Variance reduction in Monte Carlo integration via function approximation

Federico Betti
MATH-414 Stochastic Simulation, EPFL Lausanne, Switzerland

February 1, 2022

Let us begin by recalling briefly the setting. Let $\Omega = [0,1]^d$ and $f : \Omega \to \mathbb{R}$. As usual, we are interested in computing the quantity $I = \int_\Omega f(x)\,dx$. A standard way of proceeding is by a Crude Monte Carlo estimate, i.e. we take the average over M samples $\{f(x_i)\}$ with $x_i \overset{iid}{\sim} U(\Omega)$. While this is an extremely simple to implement and well working algorithm, the convergence rate is pretty low, but many ideas have been proposed in order to reduce the constant factor appearing in the variance of the estimator. The one which is treated here relies on the approximation of f by some polynomial p, where with respect to a standard quadrature formula the interpolation nodes are drawn randomly from the uniform distribution in Ω. In this case, once we fix a grid of uniformly chosen points $\{x_i\}_{i=1}^M \in \Omega$, we construct our polynomial p by taking a linear combination of the first $n+1$ Legendre polynomials, using them as our control variates. Then the estimate of I reads

$$I_{MCLS} = \frac{1}{M}\sum_{i=1}^{M}\left(f(x_i) - \sum_{j=0}^{n} c_j\phi_j(x_i)\right) + \sum_{j=0}^{n} c_j \int_\Omega \phi_j(x)dx. \tag{1}$$

Indeed the Legendre polynomials, which we denote as $\{\phi_j\}_{j=0}^n$ for the rest of the work, are valid control variates because by their orthogonality in Ω with respect to the weight function $w = 1$ we have

$$\mathbb{E}[\phi_j] = \int_\Omega \phi_j(x)dx = \int_\Omega 1 \cdot\ \phi_j(x)dx = \langle \phi_0, \phi_j\rangle_{L^2(\Omega)} = \delta_{j,0}. \tag{2}$$

Therefore due to (2), the estimator (1) reduces to

$$I_{MCLS} = \frac{1}{M}\sum_{i=1}^{M}\left(f(x_i) - \sum_{j=0}^{n} c_j\phi_j(x_i)\right) + c_0 \tag{3}$$

for some suitable coefficients $c_j \in \mathbb{R}$.

Comparison with Crude Monte Carlo

In general it is not true that every choice of the coefficients $\{c_j\}_j$ increases the performance of the estimator. However, choosing them properly and having a high correlation between f and ϕ_j, this can provide a large variance reduction. To this aim, we choose our coefficients to be the solution of the least squares fit

$$\min_{\mathbf{c}\in\mathbb{R}^{n+1}} \sum_{i=1}^{M}\left(f(x_i) - \sum_{j=0}^{n} c_j\phi_j(x_i)\right)^2, \tag{4}$$

or equivalently

$$\min_{\mathbf{c}\in\mathbb{R}^{n+1}} ||\mathbf{V}\mathbf{c}-\mathbf{f}||_2^2, \tag{5}$$

where $\mathbf{V}\in\mathbb{R}^{M,n+1}$ is a Vandermonde matrix such that $\mathbf{V}_{i,j}=\phi_{j-1}(x_i)$ and $\mathbf{f}=\{f(x_i)\}_{i=1}^M$. As long as the samples are independent, we can interpret problem (5) as a minimization of the sample variance estimate up to a constant. As in the standard framework of multiple control variates, another possible interpretation is the linear regression one: we are looking for the coefficients c_j which fit linearly the data $\{f(x_i)\}_{i=1}^M$ so that the scalar product between the feature vector of each sample (given by the basis functions ϕ_j) and the coefficients c_j gives the minimum least squares residual. For $n=0$ all the procedure above simply reduces to a standard Monte Carlo estimator: indeed problem (5) corresponds to $\min_{c_0\in\mathbb{R}}\sum_{i=1}^M(c_0-f(x_i))^2$, for which the unique solution is

$$c_0^*=\frac{1}{M}\sum_{i=1}^M f(x_i). \tag{6}$$

Hence we can interpret a Crude Monte Carlo estimator as the simple case in which we use a constant c_0 to approximate f and then we approximate I by c_0 (for this to hold it is important that $\phi_0=1$). This is a crucial observation and the reason why we expect that increasing n the approximation is better and hence the estimate of I is too. By the normal equations, the unique solution to (5) is given by

$$\mathbf{V}^T\mathbf{V}\mathbf{c}^*=\mathbf{V}^T\mathbf{f}, \tag{7}$$

where the Gram matrix $\mathbf{V}^T\mathbf{V}$ is invertible as long as $M>n$ (i.e. $\mathbf{V}$ has full column rank). In such a case, for numerical stability, we solve (7) by computing the reduced $\mathbf{QR}$ factorization of $\mathbf{V}$ (i.e. with $\mathbf{Q}\in\mathbb{R}^{M,k}$ and $\mathbf{R}\in\mathbb{R}^{k,n+1}$ where $k=\min\{M,n+1\}$), instead of inverting directly $\mathbf{V}^T\mathbf{V}$ which has conditioning number $k_2(\mathbf{V}^T\mathbf{V})=k_2(\mathbf{V})^2$. That is, we solve in the end the much cheaper problem $\mathbf{R}\mathbf{c}^*=\mathbf{Q}^T\mathbf{f}$. We can finally construct our control variate estimator by taking $c_j=c_j^*$ in (3), which gives

$$I_{MCLS}=\frac{1}{M}\sum_{i=1}^M\left(f(x_i)-\sum_{j=0}^n c_j^*\phi_j(x_i)\right)+c_0^*. \tag{8}$$

Since no pilot run is performed to estimate the coefficients c_j using another set of samples, (8) is not an unbiased estimator of I. However, we know from the theory that this algorithm is still working well as the bias is vanishing asymptotically and by this one-shot procedure we don't "waste" the pilot samples. In the end, note that by the first normal equation

$$0=(\mathbf{V}^T\mathbf{f})_1-(\mathbf{V}^T\mathbf{V}\mathbf{c}^*)_1=\sum_{i=1}^M\left((\mathbf{V}^T)_{1i}f(x_i)-\sum_{j=0}^n(\mathbf{V}^T)_{1i}V_{ij}c_j^*\right)=\sum_{i=1}^M\left(f(x_i)-\sum_{j=0}^n V_{ij}c_j^*\right), \tag{9}$$

which corresponds to the sum in the first term in (8), since $\phi_0=1$. Therefore for the optimal regression coefficients c_j^* (8) simply reduces to

$$I_{MCLS}=c_0^*. \tag{10}$$

If a pilot run is performed the equivalence between (8) and (10) is clearly not true anymore and we would have an unbiased estimator of I for all M as the Vandermonde matrices used for the estimation of I and for solving (7) are not the same. Given this observation, for the rest of the work, once we solve (5), we will simply estimate I by means of (10).

Let us now analyse the performance of the I_{MCLS} estimator on a simple one dimensional case. We aim at estimating the integral of the Runge function, namely

$$I_1 = \int_0^1 \frac{1}{25x^2+1} dx = \frac{1}{5}\arctan(5), \tag{11}$$

for which an exact value is known, using different values of n (including $n = 0$) and M. In Figure 1 we see that as we expect the performance of (10) increases as the number of Legendre polynomials n does. Moreover being n fixed we observe coherently the scaling of the error with $\frac{1}{\sqrt{M}}$.
Concerning the conditioning of the Vandermonde matrix involved in the least squares fit (7), for relatively small values of n the conditioning of the Vandermonde matrix is still under control and the latter is not affecting majorly the accuracy of the estimators. In particular, for the standard MC estimator ($n = 0$) there are no issues since for any matrix $V \in \mathbb{R}^{M,1}$ the condition number is 1. However the more we increase n we notice an increase in the conditioning of V and the ill-conditioning in the preasymptotic stage is accentuated: this is due to the number of samples being initially insufficient. As a consequence of this the error initially doesn't have the correct asymptotic behaviour, and it stabilizes only after enough samples are taken.

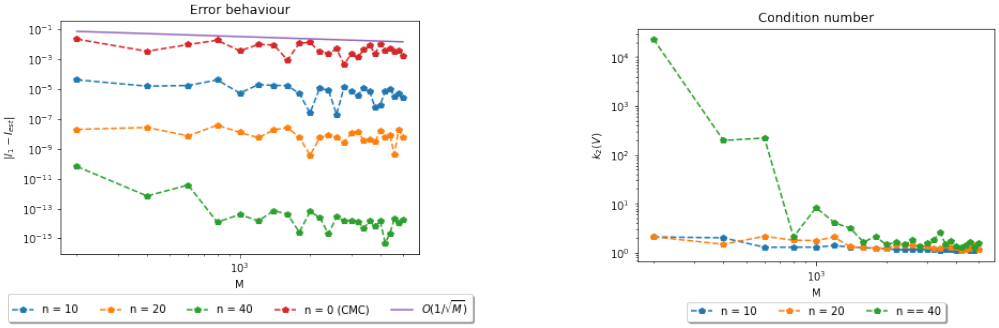

Figure 1: Errors on I_1 (left) and condition numbers of the Vandermonde Matrix (right) for fixed values of n with the $MCLS$ estimator.

To give another perspective of how well we are approximating I_1, we further compare the performance of I_{MCLS} with another variance reduction technique which works well for the one dimensional case as long as f is smooth enough, namely a *randomized midpoint quadrature estimator* of the form

$$\mu_s = \frac{1}{s}\sum_{j=1}^{s} f\left(\frac{j-1+U^{(j)}}{s}\right), \qquad U^{(j)} \sim U([0,1]). \tag{12}$$

In Figure 2 we see that also (12) performs well in this one dimensional case (but recall that its convergence rate suffers from the curse of dimensionality), but for a relatively small value of n the I_{MCLS} estimator is already outperforming it. We see that taking approximately $n = 10$ we match the accuracy of (12) with our control variate technique. Being therefore the randomized midpoint a very valid variance reduction technique in this case (see the improvement with respect to the case $n = 0$ perhaps) this again shows how powerful our I_{MCLS} estimator can be.

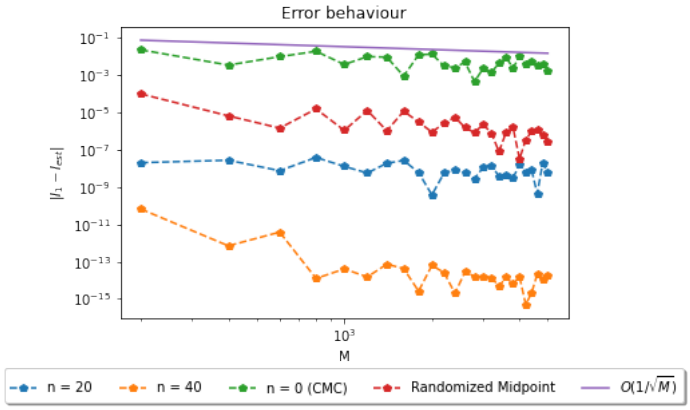

Figure 2: Comparison between the estimators (10) and (12) for fixed values of n.

What is moreover clear from Figure 1 is that increasing n can potentially be crucial to increase the performance of our estimator. Indeed, it has been shown in [3] that choosing n as a function of M can provide higher convergence rates, as long as one is able to control the conditioning of the Vandermonde matrix and the samples x_i are chosen carefully.

In Figure 3 we notice that this is actually the case; while for $n = [\sqrt{M}]$ we are able to increase n without running into conditioning issues and we have a very quick convergence, for $n = [\frac{M}{2}]$ and $N = [\frac{M}{\log(M)}]$ the condition number explodes and this results in a very poor and not converging estimation of I_1. However initially the behaviour seems promising even with a quite large condition number, and the estimate becomes worse as we increase the number of samples M. On the other hand, for $n = [M \log(M)]$ the Vandermonde matrix doesn't have full column rank and indeed the estimate is really poor. It is then clear that we could aim at lowering the conditioning number of the Vandermonde matrix by slightly modifying problem (7) to reduce the conditioning of the matrix involved. In particular, it turns out that a smarter choice of the samples helps in this direction and hence gives optimal convergence rates for the cases $n = [\frac{M}{2}]$ and $n = [\frac{M}{\log(M)}]$.

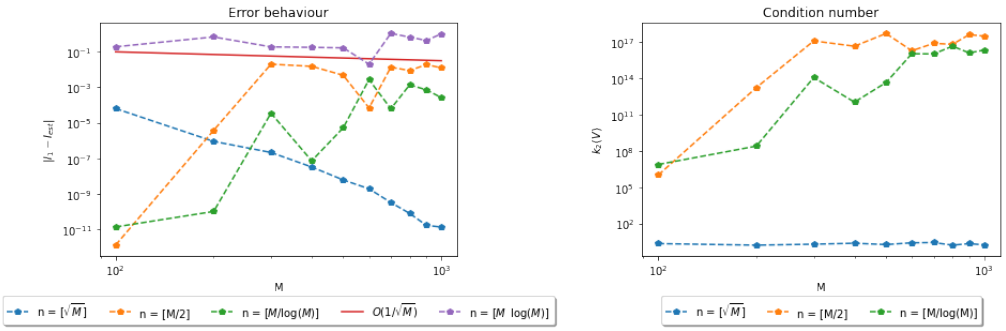

Figure 3: Errors on I_1 (left) and condition numbers of the Vandermonde Matrix (right) for values of n as a function of M using the $MCLS$ estimator.

Quasi Monte Carlo MCLS

Before moving on with the optimal choice of the samples in the following section, we present here a first (reasonable but naive) possible idea of a better selection of the samples which follows from having observed the good performance of (12) with respect to a Crude Monte Carlo. Given also the higher accuracy of I_{MCLS} taking $n > 1$ with respect to $n = 0$ for it is natural to expect even better results if we were to combine the two things. Namely, we could think of constructing a $MCLS$ estimator where the x_i are chosen as the same samples used to construct (12) in a Quasi Monte Carlo fashion. Such a point set achieves the optimal bound on the star discrepancy in one dimension. In Figure 4 on the left we see the performance of this estimator, which we call $QMCLS$, for fixed values of n: we observe the correct scaling of the error with $\frac{1}{M}$ and the differences from the case $n = 0$ to higher n are comparable to the ones observed for the $MCLS$ estimator. On the other hand we see on the right that for $n = [\frac{M}{2}]$ and $n = [\frac{M}{\log(M)}]$ the conditioning issues concerning $\mathbf{V}$ are not solved yet, as expected since we are not acting at all on the conditioning of $\mathbf{V}$ in the problem (5) but we are just covering Ω more evenly and quickly.

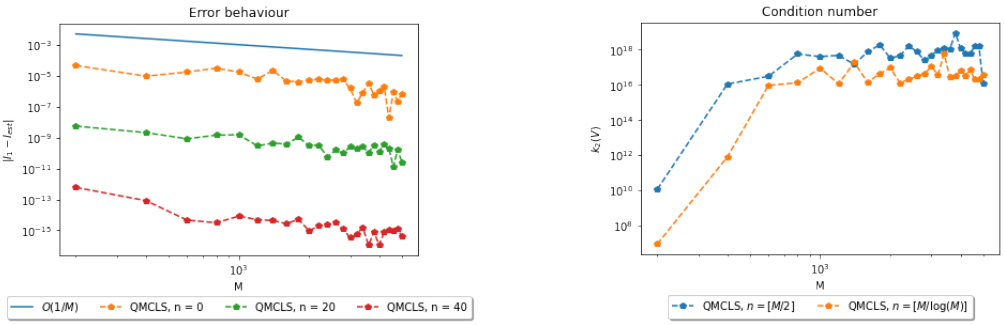

Figure 4: Errors on I_1 for fixed values of n (left) and conditioning numbers of the Vandermonde matrix for n as a function of M (right) for the $QMCLS$ estimators.

Importance sampling

To further improve in this direction we follow the idea presented in [1] and we perform importance sampling: instead of sampling from the uniform distribution in Ω we draw from the distribution

$$\frac{1}{w(x)} = \frac{\sum_{j=0}^{n} \phi_j^2(x)}{n+1}. \tag{13}$$

Using the orthogonality of the ϕ_j in Ω it is straightforward to verify that (13) is a density. Then

$$\int_\Omega \left(f(x) - \sum_{j=0}^{n} c_j \phi_j(x) \right)^2 dx = \int_\Omega w(x) \left(f(x) - \sum_{j=0}^{n} c_j \phi_j(x) \right)^2 \frac{1}{w(x)} dx \tag{14}$$

$$\approx \frac{1}{M} \sum_{i=1}^{M} \left(f(\tilde{x}_i) - \sum_{j=0}^{n} c_j \phi_j(\tilde{x}_i) \right)^2 w(\tilde{x}_i), \tag{15}$$

with $\tilde{x}_i \overset{iid}{\sim} 1/w$. Hence in this case we aim at solving a weighted version of (7):

$$\min_{\mathbf{c} \in \mathbb{R}^{n+1}} ||\sqrt{\mathbf{W}}(\mathbf{V}\mathbf{c} - \mathbf{f})||_2^2, \tag{16}$$

with $\sqrt{\mathbf{W}} = \text{diag}(\{\sqrt{w(\tilde{x}_i)}\}_{i=1}^M)$. The MCLS estimator (8) generalizes to this importance sampling case as follows: we have

$$I_{MCLS,IS} = \frac{1}{M}\sum_{i=1}^{M}\left(f(\tilde{x}_i) - \sum_{j=0}^{n} c_j\phi_j(\tilde{x}_i)\right)w(\tilde{x}_i) + c_0. \tag{17}$$

Using a similar proof to the case $W = I_M$, the first normal equation for the solution of (16) shows that taking the optimal coefficients c_j^* (17) reduces again to

$$I_{MCLS,IS} = c_0^*. \tag{18}$$

In this case, for $n = 0$ (18) corresponds to a weighted importance sampling estimator since the problem $\min_{c_0\in\mathbb{R}}\sum_{i=1}^M w(\tilde{x}_i)(c_0 - f(x_i))^2$ has as optimal solution

$$c_0^* = \sum_{i=1}^{M} f(\tilde{x}_i)\hat{w}(x_i), \quad \hat{w}(x_i) = \frac{w(\tilde{x}_i)}{\sum_{i=1}^M w(\tilde{x}_i)}. \tag{19}$$

As in the standard case, for numerical stability when $\mathbf{V}$ has full column rank we solve the normal equations for (16), that is $\mathbf{V}^T\mathbf{W}\mathbf{V}\mathbf{c}^* = \mathbf{V}^T\mathbf{W}\mathbf{f}$, using the reduced $\mathbf{QR}$ factorization of $\sqrt{\mathbf{W}}\mathbf{V}$, which gives in the end $\mathbf{R}\mathbf{c}^* = \mathbf{Q}^T\sqrt{\mathbf{W}}\mathbf{f}$. Concerning the algorithm to sample from (13), the first thing that one may think about is to perform a Metropolis-Hastings Algorithm using as proposal density the uniform distribution in Ω. However, we don't want to introduce correlation in the samples as this would not be coherent with the interpretation given in the introduction of the least squares fit (5), and even performing subsampling to recover independence seems too costly and suboptimal. Alternatively, we take inspiration from the bound proposed in [2] and we perform a basic acceptance-rejection exploiting the fact that the arcsine distribution

$$g(x) = \frac{1}{\pi\sqrt{x(1-x)}}, \quad x \in (0,1) \tag{20}$$

satisfies $1/w(x) \leq 4e\, g(x)$ in the one dimensional case. Since even in higher dimension each squared basis Legendre polynomial ϕ_j^2 can be factorized as $\phi_j^2 = \prod_{i=1}^d \phi_{j_i}^2$, i.e. the product of d one dimensional squared basis polynomials, we have that $\phi_j^2(x) \leq (4e)^d\prod_{i=1}^d g(x_i)$ where $x = \{x_i\}_{i=1}^d \in \Omega$ and hence we can use the product density $\prod_{i=1}^d g(x_i)$ as our proposal density. The drawback is the deteriorating of the acceptance rate as d grows. In the algorithm below $\frac{1}{w}$ is the target distribution.

Algorithm 1 Sampling algorithm from (13)

Input: the number of basis Legendre polynomials n, the dimension d of Ω and the number of samples M

While less than M samples are accepted:

1. Generate the proposed sample $y = \{y_i\}$ from the product distribution $\prod_{i=1}^d g(x_i)$ by sampling $y_i \overset{iid}{\sim} g$ for $i = 1, \ldots, d$.
2. Let $\frac{1}{\alpha} = (4e)^d w(y)\prod_{i=1}^d g(y_i)$, where w depends on the first $n+1$ basis polynomials, and accept the sample if $U \leq \alpha$ where $U \sim U([0,1])$, otherwise return to Step 1.

Output: M samples distributed according to (13)

Let us now discuss the performance of the importance sampling estimator on the approximation of I_1. Even though it is not explicitly asked, from the discussion before it comes natural to check also the conditioning of the weighted Vandermonde matrix $\sqrt{\mathbf{W}}\mathbf{V}$ to see if a lowering of the latter is one of the reasons of our anticipated improvements. We can indeed notice in Figure 5 a better behaviour already in the case in which we maintain the number of Legendre polynomials n fixed and not depending on M, as we seem to control better the ill-conditioning in the preasymptotic stage.

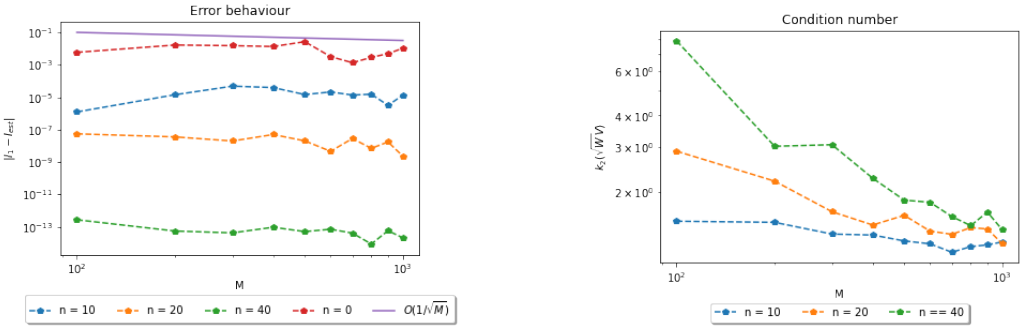

Figure 5: Errors on I_1 (left) and condition numbers of the weighted Vandermonde Matrix (right) for fixed values of n with the estimator $I_{MCLS,IS}$.

In the more interesting case in which n is chosen as a function of M, for which higher convergence rates are expected, we test once again the performance of (10) in the cases $n = [\sqrt{M}]$, $n = [\frac{M}{2}]$, $n = [\frac{M}{\log(M)}]$ and $n = [M\log(M)]$. In Figure 6, we observe a massive decrease of the condition number of the weighted Vandermonde matrix in the cases $n = [\frac{M}{2}]$ and $n = [\frac{M}{\log(M)}]$: this results in an extremely high accuracy as the error is of the order of machine precision already for small M. In particular, it has been proved in [4] that the latter is the optimal choice as it provides super-algebraic convergence as long as f is analytic, but we observe very high convergence rates also for $n = [\frac{M}{2}]$. Moreover, also in this case, we seem to be controlling better the ill-conditioning for an insufficient number of samples. The importance sampling seems instead useless for the case $n = [M\log(M)]$ which is still performing badly as the system is underdetermined.

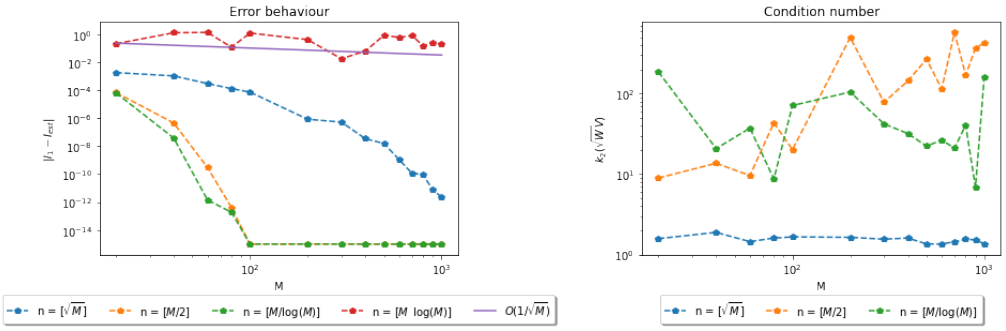

Figure 6: Errors on I_1 (left) and condition numbers of the weighted Vandermonde Matrix (right) for values of n as a function of M with the estimator $I_{MCLS,IS}$.

Fitzhugh-Nagumo model

We want now to use the previously discussed methods on a specific problem of interest, in particular the Fitzhugh–Nagumo system of ODEs. The system reads

$$\begin{cases} \dot{v} = v - \frac{v^3}{3} - w + I \\ \dot{w} = \epsilon(v + a - bw) \\ v(0) = v_0, w(0) = w_0. \end{cases} \tag{21}$$

We set $I = 1.0$, $\epsilon = 0.08$, $v_0 = w_0 = 0$ and we impose $a \sim U([0.6, 0.8])$ and $b \sim U([0.7, 0.9])$. We remark here that since (21) represent oscillatory dynamics probably we could have looked for a more proper integrator that possibly conserves first integrals of the system, hence doing a better mimic of the latter. However, for simplicity, we integrate such a system using a Forward Euler scheme in the interval $[0, T = 10]$ using a step size $\Delta t = 0.01$. Having computed the solution $(v(t), w(t))$, we are interested in computing the average of

$$Q = \frac{1}{T} \sum_n \left(\frac{(v_n)^2 + (v_{n+1})^2}{2} \right) \Delta t, \tag{22}$$

which is itself an approximation of the average of v^2 in $[0, T]$: namely we aim at estimating the integral

$$I_2 = 0.04 \int_{0.6}^{0.8} \int_{0.7}^{0.9} Q \, dbda. \tag{23}$$

In the proceeding of the work, we aim again at comparing the performance of a Crude Monte Carlo estimator and of our least squares estimator in approximating I_2. Since it is not obvious how the Legendre normalization constants for the polynomials translate to the intervals $[0.6, 0.8]$ and $[0.7, 0.9]$, we decide to manipulate the ODE (21) as follows: we solve the system for $\tilde{a}, \tilde{b} \sim U([0, 1])$ where we substitute a by $0.2\tilde{a} + 0.6$ and b by $0.2\tilde{b} + 0.7$. In this way we re-conduce ourselves to the estimation of an integral on $(0, 1)$ x $(0, 1)$ and we can generalize the one dimensional pipeline constructed before more easily. Note that this is equivalent to expressing (23) by means of

$$I_2 = 0.04 \int_0^1 \int_0^1 Q(0.2\tilde{a} + 0.6, 0.2\tilde{b} + 0.7) \, d\tilde{b}d\tilde{a}. \tag{24}$$

Moreover, it is not clear what basis polynomials should be picked (picking for example xy over x^2 or the contrary), so for each value of n we compute the Legendre series using all possible products of the one dimensional polynomials such that the sum of the respective degrees is at most n. This means that the resulting Vandermonde matrix has now dimension M, $\frac{(n+1)(n+2)}{2}$. To further test the accuracy of our estimator, since in this case an exact value of I_2 is not computable, we aim at computing the variance of (8) to then build confidence intervals. For this, we first present the result of Theorem 3.1 in [4], which holds when the number of Legendre polynomials n is fixed.

Theorem 0.1. *Fix n and the basis functions $\{\phi_j\}_{j=0}^n$. Then it holds that asymptotically*

$$\sqrt{M}(I_{MCLS} - I) \xrightarrow{d} N\left(0, \min_{\mathbf{c} \in \mathbb{R}^{n+1}} ||f - \sum_{j=0}^{n} c_j \phi_j||_2^2\right) \tag{25}$$

where $\xrightarrow{d}$ denotes convergence in distribution.

The result presented in Theorem 0.1 is perhaps expected since the least squares fit introduced in (5) was already interpreted in the introduction as a minimization of the sample variance estimate. Moreover, by a classical Monte-Carlo analysis the variance was expected in the form $\frac{1}{M}\mathbb{V}ar(f-p)$ where p is the polynomial approximating f in Ω in the L^2 norm: Theorem 0.1 shows that this is the correct asymptotic behaviour. (25) generalizes the error estimate of a standard Monte Carlo procedure where we have a variance of the form $\frac{1}{M}\min_{c\in\mathbb{R}}||f-c||_2^2$ to this case where it is instead $\frac{1}{M}\min_{\mathbf{c}\in\mathbb{R}^{n+1}}||\mathbf{f}-\sum_{j=0}^{n}c_j\phi_j||_2^2$. Hence the variance of the $MCLS$ estimator is closely related to the residual of the least squares fit when we solve (7), namely to the quality of the approximation of f in Ω. It comes then as no surprise that we obtain variance reduction by taking $n>1$ with respect to a Crude Monte Carlo. In practice, once we solve for the optimal coefficients c_j^* in (5), we can estimate the quantity $\min_{\mathbf{c}\in\mathbb{R}^{n+1}}||f-\sum_{j=0}^{n}c_j\phi_j||_2^2$ by means of

$$\hat{\sigma}_{LS}^2=\frac{1}{M-n-1}||Vc^*-f||_2^2. \tag{26}$$

We want then to compare the true error we commit with respect to the estimated error coming from Theorem 0.1 to see how reliable of an error estimate (25) can be. We first compute a reference value of I_2 using (10) for a large enough number of samples. In particular we pick $n=5$ and $M=10^4$ and our reference value is

$$I_{ref}=0.11745134770633889\pm 1.4091868002196936e-13 \tag{27}$$

where we use the half size of the 95% confidence interval as an estimation of the error. Computed this value, we carry out our analysis by testing the performance for the cases $n=0,1,2,3$. That is, for each value of n we follow the pipeline presented in Algorithm 2.

Algorithm 2 $MCLS$ estimator for I_2 with error estimate

Input: the number of samples M, the maximum degree of Legendre polynomials n, the reference value I_{ref}, α

1. Solve for the optimal coefficients c_j^* in (5).
2. Compute $I_{MCLS}=c_0^*$ as an estimate of I_{ref}.
3. Compute the true error δ with respect to (27).
4. Estimate $\hat{\sigma}_{LS}^2=\frac{1}{M-n^*}||Vc^*-f||_2^2$, where $n^*=\frac{(n+1)(n+2)}{2}$.
5. Construct a $1-\alpha$ confidence interval $[I_{MCLS}-\Delta, I_{MCLS}+\Delta]$ where

$$\Delta=c_{1-\alpha/2}\frac{\hat{\sigma}_{LS}}{\sqrt{M}},\quad c_\alpha=\Phi^{-1}(\alpha). \tag{28}$$

Output: the estimate I_{MCLS}, the true error δ, $\hat{\sigma}_{LS}^2$, the half size of the $1-\alpha$ confidence interval Δ

As we see in Figure 7, we now have a coherent and robust estimate of the error given by (26) which enters naturally in the pipeline of the estimation and for which we can build consistent and reliable confidence intervals. Note that since we are not taking n as a function of M we cannot expect in principle a higher convergence rate and we see the usual scaling of the error as $\frac{1}{\sqrt{M}}$.

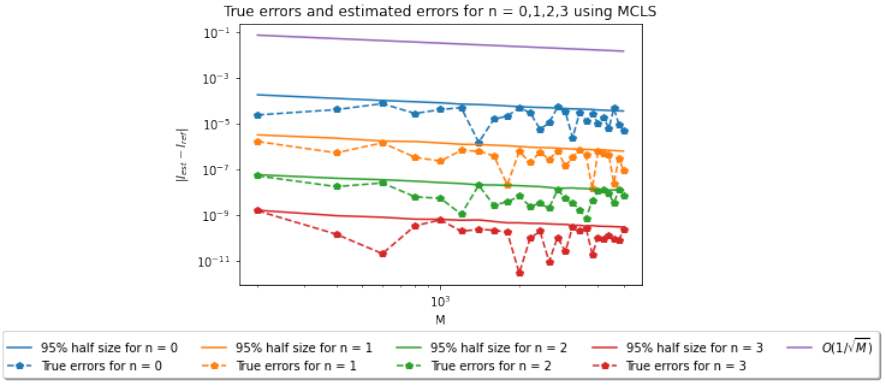

Figure 7: True and estimated errors on I_2 for the $MCLS$ estimator.

To conclude, it is natural to try to extend the $QMCLS$ estimator introduced in the previous sections to this two dimensional case. Since we cannot generalize the stratification idea applied in the one dimensional case since this is not giving a low star discrepancy set anymore, we draw the samples using the Sobol sequence. In Figure 8 we present the true errors of the $QMCLS$ estimator for I_2 in the cases ranging from $n = 0$ to $n = 2$: also in this case a better coverage of the unit square is giving better results in terms of accuracy and we observe the correct scaling with $\frac{1}{M}$.

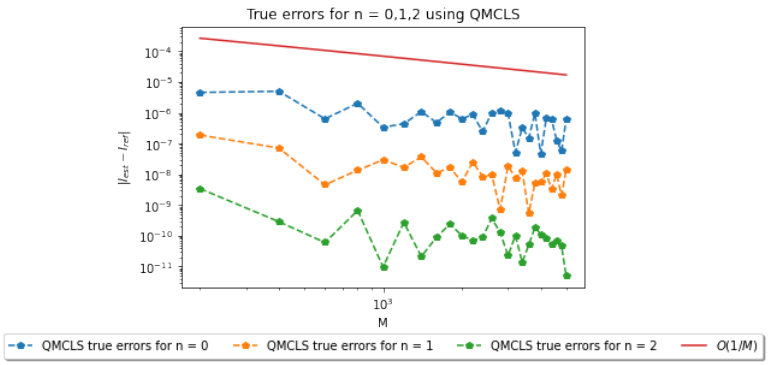

Figure 8: True errors on I_2 for the $QMCLS$ estimator.

Discussion and final remarks

Given the comparison with a randomized midpoint formula, the very high accuracy of the $MCLS$ estimator was indeed expected from the beginning as long as we could control the conditioning of V. In this direction, we achieve large improvements by applying importance sampling: the decaying of the error is way smoother, we solve the ill-conditioning problems in the preasymptotic stage and we gain extremely rapid convergence when taking $n = [\frac{M}{2}]$ and $n = [\frac{M}{\log(M)}]$. On the other hand, as long as n is fixed, Theorem 0.1 gives an intuitive error estimate related to the quality of the approximation of f. We conclude that this is a powerful variance reduction technique which generalizes rather easily to higher dimensions, even if the construction of the basis Legendre polynomials becomes expensive.

```python
# IMPORT LIBRARIES
import numpy as np
import scipy.stats as st
# the package eval_sh_legendre works by default on the interval [0,1]
from scipy.special import eval_sh_legendre
# import QMC libraries from laboratory 9 to compute QMCLS estimates
from sobol_new import *


# FUNCTIONS TO BE USED FOR THE ESTIMATIONS OF THE 1D INTEGRAL

# CRUDE MONTE CARLO ESTIMATOR FOR THE INTEGRAL OF g
def crude_monte_carlo(g, M, I_exact):
    """
    This function computes a CMC estimate of I_exact
    :param g: the function to be integrated
    :param M: the number of samples
    :param I_exact: the exact value of the integral
    :return: the value of the CMC estimate and the true error with respect to I_exact
    """
    # draw M uniform points
    U = st.uniform.rvs(size=M)
    # compute the CMC estimator
    estimate_ = np.mean(g(U))
    # compute the error with respect to the exact value
    error_ = np.abs(I_exact - estimate_)
    return estimate_, error_


# RANDOMIZED MIDPOINT QUADRATURE ESTIMATOR FOR THE INTEGRAL OF g
def randomized_midpoint(g, M, I_exact):
    """
    This function computes a randomized midpoint quadrature estimator for I_exact
    :param g: the function to be integrated
    :param M: the number of strata and thus of samples (1 point per stratum)
    :param I_exact: the exact value of the integral
    :return: the value of the stratified estimator and the true error
             with respect to I_exact
    """
    # draw a uniform random variable in each stratum
    U = st.uniform.rvs(loc=np.arange(M) / M, scale=1/M)
    # compute the stratified estimator
    estimate_ = np.mean(g(U))
    # compute the error with respect to the exact value
    error_ = np.abs(I_exact - estimate_)
    return estimate_, error_


# MCLS ESTIMATOR WITHOUT IMPORTANCE SAMPLING FOR THE INTEGRAL OF g (UNIFORMLY OR QMC DRAWN SAMPLES)
def interpolation_estimator(n, g, M, I_exact, QMC=False):
    """
    This function computes the control variate MCLS estimator of I_exact
    after interpolation of f via Legendre polynomials
    :param QMC: a boolean whether the samples are drawn in a QMC fashion by stratification
    :param n: the maximum degree of the Legendre polynomials
    :param g: the function to integrated
    :param M: the number of samples
    :param I_exact: the exact value of the integral
```

```
    :return: the value of the MCLS estimator, the true error with respect to I_exact
             and the conditioning of the Vandermonde matrix involved in solving for
             the regression coefficients
    """
    # construct a random grid of points in [0,1]
    if QMC:
        # draw samples in a QMC fashion stratifying [0,1] (low star discrepancy set)
        x_grid = st.uniform.rvs(loc=np.arange(M) / M, scale=1/M)
    else:
        # draw uniformly samples
        x_grid = st.uniform.rvs(size=M)
    # construct the Vandermonde matrix, normalization of the polynomials with \sqrt{2*i+1}
    degs = np.arange(n+1)
    degs = degs[..., np.newaxis]
    x_grid = x_grid[..., np.newaxis]
    V = (np.sqrt(2 * degs + 1) * eval_sh_legendre(degs, x_grid.T)).T
    # compute the conditioning of the Vandermonde matrix
    cond = np.linalg.cond(V)
    # solve the normal equations to find the optimal coefficients for the regression
    if n + 1 <= M:
        # if V has full column rank calculate the reduced QR factorization of V
        # ---> reduced to solving a triangular system
        Q, R = np.linalg.qr(V)
        c = np.linalg.solve(R, Q.T @ g(x_grid))
    else:
        # if V doesn't have full column rank call the solver
        c = np.linalg.solve(V.T @ V, V.T @ g(x_grid))
    # compute the estimator
    estimate_ = c[0]
    # compute the error with respect to the exact value
    error_ = np.abs(I_exact - estimate_)
    return estimate_, error_, cond


# ACCEPTANCE REJECTION METHOD TO DRAW SAMPLES FROM 1/w
def sampling_from_legendre(n, M, w):
    """
    This function performs an acceptance-rejection method to sample from the distribution 1/w using the arcsine
    distribution as the proposal in [0,1] until M samples aren't accepted
    :param n: the maximum degree of the Legendre polynomials
    :param M: the number of samples
    :param w: sampling from the distribution 1/w
    :return: the M samples distributed according to 1/w and the acceptance rate
    """
    # initialize the vector of the samples and the counter
    X = np.array([])
    count = 0
    while X.shape[0] < M:
        # increase by M the number of trials
        count += M
        # draw M samples distributed according to the proposal distribution
        Y = st.arcsine.rvs(size=M)
        # draw M uniform samples
        U = st.uniform.rvs(size=M)
        # get the accepted samples
        Accepted = np.where
                   (U <= 1 / (w(Y, n) * 4 * np.exp(1) * st.arcsine.pdf(Y)))[0]
        # check on the final number of samples accepted
```

```
        if len(Accepted) >= M-X.shape[0]:
            Accepted = Accepted[:(M-X.shape[0])]
        # concatenate the accepted samples to the already existing ones
        X = np.concatenate([X, Y[Accepted]])
    return X, M/count


# IMPORTANCE SAMPLING MCLS ESTIMATOR FOR THE INTEGRAL OF g (SAMPLES DRAWN FROM THE DISTRIBUTION 1/w)
def importance_sampling_estimator(n, g, w, M, I_exact):
    """
    This function computes the importance sampling MCLS estimator to solve conditioning problems for n = n(M)
    :param n: the maximum degree of the Legendre polynomials
    :param g: the function to be integrated
    :param w: 1/w is the distribution of the samples
    :param M: the number of samples
    :param I_exact: the exact value of the integral
    :return: the value of the importance sampling MCLS estimator, the true error with respect to I_exact, the conditioning of the weighted Vandermonde matrix involved in solving for the regression coefficients and the acceptance rate of the acceptance-rejection algorithm to draw samples distributed as 1/w
    """
    # draw the samples obtained by acceptance-rejection
    Y, accpt = sampling_from_legendre(n, M, w)
    # compute the weight matrix
    W = np.diag(w(Y, n))
    # compute the Vandermonde matrix, normalization of the Legendre polynomials with \sqrt{2*i+1}
    degs = np.arange(n + 1)
    degs = degs[..., np.newaxis]
    Y = Y[..., np.newaxis]
    V = (np.sqrt(2 * degs + 1) * eval_sh_legendre(degs, Y.T)).T
    # compute the conditioning of the weighted Vandermonde matrix
    cond = np.linalg.cond(np.sqrt(W) @ V)
    # solve the normal equations to find the optimal coefficients for the regression
    if n + 1 <= M:
        # if \sqrt{W} V has full column rank calculate its reduced QR factorization
        # ---> reduced to solving a triangular system
        Q, R = np.linalg.qr(np.sqrt(W) @ V)
        c = np.linalg.solve(R, Q.T @ np.sqrt(W) @ g(Y))
    else:
        # if V doesn't have full column rank call the solver
        c = np.linalg.solve(V.T @ W @ V, V.T @ W @ g(Y))
    # compute the importance sampling estimator
    estimate_ = c[0]
    # compute the error with respect to the exact value
    error_IS = np.abs(I_exact - estimate_)
    return estimate_, error_IS, cond, accpt


# FUNCTIONS TO BE USED FOR THE ESTIMATIONS OF THE 2D INTEGRAL

# SOLVING THE FITZHUGH-NAGUMO MODEL FOR SOME FIXED VALUES OF a AND b BY EXPLICIT EULER
def solve_Fitzhugh_Nagumo(a, b, dt, T, v0=0, w0=0, epsilon=0.08, I_=1):
    """
    This function solves the Fitzhugh-Nagumo system using the Explicit Euler Method
    :param a: a uniform random variable
    :param b: a uniform random variable
```

```
    :param dt: the integration time step for the ODE
    :param T: the final time of integration for the ODE
    :param v0: the initial condition on v
    :param w0: the initial condition on w
    :param epsilon: a parameter of the system
    :param I_: a parameter of the system
    :return: the numerical solution (v(t),w(t)) of the ODE
    """
    # initialization
    v = np.zeros(int(T/dt))
    w = np.zeros(int(T/dt))
    v[0] = v0
    w[0] = w0
    for i in range(len(v)-1):
        # compute the right hand side making the change of variable in a and b
        rhs = [v[i] - v[i] ** 3 / 3 - w[i] + I_,
               epsilon*(v[i] + 1/5*a+3/5 - (1/5*b+0.7)*w[i])]
        # Explicit Euler step
        v[i+1] = v[i] + dt * rhs[0]
        w[i+1] = w[i] + dt * rhs[1]
    return v, w


# CALCULATE THE QUANTITY OF INTEREST Q
def QoI(a, b, dt, T, v0=0, w0=0, epsilon=0.08, I_=1):
    """
    This function computes the quantity of interest
    :param a: a uniform random variable
    :param b: a uniform random variable
    :param dt: the integration time step for the ODE
    :param T: the final time of integration for the ODE
    :param v0: the initial condition on v
    :param w0: the initial condition on w
    :param epsilon: a parameter of the system
    :param I_: a parameter of the system
    :return: the quantity of interest Q
    """
    # get the numerical solution v
    v_, _ = solve_Fitzhugh_Nagumo(a, b, dt, T, v0, w0, epsilon, I_)
    # compute Q
    Q = 0.04 * dt / T * np.sum((v_[:-1] ** 2 + v_[1:] ** 2) / 2)
    return Q


# COMPUTE A CRUDE MONTE CARLO ESTIMATE OF THE AVERAGE OF Q
def crude_monte_carlo_Q(M, dt, T, Q_ref):
    """
    This function computes a CMC estimate of the average of Q
    :param M: the number of samples
    :param dt: the integration time step for the ODE
    :param T: the final time of integration for the ODE
    :param Q_ref: the reference value for the average of Q
    :return: the CMC estimate and the error with respect to Q_ref
    """
    # draw M uniform points in [0,1]^2
    X = st.uniform.rvs(size=(2, M))
    # calculate the quantity of interest for each sample
    Q = np.array([QoI(X[0, i], X[1, i], dt, T) for i in range(M)])
    # compute the estimator
    estimate_ = np.mean(Q)
```

```
    # compute the error
    error_ = np.abs(Q_ref - estimate_)
    return estimate_, error_


# CONSTRUCTION OF THE VANDERMONDE MATRIX FOR THE 2d CASE (UNIFORMLY OR QMC DRAWN SAMPLES)
def basis_2d_legendre(k, M, QMC=False):
    """
    This function computes the Vandermonde Matrix for 2d Legendre polynomials taking only products that are less of degree k combined
    :param QMC: a boolean whether the samples are drawn in a QMC fashion using the Sobol sequence
    :param k: the highest degree of the Legendre polynomials in each dimension, so that the resulting Vandermonde matrix will have dimension M,n where n = (k+1)*(k+2)/2
    :param M: the number of samples
    :return: the Vandermonde matrix, the samples and the number of degrees of freedom (i.e. columns of the Vandermonde matrix)
    """
    if QMC:
        # draw the samples in a QMC fashion using the Sobol sequence
        X = generate_points(M, 2, 0).T
    else:
        # draw uniformly samples
        X = st.uniform.rvs(size=(2, M))
    # number of basis functions of degree <= k in the two dimensional case
    n = np.int32((k+1)*(k+2)/2)
    # initialize the Vandermonde matrix
    V = np.zeros((M, n))
    # initialize the counter
    count = 0
    for i in range(k + 1):
        for j in range(k + 1):
            # if the sum of the degrees is below k
            if i + j <= k:
                # construct the count-th column of the V matrix by multiplying
                # the basis polynomials of degree i and j
                V[:, count] = np.sqrt(2*i + 1) * np.sqrt(2*j + 1) \
                              * eval_sh_legendre(i, X[0, :]) * eval_sh_legendre(j, X[1, :])
                count += 1
    return V, X, n


# MCLS ESTIMATORS FOR THE AVERAGE OF Q (UNIFORMLY OR QMC DRAWN SAMPLES)
def interpolation_estimator_Q(k, M, dt, T, Q_ref, alpha=0.05, QMC=False):
    """
    This function computes the control variate estimator of the average of Q and estimates the error
    :param QMC: a boolean whether the samples are drawn in a QMC fashion using the Sobol sequence
    :param k: the highest degree of the Legendre polynomials,
              so that the resulting Vandermonde matrix will have dimension M,n where
              n = (k+1)*(k+2)/2
    :param M: the number of samples
    :param dt: the integration time step for the ODE
    :param T: the final time of integration for the ODE
    :param Q_ref: the reference value for the average of Q
    :param alpha: a parameter for the confidence interval
```

```
    :return: the value of the MCLS estimator, the true error with respect to Q_ref
             and the half-size of the (1-alpha) confidence interval
    """
    # get the Vandermonde matrix, the samples and the number of degrees of freedom
    V, X, n = basis_2d_legendre(k, M, QMC)
    # compute Q for each sample
    Q_val = np.array([QoI(X[0, i], X[1, i], dt, T) for i in range(M)]).T
    # solve the normal equations to find the optimal coefficients for the regression
    if X.shape[1] <= M:
        # if V has full column rank calculate the reduced QR factorization of V
        # ---> reduced to solving a triangular system
        Q, R = np.linalg.qr(V)
        c = np.linalg.solve(R, Q.T @ Q_val)
    else:
        # if V doesn't have full column rank call the solver
        c = np.linalg.solve(V.T @ V, V.T @ Q_val)
    # compute the estimator
    estimate_ = c[0]
    # compute the error with respect to the reference value
    true_error = np.abs(Q_ref - estimate_)
    # compute the estimated error
    estimated_error = 1 / (M-n) * np.linalg.norm(Q_val - V @ c, 2) ** 2
    # compute the half-size of the 1-alpha confidence interval
    confidence_interval = st.norm.ppf(1-alpha/2) * np.sqrt(estimated_error / M)
    return estimate_, true_error, confidence_interval
```

All the functions used for the implementation are in the code attached above, while we refer to the notebook in the submission *InterpolationCV.ipynb* for reproducibility of the plots and of the obtained results.

References

[1] Albert Cohen and Giovanni Migliorati. "Optimal weighted least-squares methods". In: *The SMAI journal of computational mathematics* 3 (2017), pp. 181–203.

[2] Abdul-Lateef Haji-Ali et al. "Multilevel weighted least squares polynomial approximation". In: *arXiv preprint arXiv:1707.00026* (2017).

[3] Giovanni Migliorati and Fabio Nobile. "Stable high-order randomized cubature formulae in arbitrary dimension". In: *arXiv preprint arXiv:1812.07761* (2018).

[4] Yuji Nakatsukasa. "Approximate and integrate: Variance reduction in Monte Carlo integration via function approximation". In: *arXiv preprint arXiv:1806.05492* (2018).