Stochastic Simulation
Professor: Prof. Fabio Nobile
Assistant: Eliott Van Dieren

Week of 2025-11-03 to 2025-11-07

Lab 8 of Thursday 6th November 2025

On randomized QMC formulas

Let $P = \{X_1, \dots, X_N\}$, $X_i \in \mathbb{R}^d$, be a low-discrepancy sequence and denote the QMC quadrature by $\hat{\mu}_{QMC} = \frac{1}{N}\sum_{i=1}^{N} \psi(X_i)$. We are interested in estimating the error $|\mu - \hat{\mu}_{QMC}|$. Notice that since the points X_i are *not* i.i.d., we can't use a variance estimator or a CLT as in MC. In order to be able to do this, we can *randomize* the QMC formula. Let $U_j \overset{iid}{\sim} \mathcal{U}([0,1]^d)$, $j = 1, \dots, K$. If the set of points P is a low discrepancy point set, so is the *randomly shifted point set* $P_{U,j} := \{\{X_1 + U_j\}, \dots, \{X_N + U_j\}\}$, where $\{\cdot\}$ represents the fractional part. Moreover, since $U_j \overset{iid}{\sim} \mathcal{U}([0,1]^d)$, so is $\{X_i + U_j\}$ for any $i = 1, \dots, N$. Thus, we can apply a Monte Carlo estimator on $\hat{\mu}_{QMC}$, by computing K independent estimators $\hat{\mu}^j_{QMC}$ for each of the randomly shifted point sets $P_{U,j}$, and then averaging out the estimators. This in turn results in an unbiased estimator $\hat{\hat{\mu}}_{QMC}$ of μ, for which we can use the standard variance estimator and CLT results. C.f the lecture notes for more details.

On generating low-discrepancy sequences

Use the module `sobol_new.py` available on the course's website to generate Sobol sequences.[1] The `Python` [2] syntax `R = generate_points(N,d,0)` generates a matrix `R` of size $N \times d$ corresponding to N vectors of dimension d.

Exercice 1.

Consider the problem of approximating the integral

$$I_d(f) = \int_{[0,1]^d} f(x)\,dx\ ,$$

for some given function $f\colon [0,1]^d \to \mathbb{R}$. In this exercise we will investigate the approximation qualities of different estimators of $I_d(f)$ for various functions f, which differ mainly by their regularity. Specifically, for each function listed below address to the following points. Perform all computations at least for $d = 2$ and $d = 20$.

[1] These functions were adapted from https://paddy3118.blogspot.com/2019/11/quasi-random-sobol-sequence.html.

[2] Download the files `sobol_new.py` and `Sobol_new-joe-kuo-6.21201` from the course website and use them by writing `from sobol_new import *` at the beginning of your python script. Both files should be in the same directory

1) Implement a *crude Monte Carlo* estimator to approximate the integral $I_d(f)$.

   Estimate the error using the central limit theorem (CLT). Plot both the exact error (c.f. exact solutions below) and the CLT-based error estimate as functions of the number of used samples M, say, and estimate the convergence rate.

2) Implement a *Quasi Monte Carlo* (QMC) estimator to approximate the integral $I_d(f)$. Use the module `sobol_lib.py` available on the course's website to generate Sobol sequences.

   Estimate the error using the CLT by estimating the variance with a *randomized QMC*. Once again, plot both the exact error and estimated error based on random shifts as functions of the number of N and estimate the convergence rate.

List of functions

Investigate the approximation techniques for $I_d(f)$ mentioned above for the following functions $f\colon [0,1]^d \to \mathbb{R}$, with $x = (x_1, \dots, x_d)$. Please note that a testing suite with several of the function definitions listed below can be found here[3].

1) Oscillatory function: $f(x) = \cos\Big(2\pi w_1 + \sum_{j=1}^d c_j x_j\Big)$, with $c_j = 9/d$, $w_1 = \frac{1}{2}$.
   The exact solution is:
$$I_d(f) = \Re\left(e^{i2\pi w_1} \prod_{j=1}^{d} \frac{1}{ic_j}(e^{ic_j} - 1)\right) ,$$
   where i denotes the imaginary unit and $\Re(z)$ the real part of $z \in \mathbb{C}$.

2) Product peak: $f(x) = \prod_{j=1}^d \Big(c_j^{-2} + \big(x_j - w_j\big)^2\Big)^{-1}$, with $c_j = 7.25/d$ and $w_j = \frac{1}{2}$.
   Exact solution:
$$I_d(f) = \prod_{j=1}^{d} c_j\big(\arctan(c_j(1 - w_j)) + \arctan(c_j w_j)\big) .$$

3) Gaussian: $f(x) = \exp\Big(-\sum_{j=1}^d c_j^2 (x_j - w_j)^2\Big)$, with $c_j = 7.03/d$ and $w_j = \frac{1}{2}$.
   Exact solution:
$$I_d(f) = \prod_{j=1}^{d} \frac{\sqrt{\pi}}{2c_j}\Big(\mathrm{erf}(c_j(1 - w_j)) + \mathrm{erf}(c_j w_j)\Big) .$$

4) Continuous function: $f(x) = \exp\Big(-\sum_{j=1}^d c_j |x_j - w_j|\Big)$, with $c_j = 2.04/d$ and $w_j = \frac{1}{2}$.
   Exact solution:
$$I_d(f) = \prod_{j=1}^{d} \frac{1}{c_j}\big(2 - e^{-c_j w_j} - e^{-c_j(1-w_j)}\big) .$$

5) Discontinuous function:
$$f(x) = \begin{cases} 0 & \text{if } x_1 > w_1 \text{ or } x_2 > w_2 \\ \exp\Big(\sum_{j=1}^d c_j x_j\Big) & \text{otherwise,} \end{cases}$$

[3] https://people.math.sc.edu/Burkardt/c_src/testpack/testpack.html

with $c_j = 4.3/d$, $w_1 = \frac{\pi}{4}$, and $w_2 = \frac{\pi}{5}$.
Exact solution:

$$I_d(f) = \frac{\prod_{j=3}^{d}(e^{c_j}-1)}{\prod_{j=1}^{d} c_j}(e^{c_1 w_1}-1)(e^{c_2 w_2}-1)\ .$$

6) Volume of the simplex:

$$f(x) = \begin{cases} 1 & \text{if } \sum_{j=1}^{d} x_j \leqslant 1 \\ 0 & \text{otherwise.} \end{cases}$$

Exact solution:

$$I_d(f) = \frac{1}{d!}\ .$$

Solution

As predicted by the theory, the crude Monte Carlo method always shows a convergence of $\mathcal{O}(M^{-1/2})$ regardless of the regularity and the dimension of the integrand. In contrast, significant efficiency gains are possible using the Quasi Monte Carlo methodprovided the function f is smooth. These gains degenerate as the dimension d increases, though. Convergence plots of each function for $d = 2$ (left) and $d = 20$ (right) as a function of the number of samples M are shown in Figures 1 to 6. The error estimator for QMC and LHS is computed over $K = 20$ samples.

A Python implementation is shown below.

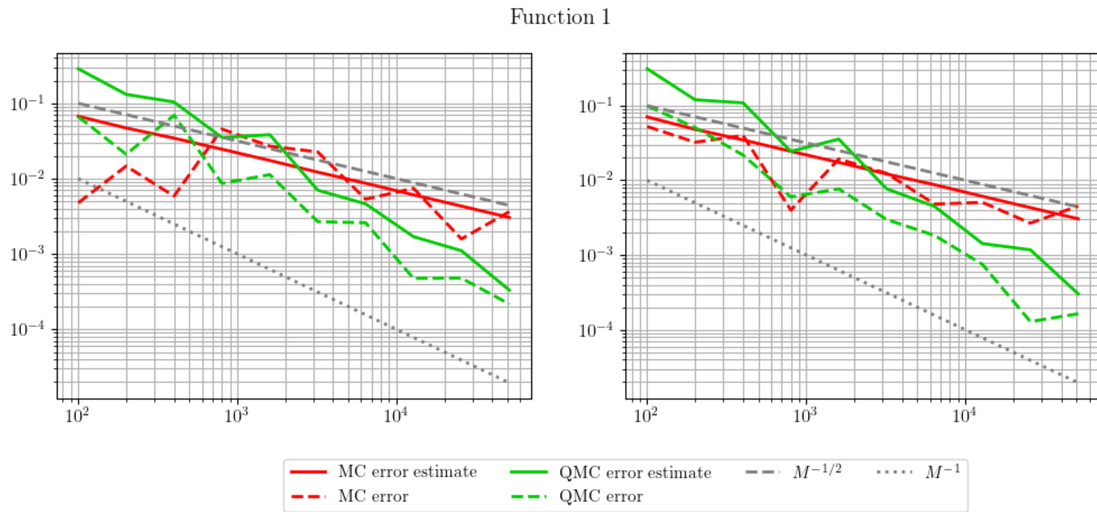

Figure 1: Function no. 1, $d = 2$ (left) and $d = 20$ (right)

Function 2

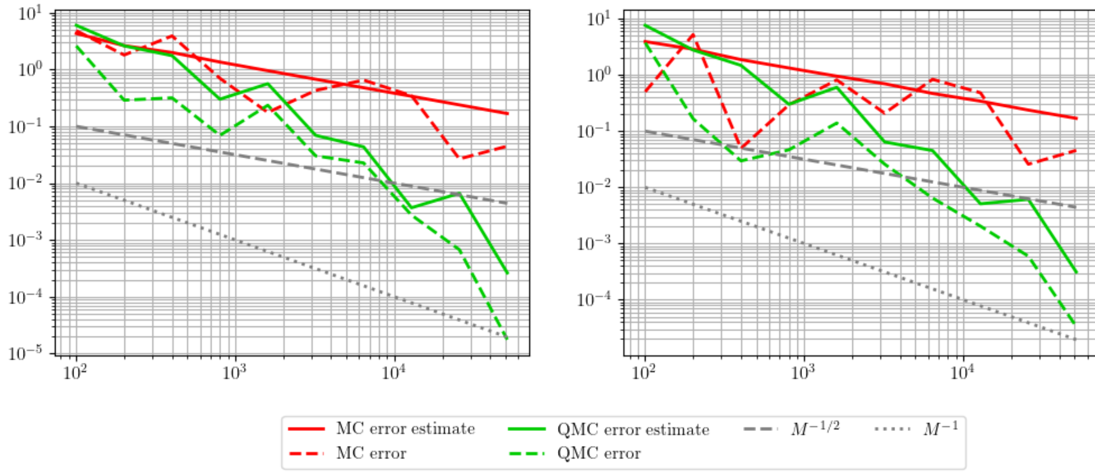

Figure 2: Function no. 2, $d = 2$ (left) and $d = 20$ (right)

Function 3

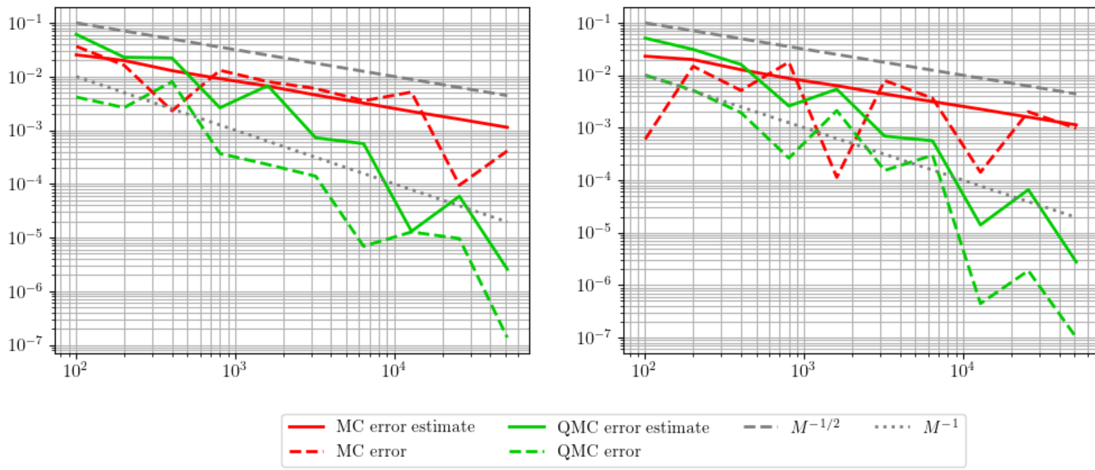

Figure 3: Function no. 3, $d = 2$ (left) and $d = 20$ (right)

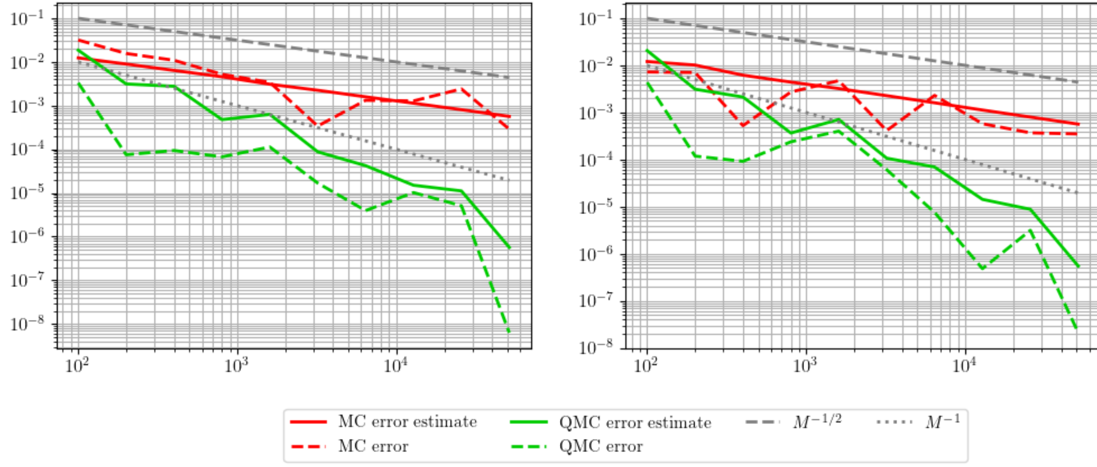

Figure 4: Function no. 4, $d = 2$ (left) and $d = 20$ (right)

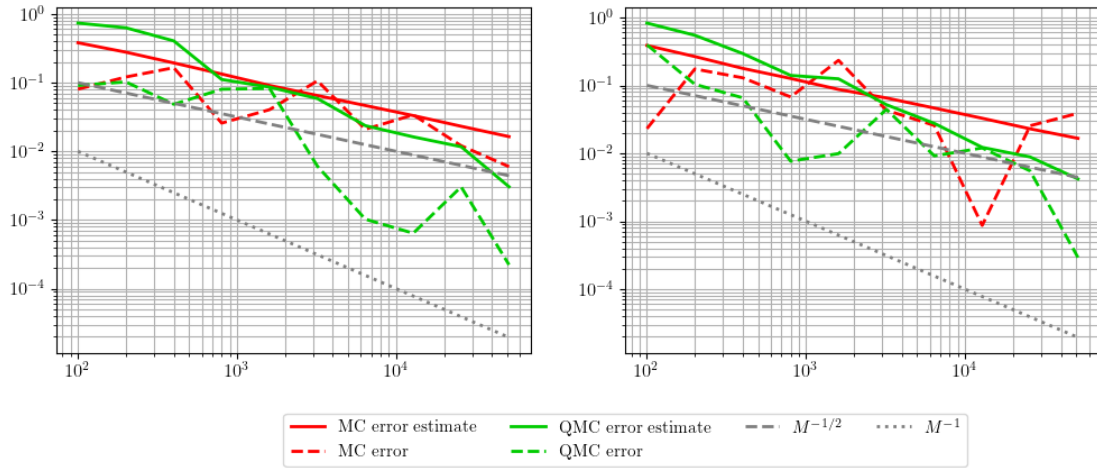

Figure 5: Function no. 5, $d = 2$ (left) and $d = 20$ (right)

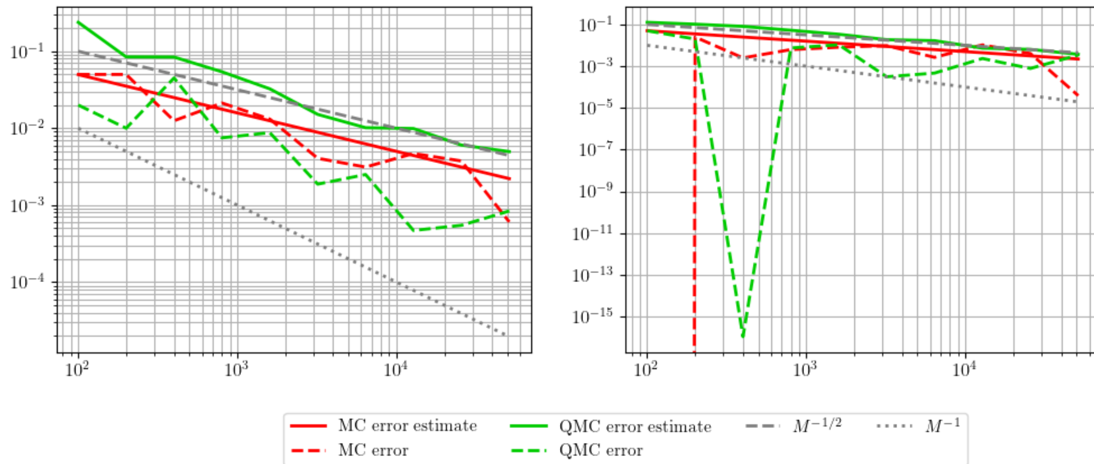

Figure 6: Function no. 6, $d = 2$ (left) and $d = 20$ (right). We remark that this is a nasty function to estimate.

Python code

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import numpy as np
import math
import matplotlib.pyplot as plt
from scipy.special import erf
import sobol_new as sn
from matplotlib import rc
from numpy import matlib
import scipy.stats as st
####################################################################
rc('font',**{'family':'serif','serif':['Computer Modern Roman'],
     'size' : '12'})
rc('text', usetex=True)
rc('lines', linewidth=2)
plt.rcParams['axes.facecolor']='w'
import matplotlib
latex_preamble = r'\usepackage{amsmath} \usepackage{amssymb}'
matplotlib.rcParams.update({
    'text.usetex': True,
    'text.latex.preamble': latex_preamble
})
####################################################################

np.random.seed(42)

def mc(typess, d,M):
    x = st.uniform.rvs( size = (d,M) )
    data, exact = evaluate(types, x)
    est = np.mean(data)
    err_est = np.std(data)/np.sqrt(M)
    err = np.abs(est-exact)
    return est, err_est, err

def qmc(types,d,N,K):
```

```
    x = sn.generate_points(int(N),int(d),0)
    x=x.T
    data = np.zeros(K)
    for i in range(K):
        dat, exact = evaluate(types, np.mod(x+matlib.repmat(
                np.random.random(size =(int(d),1)),1,int(N)),1))
        data[i] = np.mean(dat)

    est = np.mean(data)
    err_est = 3*np.std(data)/np.sqrt(K)
    # use 3 in the error estimate : P(sqrt(M)*err/std<3) \approx 0
    err = np.abs(est-exact)
    return est, err_est, err

def evaluate(types, x):
    d = x.shape[0]
    if (types == 1):
        w = 0.5
        c = 9./d
        val = np.cos(2*np.pi*w + c*np.sum(x,0))
        exact = np.real( np.exp(2*np.pi*1j*w) * ((np.exp(1j*c)-1)/(1j*c))**d )
    elif(types == 2):
        w = 0.5
        c = 7.25/d
        val = np.prod(1/(c**(-2) + (x - w)**2), 0)
        exact = (c * (np.arctan(c*(1-w)) + np.arctan(c*w)))**d
    elif(types == 3):
        w = 0.5
        c = 7.03/d
        val = np.exp(-np.sum(c**2 * (x - w)**2,0))
        exact = (np.sqrt(np.pi)*(erf(c*(1-w))+erf(c*w))/(2*c))**d
    elif(types == 4):
        w = 0.5
        c = 2.04/d
        val = np.exp(- np.sum(c*np.abs(x-w),0))
        exact = ((1/c)*(2-np.exp(-c*w)-np.exp(-c*(1-w))))**d
    elif(types == 5):
        w1 = np.pi/4
        w2 = np.pi/5
        c = 4.3/d
        val = np.exp(np.sum(c*x,0))
        val[(x[0,:]>w1) + (x[1,:]>w2)] = 0
        exact = (1/c**d) * (np.exp(c*w1)-1)*(np.exp(c*w2)-1) * \
        (np.exp(c)-1)**(d-2)
    elif(types == 6):
        val = (np.sum(x,0)<=1)
        exact = 1./math.factorial(d)

    return val, exact

for types in range(1,7):
    print(types)
    fig, axes = plt.subplots(nrows = 1, ncols = 2, figsize = (12,4))

    Mlist = np.array([50*2**i for i in range(1,11)])
    nM = np.size(Mlist)
    ds = [2,20]
    for ax in axes:
        d=ds[0]
        K = 20

        # Run the Monte Carlo
        mc_err_est = np.zeros(nM)
```

```
    mc_err = np.zeros(nM)
    for i in range(nM):
        val, mc_err_est[i], mc_err[i] = mc(types, d, Mlist[i])

    # Run the QMC
    qmc_err_est = np.zeros(nM)
    qmc_err = np.zeros(nM)
    for i in range(nM):
        dummy, qmc_err_est[i], qmc_err[i] = qmc(types, d, Mlist[i]/K, K)

    ax.loglog(Mlist, mc_err_est, color='red', label = 'MC error estimate')
    ax.loglog(Mlist, mc_err,'--', color='red', label = 'MC error')
    ax.loglog(Mlist, qmc_err_est, color=[0.0,0.8,0.0], label = 'QMC error estimate')
    ax.loglog(Mlist, qmc_err, '--', color=[0.0,0.8,0.0], label = 'QMC error')
    ax.loglog(Mlist, Mlist**-0.5, '--', label = r'$M^{-1/2}$',color='gray')
    ax.loglog(Mlist, Mlist**-1.0, ':', label = r'$M^{-1}$',color='gray')

    ax.grid(True,which='both')
    plt.suptitle(r'Function ' +str(types))

    d=ds[1]
ax.legend(fontsize = 11,loc='upper center',
          bbox_to_anchor=(0.0000001, -0.15), shadow=False, ncol=4)

plt.show()
```

Exercice 2.

Consider the random boundary value problem (BVP)

$$\begin{cases} \big(a(x,\omega)u'(x,\omega)\big)' = 0 \,, & \text{in } (0,L) \,, \\ u(0,\cdot) = 0 \,, & \\ a(L,\cdot)u'(L,\cdot) = 1 \,, & \end{cases}$$

where ω represents an elementary random event, so that $a \equiv a(x,\omega)$ is a random field. The BVP is a simplified model for a linear beam of length L, which is fixed on one side ($x = 0$) and free on the other at which a unit load is applied. Here, the random field a models the beam's spatially varying uncertain material properties. We are interested in quantifying the resulting uncertainty on the beam's displacement at the free end-point. Specifically, we are interested in studying the expected value of the random variable

$$Z \equiv Z(\omega) := u(L,\omega) = \int_0^L \frac{1}{a(x,\omega)}\, dx \,.$$

However, Z is usually not computable for a general elasticity coefficient a. Instead, we consider the computable, approximate random quantity of interest Z_I, which is obtained by approximating

the integral by the midpoint rule on a uniform grid,

$$Z_I \equiv Z_I(\omega) := h \sum_{i=1}^{I-1} \frac{1}{a(x_i + \frac{h}{2}, \omega)} ,$$

with $x_i = ih$, $i = 0, \ldots, I \in \mathbb{N}$, and $h = L/I$.

We are interested in approximating $\mathbb{E}[Z_I]$ for $L = 1$ for two different elasticity coefficients:

(i) the random field a is given by

$$a_1(x, \omega) = \mu + \frac{\sigma}{\pi^2} \sum_{n=1}^{d} \frac{\cos(\pi n x)}{n^2} Y_n(\omega) , \quad Y_n(\omega) \sim U(-1, 1) \text{ i.i.d.},$$

where $\mu = 1$ and $\sigma = 4$,

(ii) let $a_2(x, \omega) = \exp(\kappa(x, \omega))$, where

$$\kappa(x, \omega) = x + \sqrt{2} \sum_{n=1}^{d} \frac{\sin((n - \frac{1}{2})\pi x)}{(n - \frac{1}{2})\pi} Y_n(\omega) , \quad Y_n(\omega) \sim \mathcal{N}(0, 1) \text{ i.i.d.}$$

To that end, approximate $\mathbb{E}[Z_I]$ for various values of d and for various sub-divisions I using (a) a crude Monte Carlo method, (b) Quasi Monte Carlo (QMC) sampling. Use randomized QMC to estimate the error and provide asymptotic confidence intervals.

Solution

An implementation of this problem is attached. For convenience in case (ii) we transform the Gaussian variables $\{Y_n(\omega)\}$ into uniform ones using the inverse transformation. In both cases (i) and (ii) we build a LHS sample on the unit hypercube $[0, 1]^d$. The LHS error is computed by averaging over a small number K of random repetitions. For the QMC case, the QMC error is computed using the randomized QMC approach presented in the exercise description.

The results are shown in figures 7-9 for different values of d, namely $d = 2, 16, 32$. As we can see, for the selected values of d, LHS and QMC sampling both provide a considerable advantage over pure Monte Carlo sampling.

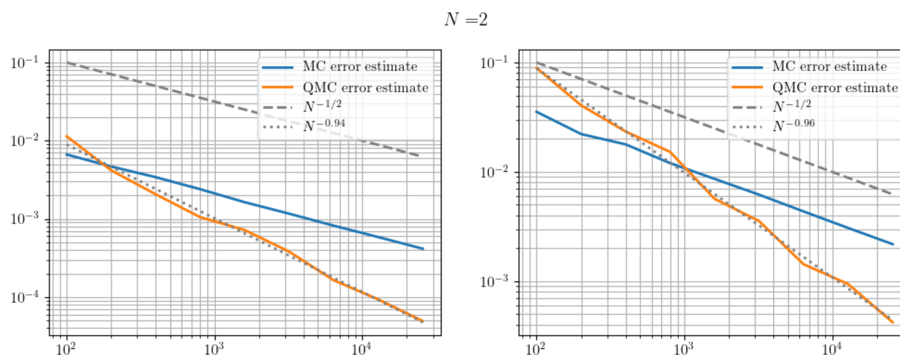

Figure 7: Error vs M for $d = 2$. Left column: $\mathbb{E}[Z(a_1)]$. Right column: $\mathbb{E}[Z(a_2)]$

N =16

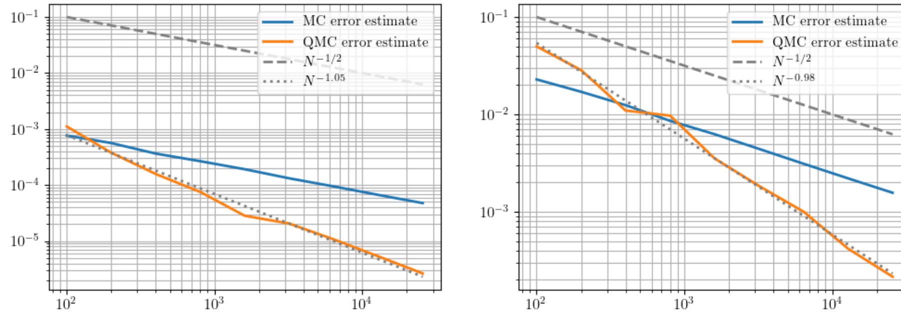

Figure 8: Error vs M for $d = 16$. Left column: $\mathbb{E}[Z(a_1)]$. Right column: $\mathbb{E}[Z(a_2)]$

N =32

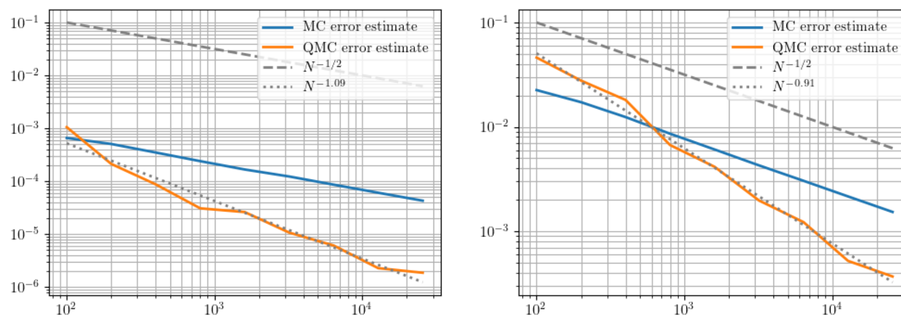

Figure 9: Error vs M for $d = 64$. Left column: $\mathbb{E}[Z(a_1)]$. Right column: $\mathbb{E}[Z(a_2)]$

Python code

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt
from matplotlib import rc
from pdb import set_trace
import sobol_new as sn
from numpy import matlib
# ----------------------------------------------
rc('font',**{'family':'serif','serif':['Computer Modern Roman'],
     'size' : '12'})
rc('text', usetex=True)
rc('lines', linewidth=2)
plt.rcParams['axes.facecolor']='w'
import matplotlib
latex_preamble = r'\usepackage{amsmath} \usepackage{amssymb}'
matplotlib.rcParams.update({
    'text.usetex': True,
    'text.latex.preamble': latex_preamble
})
#----------------------------------------------
```

```python
np.random.seed(42)

#Defines which function to evaluate

def alpha_1(x,w,mu=1.0,sigma=2.0):
    N=len(w)
    a=0
    #transforms to the region of interest [-1,1]
    w=-1+2*w
    for n in range(1,N+1):
       a=a+np.cos(n*np.pi*x)*w[n-1]/(n**2.0)
    a=mu+(sigma**2.0/np.pi**2.0)*a
    return a

def alpha_2(x,w,mu=1,sigma=4):
    N=len(w)
    a=0
    #transforms to uniform rvs
    w=norm.ppf(w)
    for n in range(1,N+1):
       a=a+np.sin((n-0.5)*np.pi*x)*w[n-1]/((n-0.5)*np.pi)
    a=x+(2**0.5)*a
    return np.exp(a)

def evaluate(types,w,L=1):
    I=w.shape[0]
    x=np.linspace(0,L,I)
    h=x[1]-x[0]
    z=np.zeros(w.shape[1])
    for j in range(w.shape[1]):
        if types==1:
            alpha=alpha_1(x,w[:,j])
        elif types==2:
            alpha=alpha_2(x,w[:,j])
        else:
            raise "type not defined. must be 1 or 2"
        z[j]=2*h*np.sum(1./(1+alpha))
        if np.isnan(z[j])==1:
            set_trace()
    return z

#------------------------------------------------------------------
#
#defines wheather to use Monte Carlo or QMC
#
#------------------------------------------------------------------
def mc(typess, d,M):
    x = np.random.random( size = (d,M) );
    data= evaluate(types, x);
    est = np.mean(data);
    err_est = np.std(data)/np.sqrt(M);
    return est, err_est

def  qmc(types,d,N,K):
    x = sn.generate_points(int(N),int(d),0)
    x=x.T
    data = np.zeros(K)
    for i in range(K):
        dat = evaluate(types, np.mod(x+matlib.repmat(
                np.random.random(size =(int(d),1)),1,int(N)),1))
        data[i] = np.mean(dat);
```

```
    est = np.mean(data);
    err_est = 3*np.std(data)/np.sqrt(K); # use 3 in the error estimate : P(sqrt(M)*err/std<3)
     ↪  approx 0
    return est, err_est

# Computes integrals via MC and QMC for an N dimensional random field
NN=[2,4,16,32]
nN=len(NN)

final_est_mc=np.zeros((nN,2))

final_er_mc=np.zeros((nN,2))

for N in NN:
    print(N)
    fig, axes = plt.subplots(nrows = 1, ncols = 2, figsize = (12,4))

    types=1
    for ax in axes:
        Mlist = 50*2**np.arange(1,10);
        nM = np.size(Mlist);
        mc_err_est = np.zeros(nM);
        qmc_err_est = np.zeros(nM);

        est_mc = np.zeros(nM);
        est_qmc = np.zeros(nM);
        K = 20;
        for i in range(nM):
            est_mc[i], mc_err_est[i] = mc(types, N, Mlist[i]);
            est_qmc[i], qmc_err_est[i] = qmc(types, N, Mlist[i]/K, K);

        print('N='+str(N))
        print('type ' +str(types))
        print('mean MC ' ,est_mc[i])
        print('mean QMC ' ,est_qmc[i])

        print('error MC ' ,mc_err_est[i])
        print('error QMC ' ,qmc_err_est[i])
        print('------------------')

        ax.loglog(Mlist, mc_err_est, '-', label = 'MC error estimate')
        ax.loglog(Mlist, qmc_err_est, '-', label = 'QMC error estimate')

        #fits line to QMC

        xx=np.log(Mlist)
        yy=np.log(qmc_err_est)


        aa,bb=np.polyfit(xx,yy,deg=1)

        ax.loglog(Mlist, Mlist**-0.5, '--', label = r'$N^{-1/2}$',color='gray')
        ax.loglog(np.exp(xx), np.exp(aa*xx+bb), ':', label =
         ↪  r'$N^{'+str(round(aa,2))+'}$',color='gray')

        ax.grid(True,which='both')
        ax.legend(fontsize = 11,loc='upper right')

        plt.suptitle(r'$N=$' +str(N))
        types+=1
    plt.show()
```