

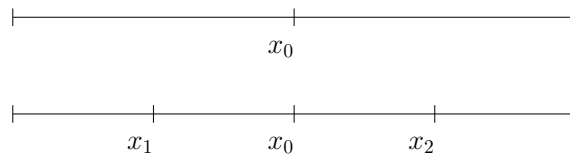
Numerical Finance Reading Course

Sheet 4 - Sample Solution

Exercise 1: Smolyak Algorithm

Solution:

We will use the following numbering of grid points for the one-dimensional nested grid:



The one-dimensional quadrature formulas in both dimensions are then given by

$$Q^{(1)} = f(x_0),$$

$$Q^{(2)} = \frac{2}{3}f(x_1) - \frac{1}{3}f(x_0) + \frac{2}{3}f(x_2)$$

and hence

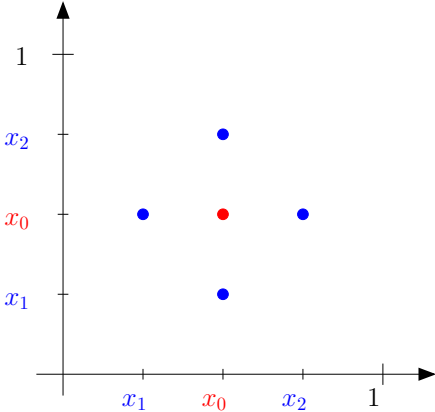
$$\Delta^{(0)} = Q^{(1)} - Q^{(0)} = f(x_0),$$

$$\Delta^{(1)} = Q^{(2)} - Q^{(1)} = \frac{2}{3}f(x_1) - \frac{4}{3}f(x_0) + \frac{2}{3}f(x_2).$$

Then the Smolyak quadrature formula yields

$$\begin{aligned} Q(1, 2) &= \sum_{|i| \leq 1} \Delta^{(i_1)} \otimes \Delta^{(i_2)} \\ &= \Delta^{(0)} \otimes \Delta^{(0)} + \Delta^{(1)} \otimes \Delta^{(0)} + \Delta^{(0)} \otimes \Delta^{(1)} \\ &= f(x_0, x_0) + \left(\frac{2}{3}f(x_1, x_0) - \frac{4}{3}f(x_0, x_0) + \frac{2}{3}f(x_2, x_0) \right) \\ &\quad + \left(\frac{2}{3}f(x_0, x_1) - \frac{4}{3}f(x_0, x_0) + \frac{2}{3}f(x_0, x_2) \right) \\ &= -\frac{5}{3}f(x_0, x_0) + \frac{2}{3}(f(x_1, x_0) + f(x_2, x_0) + f(x_0, x_1) + f(x_0, x_2)). \end{aligned}$$

The corresponding grid is



Exercise 2: Comparison Monte Carlo & Quasi-Monte Carlo

Compute the integral

$$I_3[f] = \int_{[0,1]^3} x_1^2 x_2^2 x_3^2 dx_1 dx_2 dx_3,$$

using

- Monte Carlo integration,
- Quasi-Monte Carlo integration (e.g. with the Halton sequence).

Visually compare both methods by plotting their integration errors and their theoretical convergence rates.

Solution:

```
#include <iostream>
#include <fstream>
#include <cmath>

using namespace std;

const double primes[50] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
37, 41, 43, 47, 53, 59, 61, 67, 71,
73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151,
157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229};

/*
 * Linear Congruential Generator
 */
class LCG {
private:
    long long int a;
    long long int b;
    long long int M;

    long long int state;

public:
    LCG( long long int A, long long int B, long long int prime, int seed):
        a(A), b(B), M(prime), state((long long)seed) {};

    double uni(){
        state = (a* state + b) % M;
        return (double)state / (double)M;
    }
};

/*
 * Computes the radical Inverse function of integer n to base b.
 */
double radInv(int n, int b){

    double z = 0;
    int d;
    int denom = 1;
    do{
        d = n % b;
        n = n / b;
        z = b*z + d;
    }
```

```

        denom = denom * b;
    }while(n > 0);

    z = (double)z / (double)denom;
    return z;
}

int main(){

    /*
     * UNIX RAND Generator
     */
    long long int a = 1103515245LL;
    long long int b = 12345LL;
    long long int M = 2147483648LL;
    long long int seed = 1;
    LCG urand(a, b, M, seed);

    // Analytical Solution
    double sol = 1./27.;

    // Integrate for different N to obtain convergence rate
    for(int N = 2; N <= (1<<20); N *= 2){
        double MCintegral = 0;
        double QMCintegral = 0;
        for(int j = 0; j < N; j++){
            // Compute the integrand for both methods
            double MCprod = 1;
            double QMCprod = 1;
            for(int k = 0; k < 3; k++){
                // MC: random numbers
                double u = urand.uni();
                MCprod *= u*u;
                // QMC: Halton sequence based on first 3 prime numbers
                double h = radInv(j, primes[k]);
                QMCprod *= h*h;
            }
            // Add integrand
            MCintegral += MCprod;
            QMCintegral += QMCprod;
        }

        // Write sample size and (absolute) errors to standard out
        cout << N << " " << abs(MCintegral/(double)N - sol) << " "
            << abs(QMCintegral/(double)N - sol) << endl;
    }

    return 0;
}

```

