1. In this case, we even have an analytical expression for the partial derivatives of u:

$$\partial_{\boldsymbol{y}}^{\boldsymbol{\nu}} u(x,\boldsymbol{y}) = (-1)^{|\boldsymbol{\nu}|}|\boldsymbol{\nu}|! \int_0^x \left(\int_w^1 f(z)\,\mathrm{d}z \right) \frac{1}{a(w,\boldsymbol{y})^{|\boldsymbol{\nu}|+1}} \prod_{j=1}^{s} \psi_j(w)^{\nu_j}\,\mathrm{d}w.$$

The parametric regularity bound is an immediate consequence of the above identity.

2. Note that $|\frac{\partial^{|\mathfrak{u}|}}{\partial \boldsymbol{y}_{\mathfrak{u}}} F(\boldsymbol{y})| \le \frac{\|f\|_{L^2(D)}}{a_{\min}} |\mathfrak{u}|! \prod_{j\in\mathfrak{u}} b_j$. By direct computation:

$$\begin{aligned}
\|F\|_{s,\boldsymbol{\gamma}}^2 &= \sum_{\mathfrak{u}\subseteq\{1,\dots,s\}} \frac{1}{\gamma_{\mathfrak{u}}} \int_{[0,1]^{|\mathfrak{u}|}} \left(\int_{[0,1]^{s-|\mathfrak{u}|}} \frac{\partial^{|\mathfrak{u}|}}{\partial \boldsymbol{y}_{\mathfrak{u}}} F(\boldsymbol{y})\,\mathrm{d}\boldsymbol{y}_{-\mathfrak{u}} \right)^2 \mathrm{d}\boldsymbol{y}_{\mathfrak{u}} \\
&\overset{\substack{\text{Cauchy–}\\ \text{Schwarz}}}{\le} \sum_{\mathfrak{u}\subseteq\{1,\dots,s\}} \frac{1}{\gamma_{\mathfrak{u}}} \int_{[0,1]^s} \left| \frac{\partial^{|\mathfrak{u}|}}{\partial \boldsymbol{y}_{\mathfrak{u}}} F(\boldsymbol{y}) \right|^2 \mathrm{d}\boldsymbol{y} \\
&\le \frac{\|f\|_{L^2(D)}^2}{a_{\min}^2} \sum_{\mathfrak{u}\subseteq\{1,\dots,s\}} \frac{(|\mathfrak{u}|!)^2 \prod_{j\in\mathfrak{u}} b_j^2}{\gamma_{\mathfrak{u}}}.
\end{aligned} \tag{1}$$

3. This is a special case of the following result:

Lemma. *Let (α_i) and (β_i) be sequences of positive real numbers. The expression*

$$g(\boldsymbol{\gamma}) := \left(\sum_i \alpha_i \gamma_i^{\lambda} \right)^{\frac{1}{\lambda}} \left(\sum_i \beta_i \gamma_i^{-1} \right)$$

is minimized by $\gamma_i = c\big(\frac{\beta_i}{\alpha_i}\big)^{\frac{1}{1+\lambda}}$ for arbitrary $c > 0$.

Proof. Let us find out when the gradient vanishes:

$$\begin{aligned}
0 = \partial_j g(\boldsymbol{\gamma}) = \frac{1}{\lambda} \left(\sum_i \alpha_i \gamma_i^{\lambda} \right)^{1/\lambda - 1} \lambda \alpha_j \gamma_j^{\lambda-1} \left(\sum_i \beta_i \gamma_i^{-1} \right) \\
- \beta_j \gamma_j^{-2} \left(\sum_i \alpha_i \gamma_i^{\lambda} \right)^{1/\lambda}.
\end{aligned}$$

After some trivial simplifications, we can see that this is equivalent to

$$\gamma_j^{\lambda+1} = \frac{\beta_j}{\alpha_j} \frac{\sum_i \alpha_i \gamma_i^{\lambda}}{\sum_i \beta_i \gamma_i^{-1}}.$$

Furthermore, this condition is satisfied if

$$\gamma_j = c \left(\frac{\beta_j}{\alpha_j} \right)^{1/(1+\lambda)},$$

where $c > 0$ is arbitrary. □

We apply the above result in the special case $\alpha_{\mathfrak{u}} = \left(\frac{2\zeta(2\lambda)}{(2\pi^2)^\lambda}\right)^{|\mathfrak{u}|}$ and $\beta_{\mathfrak{u}} = (|\mathfrak{u}|!)^2 \prod_{j\in\mathfrak{u}} b_j^2$ and choose $c = 1$ to follow the convention that $\gamma_\varnothing = 1$.

In complete analogy to the lecture notes, the constant in the QMC error bound can be bounded independently of the dimension s provided that the diffusion coefficient $a(x, \boldsymbol{y})$ is obtained as a truncation of the infinite series

$$a_\infty(x, \boldsymbol{y}) = a_0(x) + \sum_{j=1}^{\infty} y_j \psi_j(x), \quad x \in D,\ \boldsymbol{y} \in [-1/2, 1/2]^{\mathbb{N}},$$

where $a_0 \in L^\infty(D)$, $\psi_j \in L^\infty(D)$ for all $j \geq 1$, $(\|\psi_j\|_{L^\infty(D)})_{j\geq 1} \in \ell^p(\mathbb{N})$ for some $p \in (0, 1)$, and there exist constants $a_{\min}, a_{\max} > 0$ such that $0 < a_{\min} \leq a_\infty(x, \boldsymbol{y}) \leq a_{\max} < \infty$ for all $x \in D$ and $\boldsymbol{y} \in [-1/2, 1/2]^{\mathbb{N}}$. In particular, by choosing the POD weights

$$\gamma_{\mathfrak{u}} = \left(|\mathfrak{u}|! \prod_{j\in\mathfrak{u}} \frac{b_j}{\sqrt{2\zeta(2\lambda)/(2\pi^2)^\lambda}}\right)^{\frac{2}{1+\lambda}}, \quad \lambda = \begin{cases} \frac{p}{2-p} & \text{if } p \in (2/3, 1), \\ \frac{1}{2-2\delta} \text{ for arbitrary } \delta \in (0, 1/2) & \text{if } p \in (0, 2/3], \end{cases} \tag{2}$$

the QMC approximation for the expected value of the ODE problem satisfies

$$\text{R.M.S. error} \lesssim \varphi(n)^{-\min\{1/p-1/2, 1-\delta\}}, \tag{3}$$

where the implied coefficient is independent of the dimension s.

4. (a) It is clear that the parametric regularity bounds for both the continuous solution $u(s, \boldsymbol{y})$ and the discretized solution $\boldsymbol{u}(\boldsymbol{y})$ are proportional to the parametric regularity bounds of $\frac{1}{a(x,\boldsymbol{y})}$. Thus, the discretized solution $\boldsymbol{u}(\boldsymbol{y}) = [u_1(\boldsymbol{y}), \ldots, u_{100}(\boldsymbol{y})]^{\mathrm{T}} \in \mathbb{R}^{100}$ satisfies

$$|\partial_{\boldsymbol{y}}^{\boldsymbol{\nu}} u_k(\boldsymbol{y})| \lesssim |\boldsymbol{\nu}|! \boldsymbol{b}^{\boldsymbol{\nu}},$$

where $\boldsymbol{b} = (b_j)_{j=1}^s$ with $b_j = \frac{\|\psi_j\|_{L^\infty(D)}}{a_{\min}}$. In consequence, the quantity of interest $F(\boldsymbol{y}) = u_k(\boldsymbol{y} - \frac{\mathbf{1}}{\mathbf{2}})$, $\boldsymbol{y} \in [0, 1]^s$, satisfies the bound (1). It follows that using the weights (2) as inputs to the CBC algorithm, we obtain a QMC rule also for the spatially discretized problem with dimension-independent QMC convergence rate (3) under the assumption that a is obtained as a truncation of an infinite series a_∞ with suitably fast decaying fluctuations $(\|\psi_j\|_{L^\infty(D)})_{j\geq 1} \in \ell^p(\mathbb{N})$, $p \in (0, 1)$.

(b) Please see the script `qmcdemo.py`.

```
import numpy as np
import matplotlib.pyplot as plt
from sympy.ntheory import primefactors, isprime
from scipy.special import zeta
from joblib import Parallel, delayed

def generator(n):
    # For prime n, find the primitive root modulo n
    if not isprime(n):
        raise ValueError('n must be prime')
    factorlist = primefactors(n-1)
    g = 2; i = 1
    while i <= len(factorlist):
        if pow(g,int((n-1)/factorlist[i-1]),n) == 1:
            g = g+1; i = 0
        i = i+1
    return g

def fastcbc(s,n,Gammaratio,gamma):
    # Fast CBC construction with POD weights
    m = int((n-1)/2)

    # Rader permutation
    g = generator(n)
    perm = np.ones(m,dtype=int)
    for j in range(1,m):
        perm[j] = np.mod(perm[j-1]*g,n)
    perm = np.minimum(perm,n-perm)-1
    permflip = np.flip(perm)

    # Precompute the FFT of the first column (permuted indices)
    bernoulli = lambda x: x**2-x+1/6
    fftomega = np.fft.fft(bernoulli(np.mod((perm+1)*(perm[-1]+1)/n,1)))
    z = np.zeros(s)
    p = np.zeros((s,m))

    for d in range(s):
        pold = np.vstack((np.ones((1,m)),p))
        x = gamma[d]*Gammaratio @ pold[:-1,:]
        if d == 0:
            minind = 1
        else:
            tmp = np.real(np.fft.ifft(fftomega * np.fft.fft(x[permflip])))
            minind = perm[np.argmin(tmp)]+1
        z[d] = minind
        omega = bernoulli(np.mod(minind*np.arange(1,m+1)/n,1))
        for l in range(d+1):
```

```
            p[l,:] = pold[l+1,:] + omega * pold[l,:] * Gammaratio[l] * gamma[d]
    return z

if __name__ == '__main__':
    # Discretize the ODE
    h = .01 # mesh size
    x = np.arange(0,1+h,h) # mesh
    ncoord = len(x)
    G = np.tril(np.ones(ncoord))
    G = G - .5*np.eye(ncoord)
    G[:,0] = .5
    X,_ = np.meshgrid(1-x,1-x)
    G = h*X*G
    G = G[50,:] # pick the row corresponding to coordinate x = 0.5

    # Specify the parametric diffusion coefficient
    s = 100 # stochastic dimension
    decay = 2 # decay rate of stochastic fluctuations
    indices = np.arange(1,s+1)
    # Precompute the deterministic part
    deterministic = np.sin(np.pi*np.outer(x,indices))/indices**decay
    a = lambda y: 1 + deterministic @ y # diffusion coefficient

    # ODE solution
    u = lambda y: G @ (1/a(y))

    # Weight structure
    amin = 1-zeta(decay)/2
    b = np.arange(1,s+1,dtype=float)**(-decay)/amin
    delta = .05
    lam = 1/(2-2*delta)
    Gammaratio = np.arange(1,s+1)**(2/(1+lam))
    gamma = (b/np.sqrt(2*zeta(2*lam)/(2*np.pi**2)**lam))**(2/(1+lam))

    # Solve the expected value of the quantity of interest
    nlist = [17,31,67,127,257,503,1009,2003,4001,8009,16007,32003,64007]
    R = 16 # number of random shifts
    rms = [] # store the computed RMS errors
    with Parallel(n_jobs=-1) as parallel:
        for n in nlist:
            print('n = ' + str(n))
            z = fastcbc(s,n,Gammaratio,gamma) # find generating vector
            shift = np.random.uniform(0,1,s)
            results = []
            for r in range(R):
                shift = np.random.uniform(0,1,s) # random shift
                tmp = parallel(delayed(u)(np.mod(i*z/n+shift,1)-1/2) for i in range(n))
```

```
            # Compute the QMC mean for each random shift
            results.append(np.mean(tmp))
        qmcavg = np.mean(results) # average over the random shifts
        rmserror = np.linalg.norm(qmcavg-results)/np.sqrt(R*(R-1)) # RMS errors
        rms.append(rmserror) # save the RMS errors for each n

# Least squares fit for the error
A = np.ones((len(nlist),2))
nscaled = [R*i for i in nlist]
A[:,1] = np.log(nscaled)
lsq = np.linalg.solve(A.T @ A, A.T @ np.log(rms))
lsq[0] = np.exp(lsq[0])

# Visualize the results
fig, ax = plt.subplots()
ax.loglog(nscaled,lsq[0]*nscaled**lsq[1],'--b',linewidth=2,label='slope: '
+ str(lsq[1]))
ax.loglog(nscaled,rms,'.r','QMC error')
ax.set_title('QMC error ($s = ' + str(s) + '$)',fontsize=15)
ax.set_xlabel('number of nodes $nR$',fontsize=13)
ax.set_ylabel('R.M.S. error',fontsize=13)
ax.legend()
plt.show()
```

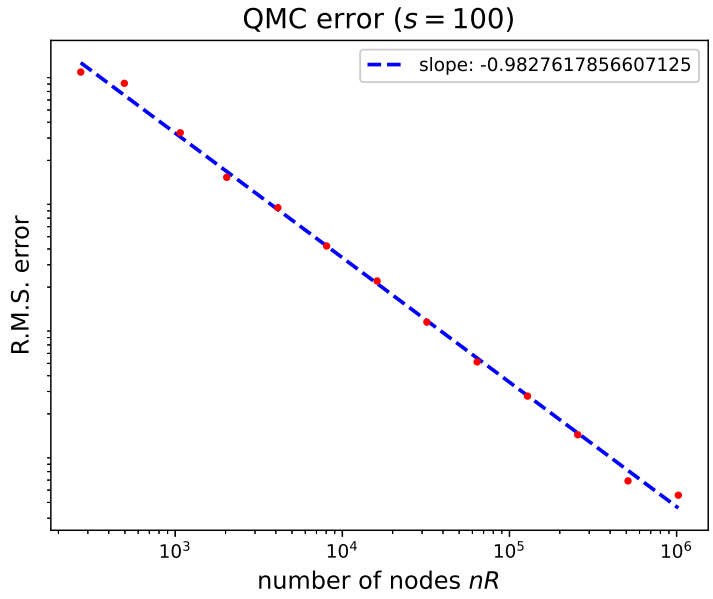

Figure 1: Output of the script `qmcdemo.py` corresponding to a 100-dimensional affine and uniform input random field for QMC point sets and $R = 16$ random shifts obtained using the CBC algorithm constructed for the POD weights (2).