```cpp
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <vector>
#include <random>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>
#include <format>

double F(double x, int d);
double monteCarloEstimate(int M, int d, std :: mt19937 gen);
double Importance_Sampling(int M, int d,std ::  mt19937 gen);
double Rejection_Sampling( int M, int d, std :: mt19937 gen);
double stad(std::vector<double> x);
double mean(std::vector<double> x);

int main()
{
    int itera[] = {1, 10, 100, 1'000, 10'000, 100'000, 1'000'000};
    float dims[] = {1, 2, 4, 8, 16};

    int M_max = 7;
    int d_max = 5;
    int nrows = M_max; //M
    int ncols = d_max;//d

    double err_MC[ncols][nrows];
    double err_IS[ncols][nrows];
    double err_RS[ncols][nrows];

    std :: vector <double> estimates_MC (100);
    std :: vector <double> estimates_IS (100);
    std :: vector <double> estimates_RS (100);

    for (int d = 3; d < d_max ; d++){
        for (int M = 0; M < M_max ; M++){
            std:: cout << "d: "<< dims[d] << " M is : "<<itera[M]<< "\n";
            std:: cout << "\n";

            for (int i= 0; i < 100; i++){
                int seed = rand();
                std :: mt19937 gen(seed);
                estimates_MC[i] = monteCarloEstimate(itera[M], dims[d], gen);
                estimates_IS[i] = Importance_Sampling(itera[M], dims[d], gen);
                estimates_RS[i] =  Rejection_Sampling(itera[M], dims[d], gen);

            }
            err_MC[M][d] = stad(estimates_MC);
            err_IS[M][d] = stad(estimates_IS);
            err_RS[M][d] = stad(estimates_RS);
        }
    }
    std:: cout << "Matrix of errors for MC is : "<<  "\n";
    for (int i = 0; i < nrows; i++)
        for (int j = 0; j < ncols; j++)
            std :: cout << err_MC[i][j] << ",\n"[j == ncols-1];

    std :: cout << '\n';
    std :: cout << '\n';

    std:: cout << "Matrix of errors for IS is : "<< "\n";
    for (int i = 0; i < nrows; i++)
        for (int j = 0; j < ncols; j++)
            std :: cout << err_IS[i][j] << ",\n"[j == ncols-1];

    std :: cout << '\n';
    std :: cout << '\n';

    std:: cout << "Matrix of errors for RS is : "<< "\n";
    for (int i = 0; i < nrows; i++)
        for (int j = 0; j < ncols; j++)
            std :: cout << err_RS[i][j] << ",\n"[j == ncols-1];
        return 0;
}


double mean(std::vector<double> x){
    double sum;
    for (int i = 0; i< x.size(); i++){
        sum += x[i];
    }
    return sum/x.size();
}

double stad(std::vector<double> x){
    double average ;
    average = mean(x);
    double sum;
    for (int i = 0; i< x.size(); i++){
        sum += pow(fabs(average - x[i]), 2)/x.size();
    }
    return sqrt(sum);
}

double F(std :: vector <double> x , int d)
{
    double sum = 0;
    for (int i = 0; i < d ; i++){
        sum += pow(x[i], 2);
    }
    return pow(2, -d)*sum;
}

float g(std :: vector <double> x, int d )
{
    double s = 1;
    for (int i = 0; i < d ; i++){
        if (x[i] < -1 ){
            return 0;
        }
        else if (x[i] >1  ){
            return 0;
        }
    }
    s = F(x, d);
    return s;
};

float normal_pdf(float x, float s)
{
    static const float inv_sqrt_2pi = 0.3989422804014327;
    float a = x/ s;
    return inv_sqrt_2pi / s * std::exp(-0.5f * a * a);
};

float multiply(std :: vector <double> x, int d )
{
    double multi = 1;
    for (int i = 0; i < d ; i++){
        multi *= normal_pdf(x[i], 0.5);
    }
    return multi;
};


double monteCarloEstimate(int M, int d, std :: mt19937 gen)
//Function to execute Monte Carlo integration on predefined function
{

        double totalSum = 0;
        double randNum, functionVal;
        int iter = 0;
        std :: vector <double> x_ (d);
    std :: uniform_real_distribution< double> distrib(-1,1);
        while (iter<M)
        {
                // generate random vector
                for (int i = 0; i < d ; i++){
            x_[i] = distrib(gen);
                }
                totalSum += F(x_, d);
                iter++;
        }

    double estimate = totalSum * pow(2, d)/(M);
        return estimate;
}

double Importance_Sampling(int M, int d,std ::  mt19937 gen)
//Function to execute Monte Carlo integration on predefined function
{
    double totalSum = 0;
    double randNum, functionVal;
    int iter = 0;
    std :: vector <double> x_ (d);
    std :: normal_distribution<double> distrib(0.,0.5);
    while (iter<M)
    {
        // generate random vector
        for (int i = 0; i < d ; i++){
            x_[i] = distrib(gen);
        }
        totalSum += g(x_, d) / multiply (x_, d);
        iter++;
    }

    double estimate = totalSum / M;
    return estimate;
}

double Rejection_Sampling(int M, int d, std :: mt19937 gen){
    std::random_device rd;
    std::uniform_real_distribution<> dis_x(-1, 1);
    double estimate;

    std::uniform_real_distribution<> dis_y(0, 1);
    double gamma = 1.5;
    double SUM1 =0. ; double SUM2 = 0.;
    int c = 0;


    for(int m = 0; m < M; ++m){
        std::vector<double> x(d);
        while (true){
            for(int i = 0; i < d; ++i) {
                x[i] = dis_x(gen);
            }
            double y; double q; double w;
            y = dis_y(gen);
            if (y < gamma * (0.1 + F(x, d))){
                q = 0.1 + F(x, d);
                w = 1/q;
                SUM1 += w * g(x, d);
                SUM2 += w;
                c += 1;
                break;
            }
        }
    }
    estimate = pow(2,d) * SUM1/SUM2;
    return estimate;
}
```