QMC integration of non-smooth function
Application to pricing exotic functions

Benoît Müller

Stochastic Simulations: Project 5

Preparations

- Uniform into Gaussian by Box-Müller:

$$\xi_{2k,2k-1} = \sqrt{-2\log U_{2k}}(\cos,\sin)(2\pi U_{2k-1}) \quad \text{for } k \in \{1,\ldots,m/2\}$$

- Gaussian into Brownian by Gaussian increments:

$$w_{t_i} = w_{t_{i-1}} + \sqrt{t_i - t_{i-1}}\xi_i = \sum_{k=1}^{i} \sqrt{t_k - t_{k-1}}\xi_k \quad \text{for } i \in \{1,\ldots,m\}$$

- Brownian into Payoff: $S_t = S_0 \exp\left((r - \sigma^2/2)t + \sigma w_t\right)$

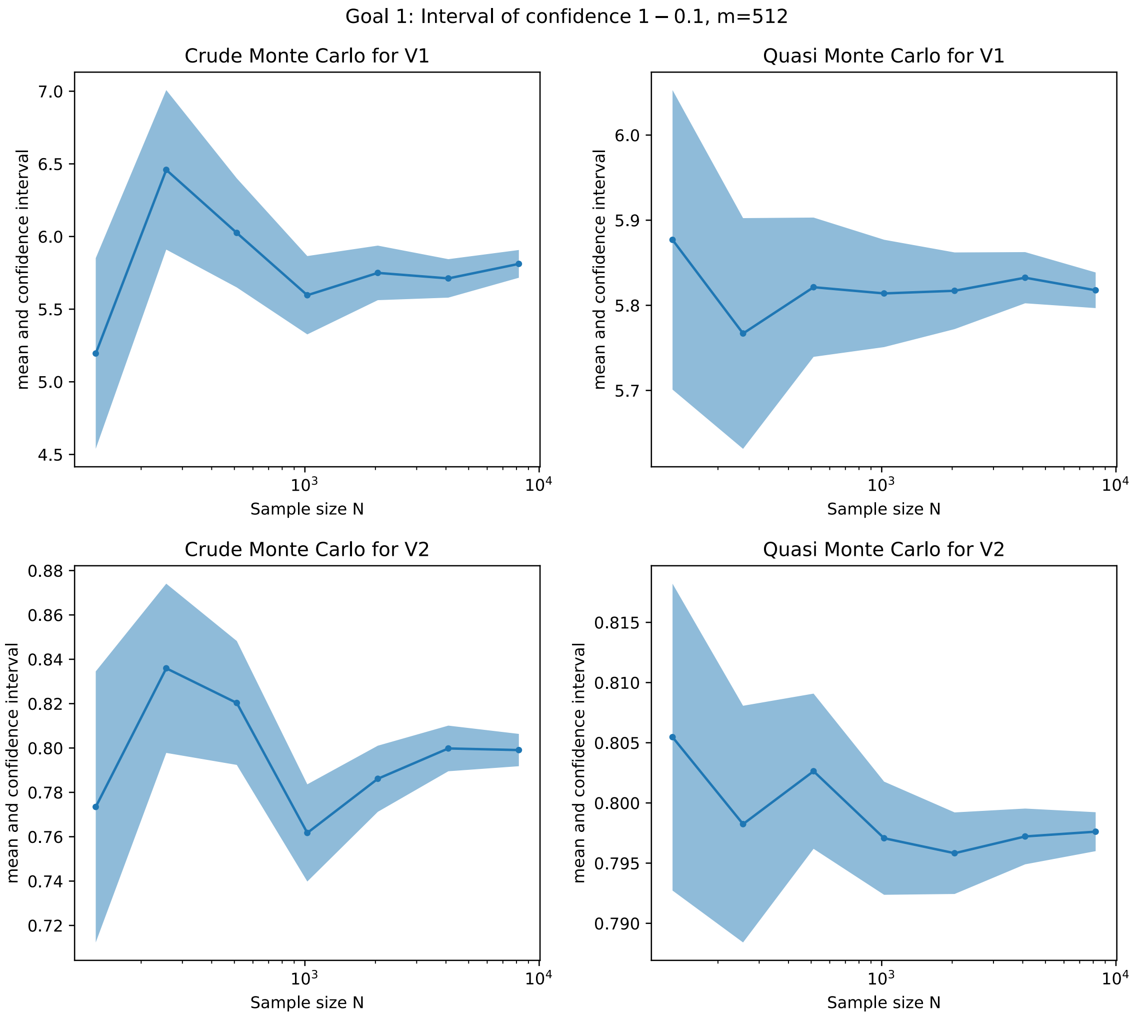
$$\phi(U) = \frac{1}{m}\sum_{i=1}^{m} S_i\left(W_i(U)\right) - K$$

Part I: Object oriented code

- « mes_stats.py » : can store a sample and compute the confidence interval
- « Payoff.py » :
  - store the fixed parametres.
  - methods to compute all the quantities and functions of the problem. Everything start from a uniform sample, generated with numpy.random .
  - In general we compute quantities related to V1 and V2 together to avoid repetitions.
  - This gives us a function « transform » that transform a uniform sample U into the Payoff.
  - (Q)MC: use a (quasi-)random uniform U, and use « transform ». For the QMC, method, we repeat the process a number of time to randomize it.

Part I: Results

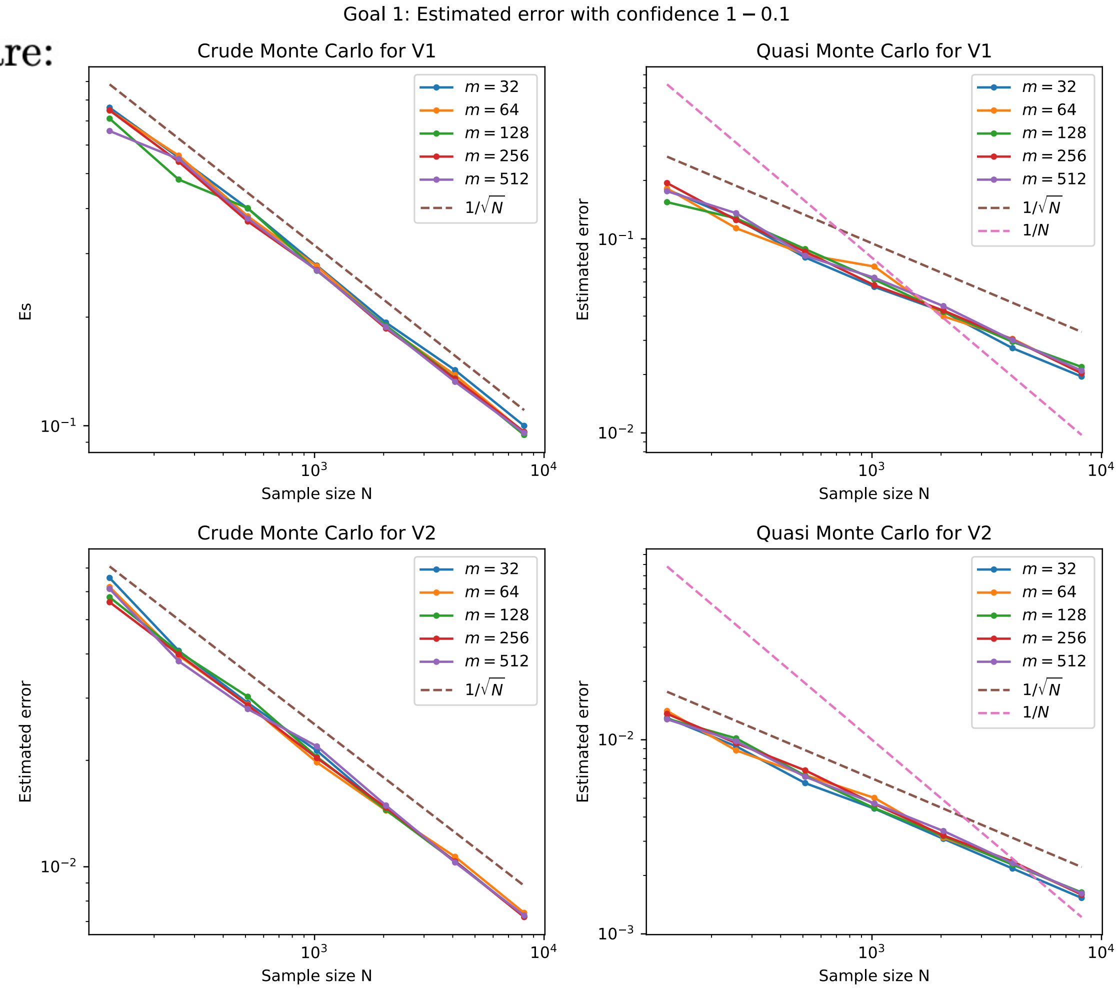
- Intervals:



Part I: Results

The computed intervals for m = 512 , N = 8192 and alpha = 0.1 are:
V1: MC: 5.811725874367571 ± 0.09548596396958434
QMC: 5.817669062629391 ± 0.02093917604003062
V2: MC: 0.799072265625 ± 0.007282353659082176
QMC: 0.797613525390625 ± 0.0016172925932021554

- Errors:



Part II: Pre-integration

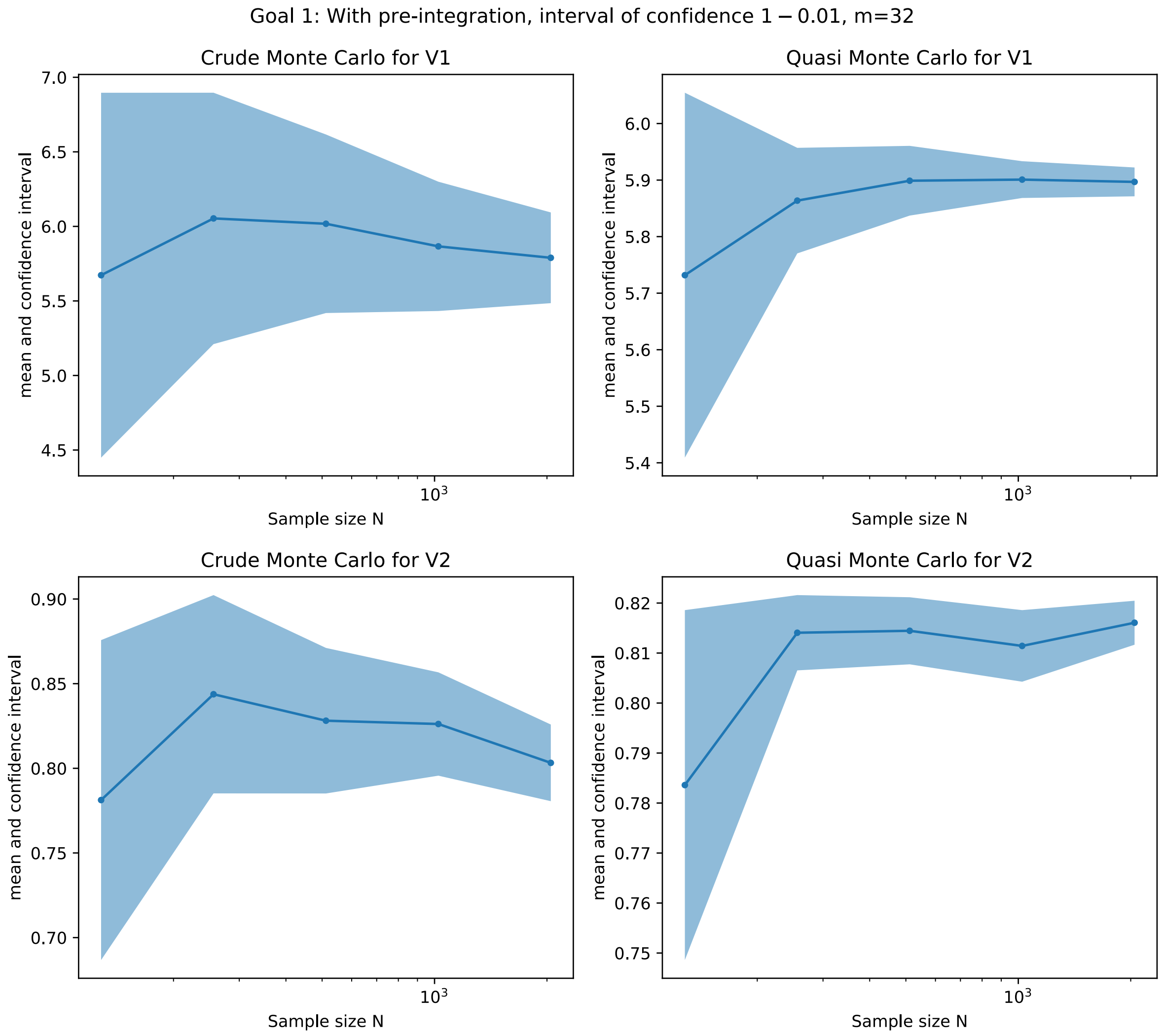
- With respect to *U_j*
- Choice of *j*: Looking at the gaussian generation, an index that appears in the radius, and not in the angle –> monotonicity

$$W_i(U) = \sum_{k=1}^{i} \sqrt{t_k - t_{k-1}} Z_k(U) = \sqrt{t_1}\sqrt{-2\log U_2}(\cos + \sin)(2\pi U_1) + \sum_{k=3}^{i} \sqrt{t_k - t_{k-1}} Z_k(U),$$

- Code:
  - « *psi* »: compute a good approximation of the support of the function we want to integrate. It uses the monotonicity and the optimisation tool « *scipy.optimize.root_scalar*».
  - « *integrate* »: Used a sample of *m-1* points and integrate for the reminding dimension. We use « *scipy.integrate.fixed_quad* » that operate a simple quadrature of fixed order.
  - «*PI(Q)MC*» Fix a sample *U* and use «integrate» for the reminding dimension

Part II: Results

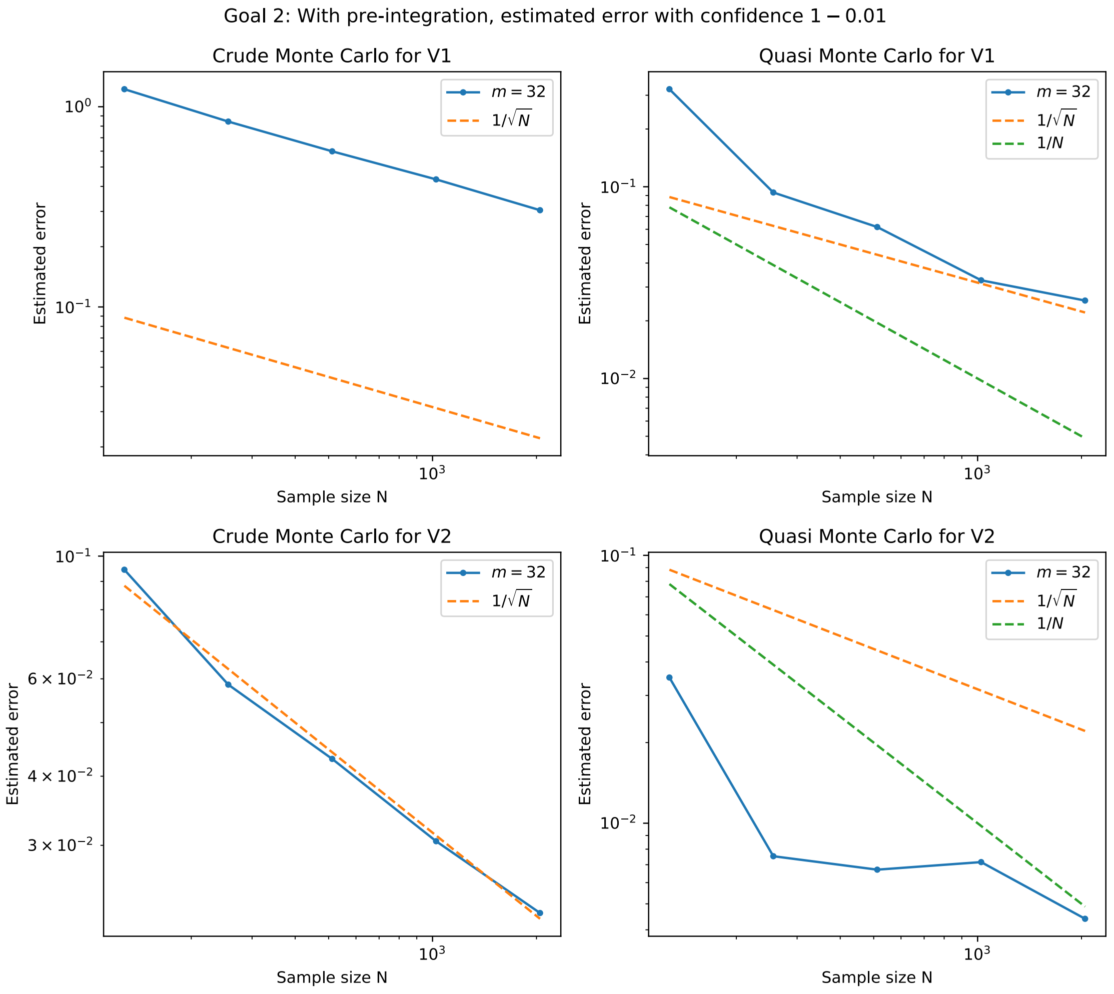
- Intervals:



Part II: Results

V1: PIMC: 5.789356690015531 ± 0.304260988271941
PIQMC: 5.896785954787449 ± 0.025505059981263036
V2: PIMC: 0.80322265625 ± 0.022634123444309658
PIQMC: 0.816064453125 ± 0.004397258072902301

- Errors:



Part III: Sample size for relative accuracy

- Use the last run
- Suppose the variance no more changing a lot
- Conclude the desired size:

$$N = \frac{c_{1-\alpha}^2 \hat{\sigma}_{\tilde{N}}^2}{\mathrm{tol}^2} = \frac{\mathrm{err}^2 \tilde{N}}{\mathrm{tol}^2} = \frac{\mathrm{err}^2 \tilde{N}}{10^{-4} \hat{\mu}_{\tilde{N}}^2}$$

- F the run with m = 512 , N = 8192 and alpha = 0.1

  It gives N=22’113 for V1

  N=6’803 for V2 (already attained)

Part III: Variance reduction technique

- Augmenting the K –> acceptance rate of the characteristic function very low –> lot of runs for very few information.
- Artificially push the distribution to the right by importance sampling.

$$\hat{\mu}_{\mathrm{IS}} = \frac{1}{N}\sum_{i=1}^{N}\psi(X_{0:m}^{(i)})w(X_{0:m}^{(i)})$$

Where $w(X_{0:m}) = \prod_{n=0}^{m-1}\frac{p(X_n, X_{n+1})}{q(X_n, X_{n+1})}$ is the likelinood ration in the importance sampling estimator

And p and q are the old and new density functions

- We can explicitely compute the ratio by the formulas of the course, and apply Algorithm 6.5(p.62) «Importance sampling for SDEs».

A Posteriori

- Do the integration in the simplest formulation: we know the pdf of the brownian motion (Gaussian process). Then turn back to uniform in $[0,1]^m$
- Vectorize the (Q)MC methods with respsect to N.
- …