i=1}^Q x_i}{Q})$ (the source terms of these Poisson equations are $-\frac{1}{2}\Delta f_1$ and the boundary conditions are f_1). The simulation scheme for the Brownian motion is the Euler scheme with a test near the boundary based on the half-space approximation [6] with M trajectories and a step size Δt . We do not test less smooth solutions here because we remark on the numerical experiments that we already have some difficulties using this method with very smooth solutions. We denote by S the number of steps until convergence and we give CPU times in seconds. In dimension 3, we have the following table.

d	$L_{3,d}$	M	Δt	$e_{f_1}(\frac{1}{2})$	$e_{f_1}(0)$	CPU	S
3	32	100	0.04	4×10^{-4}	2×10^{-3}	0.5	8
5	62	180	0.02	1×10^{-4}	1×10^{-5}	6.2	15
7	98	300	0.02	1×10^{-4}	2×10^{-5}	22	14
10	165	800	0.005	5×10^{-6}	6×10^{-6}	403	12
12	216	1200	0.003	7×10^{-6}	2×10^{-6}	2114	16

On this example and on the next ones, we have chosen the step size and the number of Brownian trajectories so that the convergence of the algorithm is always achieved. To do so, we have to increase the number of trajectories and also to diminish the step size of the Euler scheme when d increases. We notice that we obtain an accuracy on the solutions at the two reference points which is comparable to the approximations of the exact solution at the same points. Nevertheless, the accuracy obtained on the solutions is one digit less accurate than the accuracy on the corresponding approximations of the exact solutions for $d = 10$. This value seems also to be the limit for the method to be acceptable in terms of CPU times. We now compare with the tensor product approximation on Tchebychef polynomials.

N	$(N+1)^3$	M	Δt	$e_{f_1}(\frac{1}{2})$	$e_{f_1}(0)$	CPU	S
3	64	100	0.03	1×10^{-4}	2×10^{-4}	1.5	10
5	216	200	0.01	2×10^{-7}	2×10^{-7}	23.5	14
7	512	800	0.002	3×10^{-8}	2×10^{-8}	2800	16

The results are once again comparable to the exact solutions except maybe for $N = 7$ where the accuracy was 10 digits instead of 8 here. We could certainly reach this accuracy by diminishing Δt and increasing M but this leads to greater CPU times which were anyhow too large in this case. As the solution is very smooth, the solution is well enough accurate taking $N = 5$ which requires CPU times equal

to 23.5 seconds. The following table summarizes the results obtained in dimension 4 to 5.

d	$L_{4,d}$	M	Δt	$e_{f_1}(\frac{1}{2})$	$e_{f_1}(0)$	CPU	S
3	80	100	0.02	2×10^{-3}	1×10^{-3}	2.8	11
5	168	300	0.02	5×10^{-5}	6×10^{-6}	37	14
7	280	600	0.01	2×10^{-5}	1×10^{-5}	350	16
10	504	1000	0.005	8×10^{-6}	4×10^{-6}	3200	15
d	$L_{5,d}$	M	Δt	$e_{f_1}(\frac{1}{2})$	$e_{f_1}(0)$	CPU	S
2	112	100	0.03	2×10^{-3}	7×10^{-4}	3.3	11
3	192	200	0.02	2×10^{-4}	4×10^{-4}	21	12
5	432	400	0.02	2×10^{-5}	6×10^{-6}	200	13
7	752	700	0.01	2×10^{-5}	7×10^{-6}	1600	12

We obtain the same kind of results than in dimension 3 concerning the accuracy on the solutions. The values $d = 7$ in dimension 4 and $d = 5$ in dimension 5 seem to be the upper limits for using this method at a reasonable cost. As the solutions are very smooth, these values are nevertheless sufficient to obtain an accuracy of 5 digits on these solutions.

3.4. Conclusions. On all the numerical examples studied, we have obtained a good accuracy on the solutions of the Poisson equations. However, the algorithm is not efficient enough especially when d increases. As a consequence we cannot expect to use it for less smooth solutions in its present form. Some more sophisticated numerical schemes need to be developed to improve this efficiency using for example the ideas developed in [10]. There is not a good balance between the evaluation of the source terms $E_x(\int_0^{\tau_D} f(B_s)ds)$ and the boundary terms $E_x(g(B_{\tau_D}))$. Indeed in a Monte Carlo simulation starting at x , many evaluations of the function f are done instead of only one of the function g . One can also think of using quasi-Monte Carlo simulations [14] or make some parallel versions of the algorithm. Finally, one can use the control variates method after convergence of the algorithm at a fixed point to obtain an even more accurate approximation.

4. THE HYBRID GALERKIN FORMULATION

4.1. Introduction. The basis functions that we have used in our approximations do not verify automatically the boundary conditions. In order to use them as test functions anyway, we adopt here an hybrid variational formulation which was proposed in [4] following the ideas of the Nitsche method [1,19,24]. This hybrid variational formulation for the Poisson equation

$$-\Delta u = f$$

in a domain Ω with boundary conditions $u = u_0$ on Γ_D , $\frac{\partial u}{\partial n} = g$ on Γ_N writes

$$\int_{\Omega} \nabla u \cdot \nabla v dx - \int_{\Gamma_D} (\frac{\partial u}{\partial n} v + \frac{\partial v}{\partial n} u) ds = \int_{\Omega} f v dx + \int_{\Gamma_N} g v ds - \int_{\Gamma_D} u_0 \frac{\partial v}{\partial n} ds$$

where v is a test function. In the original method, a penalization term of the form $r \int_{\Gamma_D} u v ds$ is added to the left handside of the variational formulation in order to enforce the coercivity of the symmetric bilinear form. It is shown in [4] that

in the case of elliptic problems, it is not necessary to add this penalization term to obtain the uniqueness and the convergence of the solution of the discretized problem. However, the relative matrix is no longer positive but is still invertible. We can point out that for test functions vanishing on the boundary, this formulation is exactly the same as the classical one. This method has already been used with various test functions like finite elements which do not respect the shape of the boundary or with wavelets. Moreover, there is no loss of accuracy of using this method than the classical one at least in the one dimensional case.

The test functions used here are product of Tchebychev polynomials of each variables. We solve the Poisson equation using different kinds of approximations based on these test functions. We first describe the variational formulations based on these approximations. Then we make some numerical tests to compare with the previous Monte Carlo method and also to study the impact of less smooth solutions on this method. Finally, we introduce additional approximation spaces to overcome part of the bad conditioning problems which appear when the size of the approximation space increases.

4.2. Description of the variational formulations. We first make in detail the description of this formulation using different polynomial basis in the case of Dirichlet homogeneous boundary conditions on cubes that is $u_0 = 0$ and $\Gamma_D = \Gamma_\Omega$ on $D = [-1, 1]^Q$ for $Q = 1, 2, 3$. Then, we describe shortly how to extend this method to higher dimensions and to more general boundary conditions.

4.2.1. *The one-dimensional case.* In dimension one, the approximation of the solution writes

$$u_N(x) = \sum_{k=0}^N \alpha_k T_k(x)$$

where the coefficients α_k are solutions of the $N + 1$ equations

$$\sum_{k=0}^N \alpha_k (\theta_{k,j} + \gamma_{k,j}) = \beta_j$$

with

$$\beta_j = \int_{-1}^1 f(x) T_j(x) dx, \quad b_{k,j} = \int_{-1}^1 T_k'(x) T_j'(x) dx$$

and

$$\gamma_{k,j} = T_k(-1) T_j'(-1) + T_j(-1) T_k'(-1) - T_k(1) T_j'(1) - T_j(1) T_k'(1).$$

As $T_j(1) = 1, T_j(-1) = (-1)^j, T_j'(1) = j^2, T_j'(-1) = (-1)^{j+1} j^2$, we have

$$\gamma_{k,j} = ((-1)^{k+j+1} - 1)(j^2 + k^2).$$

Letting $a_{k,j} = b_{k,j} + \gamma_{k,j}$, we have to solve the linear system $A\alpha = \beta$. As the matrix is not positive and definite, a LU factorisation appears as a good choice for the resolution. The coefficients $b_{k,j}$ can be computed exactly and stored once and for all. The approximations of the coefficients β_j are computed using quadrature formulas described in section 2. More precisely, we first obtain an approximation of the function of the form

$$f(x) \simeq \sum_{k=0}^N c_k T_k(x)$$

where the c_k are obtained via quadrature formulas. These quadratures can either be our quadratures or the usual quadratures based on Gauss points. Then we have

$$\beta_j \simeq \int_{-1}^1 \sum_{k=0}^N c_k T_k(x) T_j(x) dx \simeq \sum_{k=0}^N c_k s_{k,j}$$

with

$$s_{k,j} = \int_{-1}^1 T_k(y) T_j(y) dy.$$

The coefficients $b_{k,j}$ and $s_{k,j}$ can be computed exactly and stored once and for all. They are also useful in higher dimensions.

4.2.2. The bidimensional case. We describe how to use the variational formulation on the reduced basis. The approximation of the solution writes

$$u_d(x, y) = \sum_{k=1}^{L_{2,d}} \alpha_k T_{l_1(k)}(x) T_{l_2(k)}(y)$$

where the two lists $l_1(k)$ and $l_2(k)$ are used to locate to which basis functions the $L_{2,d}$ coefficients α_k belonging to $W_{2,d}$ correspond to. These coefficients α_k are solutions of the $L_{2,d}$ equations

$$\sum_{k=1}^{L_{2,d}} \alpha_k (\theta_{k,j} + \gamma_{k,j}) = \beta_j$$

with

$$\beta_j = \int_{-1}^1 \int_{-1}^1 f(x, y) T_{l_1(j)}(x) T_{l_2(j)}(y) dx dy.$$

and

$$\theta_{k,j} = b_{l_1(k), l_1(j)} s_{l_2(k), l_2(j)} + s_{l_1(k), l_1(j)} b_{l_2(k), l_2(j)}.$$

We now compute the last term

$$\gamma_{k,j} = \int_{\Gamma_D} \frac{\partial T_{l_1(k)}(x) T_{l_2(k)}(y)}{\partial n} T_{l_1(j)}(x) T_{l_2(j)}(y) + \frac{\partial T_{l_1(j)}(x) T_{l_2(j)}(y)}{\partial n} T_{l_1(k)}(x) T_{l_2(k)}(y) ds$$

which is more complicated. We write $\Gamma_D = \cup_{i=1,4} \Gamma_i$ where the Γ_i are the 4 parts of the boundaries starting with $\Gamma_1 = [-1, 1] \times (-1)$ and so on. The integrals on the boundaries are respectively equal to

$$\gamma_{k,j}^{(1)} = -(T_{l_2(k)}(-1) T'_{l_2(j)}(-1) + T'_{l_2(k)}(-1) T_{l_2(j)}(-1)) \int_{-1}^1 T_{l_1(j)}(x) T_{l_1(k)}(x) dx$$

that is

$$\gamma_{k,j}^{(1)} = -s_{l_1(k), l_1(j)} (-1)^{l_2(k)+l_2(j)+1} (l_2(j)^2 + l_2(k)^2),$$

then

$$\gamma_{k,j}^{(2)} = s_{l_1(k), l_1(j)} (l_2(j)^2 + l_2(k)^2), \quad \gamma_{k,j}^{(3)} = s_{l_2(k), l_2(j)} (l_1(j)^2 + l_1(k)^2),$$

$$\gamma_{k,j}^{(4)} = -s_{l_2(k), l_2(j)} (-1)^{l_1(k)+l_1(j)+1} (l_1(j)^2 + l_1(k)^2)$$

and finally

$$\gamma_{k,j} = -\gamma_{k,j}^{(1)} + \gamma_{k,j}^{(2)} + \gamma_{k,j}^{(3)} - \gamma_{k,j}^{(4)}.$$

The coefficients β_j are computed using the quadrature formulas of section 2. The approximation of f is

$$f_d(x, y) = \sum_{k=1}^{L_{2,d}} c_k T_{l_1(k)}(x) T_{l_2(k)}(y)$$

and hence

$$\beta_j \simeq \sum_{k=1}^{L_{2,d}} c_k s_{l_1(k), l_1(j)} s_{l_2(k), l_2(j)}$$

that is

$$\beta_j \simeq \sum_{k=1}^{L_{2,d}} \sum_{i=1}^{[2.5 \times L_{2,d}]} \lambda_{i,k} f(X_1^{(i)}, X_2^{(i)}) s_{l_1(k), l_1(j)} s_{l_2(k), l_2(j)}$$

where the weights $\lambda_{i,k}$ and the points $(X_1^{(i)}, X_2^{(i)})$ have been defined in section 2. We can use the same methodology for the formulation based on tensor product approximations of degree N . The size of the approximation space is $(N+1)^2$ instead of $L_{2,d}$, two new lists are created to locate the basis functions corresponding to each coefficients of the approximation and the β_j are computed using Gauss-Tchebychev product rules.

4.2.3. The tridimensional case. We describe only the approximation of the solution on the sparse basis which writes

$$u_d(x, y) = \sum_{k=1}^{L_{3,d}} \alpha_k T_{l_1(k)}(x) T_{l_2(k)}(y) T_{l_3(k)}(z).$$

The coefficients α_k are solutions of the $L_{3,d}$ equations

$$\sum_{k=1}^{L_{3,d}} \alpha_k (\theta_{k,j} + \gamma_{k,j}) = \beta_j$$

with

$$\beta_j = \int_{-1}^1 \int_{-1}^1 \int_{-1}^1 f(x, y, z) T_{l_1(j)}(x) T_{l_2(j)}(y) T_{l_3(j)}(z) dx dy dz$$

and

$$\begin{aligned} \theta_{k,j} = & b_{l_1(k), l_1(j)} s_{l_2(k), l_2(j)} s_{l_3(k), l_3(j)} + b_{l_2(k), l_2(j)} s_{l_1(k), l_1(j)} s_{l_3(k), l_3(j)} + \\ & b_{l_3(k), l_3(j)} s_{l_1(k), l_1(j)} s_{l_2(k), l_2(j)}. \end{aligned}$$

The coefficient $\gamma_{k,j}$ is now a sum of 6 terms corresponding to each side of the cube $[-1, 1]^3$. We have

$$\begin{aligned} \gamma_{k,j}^{(1)} &= -s_{l_1(k), l_1(j)} s_{l_3(k), l_3(j)} (-1)^{l_2(k)+l_2(j)+1} (l_2(j)^2 + l_2(k)^2), \\ \gamma_{k,j}^{(2)} &= s_{l_1(k), l_1(j)} s_{l_3(k), l_3(j)} (l_2(j)^2 + l_2(k)^2), \\ \gamma_{k,j}^{(3)} &= s_{l_2(k), l_2(j)} s_{l_3(k), l_3(j)} (l_1(j)^2 + l_1(k)^2), \\ \gamma_{k,j}^{(4)} &= -s_{l_2(k), l_2(j)} s_{l_3(k), l_3(j)} (-1)^{l_1(k)+l_1(j)+1} (l_1(j)^2 + l_1(k)^2), \\ \gamma_{k,j}^{(5)} &= s_{l_1(k), l_1(j)} s_{l_2(k), l_2(j)} (l_3(j)^2 + l_3(k)^2), \\ \gamma_{k,j}^{(6)} &= -s_{l_1(k), l_1(j)} s_{l_2(k), l_2(j)} (-1)^{l_3(k)+l_3(j)+1} (l_3(j)^2 + l_3(k)^2) \end{aligned}$$

and finally

$$\gamma_{k,j} = -\gamma_{k,j}^{(1)} + \gamma_{k,j}^{(2)} + \gamma_{k,j}^{(3)} - \gamma_{k,j}^{(4)} + \gamma_{k,j}^{(5)} - \gamma_{k,j}^{(6)}.$$

The coefficients β_j are computed in the same way than in dimension 2.

4.2.4. *Extension to general problems.* The extension of the method to problems in dimension Q with Dirichlet homogeneous boundary conditions is easy. The coefficient $\theta_{k,j}$ is a sum of Q terms with first term equal to

$$b_{l_1(k), l_1(j)} \prod_{i=2}^Q s_{l_i(k), l_i(j)},$$

the coefficient $\gamma_{k,j}$ is a sum of $2Q$ terms with first 2 terms equal to

$$- \prod_{i \neq 2}^Q s_{l_i(k), l_i(j)} (-1)^{l_2(k) + l_2(j) + 1} (l_2(j)^2 + l_2(k)^2)$$

and

$$\prod_{i \neq 2}^Q s_{l_i(k), l_i(j)} (l_2(j)^2 + l_2(k)^2)$$

and the coefficients β_j are computed using quadrature formulas in dimension Q . In the case of more general boundary conditions, we also have to compute the terms

$$\int_{\Gamma_N} g v ds - \int_{\Gamma_D} u_0 \frac{\partial v}{\partial n} ds.$$

We assume for the sake of simplicity that Γ_N and Γ_D are constituted of faces of the hypercube $[-1, 1]^Q$. We use the same method than for the computation of the β_j . We compute approximations \tilde{g} and \tilde{u}_0 of the functions g and u_0 on spaces of size $L_{Q-1,d}$ and then we integrate exactly the products $\tilde{g}v$ or $\tilde{u}_0 \frac{\partial v}{\partial n}$. This last computation leads to integrate terms of the form

$$\int_{-1}^1 T_k(x) T_j'(x) dx$$

which are computed and stored once and for all.

4.3. **Numerical results.** We first study equations in dimension 3 with either a very smooth solution f_1 or a less smooth solution f_2 . In the following table, the solutions are computed at the two reference points and we denote by $\kappa(A)$ the condition number of the matrix A .

d	$L_{3,d}$	$e_{f_1}(\frac{1}{2})$	$e_{f_1}(0)$	$e_{f_2}(\frac{1}{2})$	$e_{f_2}(0)$	$\kappa(A)$
3	32	2×10^{-4}	1×10^{-3}	7×10^{-3}	3×10^{-2}	16
5	62	8×10^{-5}	1×10^{-5}	2×10^{-3}	2×10^{-4}	58
7	98	5×10^{-6}	2×10^{-5}	6×10^{-4}	1×10^{-3}	147
10	165	7×10^{-8}	1×10^{-7}	4×10^{-5}	3×10^{-5}	790
15	276	2×10^{-8}	3×10^{-8}	2×10^{-5}	7×10^{-5}	1111
30	700	4×10^{-10}	5×10^{-9}	1×10^{-4}	6×10^{-5}	2.8×10^7
60	1702	7×10^{-5}	2×10^{-4}	2×10^7	2×10^7	7.1×10^{32}

Until $d = 15$, the condition number $\kappa(A)$ is relatively small and the approximate solution is as accurate as the expansion of the exact solution on the same

approximation basis. When $d = 30$ and even more when $d = 60$, $\kappa(A)$ becomes too large which perturbs the solutions. It is well-known [2] that approximations on polynomials of high degree have a bad impact on $\kappa(A)$. In order to diminish $\kappa(A)$, we change the approximation spaces by using an additional test which keeps only basis functions of maximal degree of each monomial equal to 10. We denote by $L'_{3,d}$ the size of this space and we build quadrature formulas and approximations relative to this space using the least-square method of section 2. As a comparison, we now study the previous example for $d = 15, 30, 60$.

d	$L'_{3,d}$	$e_{f_1}(\frac{1}{2})$	$e_{f_1}(0)$	$e_{f_2}(\frac{1}{2})$	$e_{f_2}(0)$	$\kappa(A)$
15	216	2×10^{-8}	3×10^{-8}	3×10^{-5}	2×10^{-5}	409
30	400	4×10^{-10}	5×10^{-9}	2×10^{-4}	1×10^{-4}	3430
60	643	3×10^{-12}	1×10^{-11}	3×10^{-4}	7×10^{-4}	40015

The condition number has really decreased and is now only equal to 40015 when $d = 60$. This new criterion allows to take larger values of d without having bad conditioning problems especially for smooth solutions. Furthermore, the number of unknowns has also decreased significantly. We now look at tensor product approximations.

N	$(N+1)^3$	$e_{f_1}(\frac{1}{2})$	$e_{f_1}(0)$	$e_{f_2}(\frac{1}{2})$	$e_{f_2}(0)$	$\kappa(A)$
3	64	1×10^{-4}	9×10^{-4}	4×10^{-3}	4×10^{-2}	29
5	216	3×10^{-7}	3×10^{-7}	3×10^{-3}	2×10^{-3}	80
7	512	1×10^{-10}	2×10^{-10}	1×10^{-3}	1×10^{-3}	71
10	1331	3×10^{-14}	5×10^{-14}	6×10^{-4}	3×10^{-5}	145

No bad conditioning problems occur here as we have only taken values of $d \leq 10$. As noticed in section 2, the accuracy on the smooth solutions is good but the accuracy on less smooth solutions is really worse than with the sparse basis. Moreover, the complexity of this method increases quickly with Q so we no longer use it in the last two examples in dimension 4 and 5 whose results are described in the following table.

d	$L_{4,d}$	$e_{f_1}(\frac{1}{2})$	$e_{f_1}(0)$	$e_{f_2}(\frac{1}{2})$	$e_{f_2}(0)$	$\kappa(A)$
3	80	1×10^{-4}	7×10^{-4}	7×10^{-3}	5×10^{-2}	44
5	168	3×10^{-5}	1×10^{-5}	2×10^{-3}	1×10^{-3}	260
7	280	2×10^{-6}	1×10^{-5}	9×10^{-4}	2×10^{-3}	666
10	504	5×10^{-9}	5×10^{-8}	3×10^{-5}	4×10^{-5}	6007
d	$L_{5,d}$	$e_{f_1}(\frac{1}{2})$	$e_{f_1}(0)$	$e_{f_2}(\frac{1}{2})$	$e_{f_2}(0)$	$\kappa(A)$
3	192	5×10^{-5}	5×10^{-4}	9×10^{-3}	6×10^{-2}	126
5	432	1×10^{-5}	4×10^{-6}	4×10^{-3}	3×10^{-3}	1267
7	752	8×10^{-7}	5×10^{-6}	1×10^{-3}	5×10^{-3}	3273
10	1432	2×10^{-9}	3×10^{-8}	2×10^{-4}	2×10^{-4}	3.53×10^5

The approximate solutions are as accurate as the expansion of the exact solutions on the same approximation basis as we have taken only small values of d here.

Nevertheless these small values are sufficient to obtain accuracies of 8 or 9 digits on the smooth solutions and 4 or 5 digits on the less smooth solutions.

The complexity of the method depends mainly of the computation of the coefficients β_j and of the resolution of the linear system which both are a $O(L_{Q,d}^3)$ as we use a direct method to solve this linear system. The CPU time of resolution for $d = 10$ in dimension 4 was 10 seconds and 36 seconds for $d = 10$ in dimension 5 using Matlab on a standard PC. We do not count in this CPU times the time of construction of the quadratures which we consider as preprocessing. This CPU times are a lot smaller than the ones of the sequential Monte Carlo algorithm. They can still be reduced by some more preprocessing for the computation of the matrix coefficients or by using another method of resolution of the linear system. We could for example compute the coefficients β_j writing

$$\beta_j \simeq \sum_{i=1}^{[2.5 \times L_{2,d}]} \mu_{i,j} f(X_1^{(i)}, X_2^{(i)})$$

with

$$\mu_{i,j} \simeq \sum_{k=1}^{L_{2,d}} \lambda_{i,k} s_{l_1(k), l_1(j)} s_{l_2(k), l_2(j)}$$

and by storing the coefficients $\mu_{i,j}$.

5. CONCLUSION

We have studied two different ways to use a sparse polynomial basis for the numerical approximations of the Poisson equation over hypercubes and compared it to standard tensor product approximations. Both methods have provided a very good approximation of the solution whenever it is smooth on problems in dimensions up to 5. To make the sequential Monte Carlo method more efficient, one has to develop new tools to make the evaluation of the source term cheaper. The method based on the variational formulation is a lot more efficient in terms of CPU times and is also efficient for less smooth solutions. Part of the bad conditioning problems which happen with this method when the size of the sparse basis increases have been handled by adding another criterion on the choice of the basis functions. This method can certainly be still efficient in higher dimensions, maybe up to dimension 10, if the solution is very smooth. Concerning less smooth solutions, one can also think of using piecewise sparse polynomial approximations in order to diminish the condition number of the linear system or to adapt the sequential Monte Carlo domain decomposition method developed in [9] to this kind of approximation basis.

REFERENCES

- [1] I. BABUSKHA, U. BANERJEE, J.E. OSBORN, Survey of meshless methods and generalized finite elements methods: a unified approach, *Acta Numerica*, pp 1-125, 2003.
- [2] C. BERNARDI, Y. MADAY, *Approximations spectrales de problèmes aux limites elliptiques*, Springer-Verlag, 1992.
- [3] C. CANUTO, M. Y. HUSSAINI, A. QUARTERONI, T. A. ZANG, *Spectral methods in fluid dynamics*, Springer-Verlag, 1988.
- [4] S. DUMONT, O. GOUBET, T. HA-DUONG, P. VILLON, Mesh free methods and boundary conditions, *Int. J. Numer. Meth. Engng.*, Vol 67, pp 989-1011, 2006.
- [5] M. FREIDLIN. *Functional Integration and Partial Differential Equations*, Princeton University Press, 1985.

- [6] E. GOBET, Euler schemes and half-space approximations for the simulation of diffusion in a domain, ESAIM Probability and Statistics, Vol 5, pp 261-297, 2001.
- [7] E. GOBET, S. MAIRE, A spectral Monte Carlo method for the Poisson equation, Monte Carlo methods and applications Vol 10, no.3-4, pp. 275-285, 2004.
- [8] E. GOBET, S. MAIRE, Sequential control variates for functionals of Markov processes, SIAM Journal on Numerical Analysis, 43, no.3, pp. 1256-1275, 2005.
- [9] E. GOBET, S. MAIRE, Sequential Monte Carlo domain decomposition for the Poisson equation, Proceedings of the 17 imacs World congress, Scientific Computation, Applied Mathematics and Simulation (2005).
- [10] C. HWANG, M. MASCAGNI, J.A. GIVENS. A Feynman-Kac path-integral implementation for Poisson's equation using an h-conditioned Green function. Mathematics and computers in simulation 62, 2003.
- [11] S. JOE, I. SLOAN, Imbedded lattice rules for multidimensional integration, SIAM J. Numer. Anal. Vol.29, no. 4 pp. 1119-1135, 1992.
- [12] A. R. KROMMER, C. W. UEBERHUBER. Computational integration. SIAM, 1998.
- [13] B. LAPEYRE, E. PARDOUX, R. SENTIS, Méthodes de Monte-Carlo pour les équations de transport et de diffusion. Springer-Verlag, 1998.
- [14] C. LÉCOT, F. EL KHETTABI, Quasi-Monte Carlo simulation of diffusion. Dagstuhl Seminar on Algorithms and Complexity for continuous problems (1998), Journal of complexity 15, no.3, pp. 342-359, 1999.
- [15] S. MAIRE, An iterative computation of approximations on Korobov-like spaces, Journal of Computational and Applied Mathematics, 157, pp. 261-281, 2003.
- [16] S. MAIRE, Polynomial Approximations of multivariate smooth functions from quasi-random data, Statistics and Computing, 14, pp. 333-336, 2004.
- [17] S. MAIRE, C. DE LUIGI, Quasi-Monte Carlo quadratures for multivariate smooth functions, Applied Numerical mathematics, 56, pp. 146-162, 2006.
- [18] H. NIEDERREITER, Quasi-Monte Carlo methods and pseudorandom numbers, Bull. Amer. Math. Soc. 84, pp. 957-1041, 1978.
- [19] J. A. NITSCHKE, Convergence of non conforming methods, Proc. Sympos. Math. Res. Center, Univ. Wisconsin, Madison, pp 15-53, 1974.
- [20] E. NOVAK, K. RITTER, High dimensional integration of smooth functions over cubes, Numerische Mathematik, 75, pp.79-97, 1996.
- [21] G. PAGES, A space vector quantization for numerical Integration, Journal of computational and applied mathematics, 89, pp. 1-38, 1997.
- [22] T. VON PETERSDORFF, C. SCHWAB, Numerical solution of parabolic equations in high dimensions, M2AN Math. Model. Numer. Anal. 38, no.1, pp. 93-127, 2004.
- [23] I. H. SLOAN, P. J. KACHOYAN, Lattice methods for multiple integration: Theory, error analysis and examples, SIAM J. Numer. Anal. 24, pp. 116-128, 1987.
- [24] R. STENBERG, On some techniques for approximating boundary conditions in the finite element method, Journal of Computational and applied Mathematics. 63, pp 139-148, 1995.

CHRISTOPHE DE LUIGI, ISITV, UNIVERSITÉ DU SUD TOULON-VAR, AVENUE G. POMPIDOU
BP 56, F-83262 LA VALETTE DU VAR CEDEX, FRANCE

E-mail address: deluigi@univ-tln.fr

SERGE DUMONT, LAMFA, FACULTÉ DE MATHÉMATIQUES ET D'INFORMATIQUE UNIVERSITÉ
DE PICARDIE JULES VERNE, RUE SAINT LEU, F-80039 AMIENS CEDEX 1, FRANCE

E-mail address: Serge.Dumont@u-picardie.fr

SYLVAIN MAIRE, ISITV, UNIVERSITÉ DU SUD TOULON-VAR, AVENUE G. POMPIDOU BP 56,
F-83262 LA VALETTE DU VAR CEDEX, FRANCE

E-mail address: maire@univ-tln.fr