

Завдання 1

Розв'язуємо стохастичне диференціальне рівняння

$$dX_t = (\mu_1 X_t + \mu_2)dt + (\sigma_1 X_t + \sigma_2)dW_t.$$

Покладемо $S_t = e^{at+bW_t}$ і застосуємо формулу Іто для добутку $X_t S_t$ (точніше, багатовимірну формулу Іто для $g = xe^{at+by}$ та $Y_t = (X_t, W_t)$) з урахуванням формули для dX_t

$$\begin{aligned} dX_t &= (\mu_1 X_t + \mu_2)dt + (\sigma_1 X_t + \sigma_2)dW_t, \\ dX_t S_t &= \left(a + \frac{b^2}{2}\right)X_t S_t dt + S_t dX_t + bX_t S_t dW_t + bS_t(\sigma_1 X_t + \sigma_2)dt = \\ &= S_t \left(\left(\left(a + \frac{b^2+2b\sigma_1}{2} + \mu_1\right)X_t + \mu_2 + b\sigma_2 \right) dt + ((b + \sigma_1)X_t + \sigma_2)dW_t \right), \end{aligned}$$

тож, якщо покласти $b = -\sigma_1$, $a = -\mu_1 + \frac{\sigma_1^2}{2}$, то деякі доданки скоротяться і отримаємо

$$\begin{aligned} dX_t e^{-(\mu_1 - \frac{\sigma_1^2}{2})t - \sigma_1 W_t} &= e^{-(\mu_1 - \frac{\sigma_1^2}{2})t - \sigma_1 W_t} ((\mu_2 - \sigma_1 \sigma_2)dt + \sigma_2 dW_t), \\ X_t &= e^{(\mu_1 - \frac{\sigma_1^2}{2})t + \sigma_1 W_t} \left(X_0 + \int_0^t e^{-(\mu_1 - \frac{\sigma_1^2}{2})s - \sigma_1 W_s} (\mu_2 - \sigma_1 \sigma_2) ds + \int_0^t \sigma_2 e^{-(\mu_1 - \frac{\sigma_1^2}{2})s - \sigma_1 W_s} dW_s \right). \end{aligned}$$

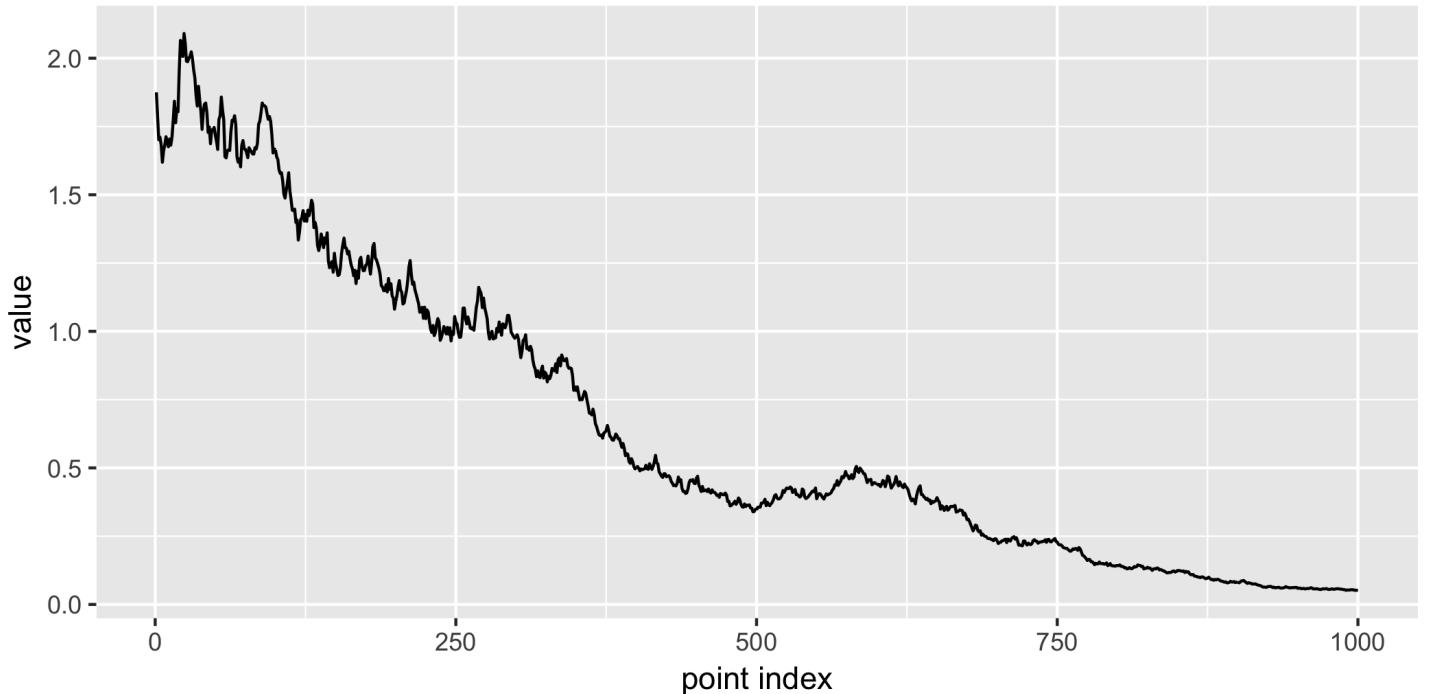
Врахуємо умови варіанту і отримаємо

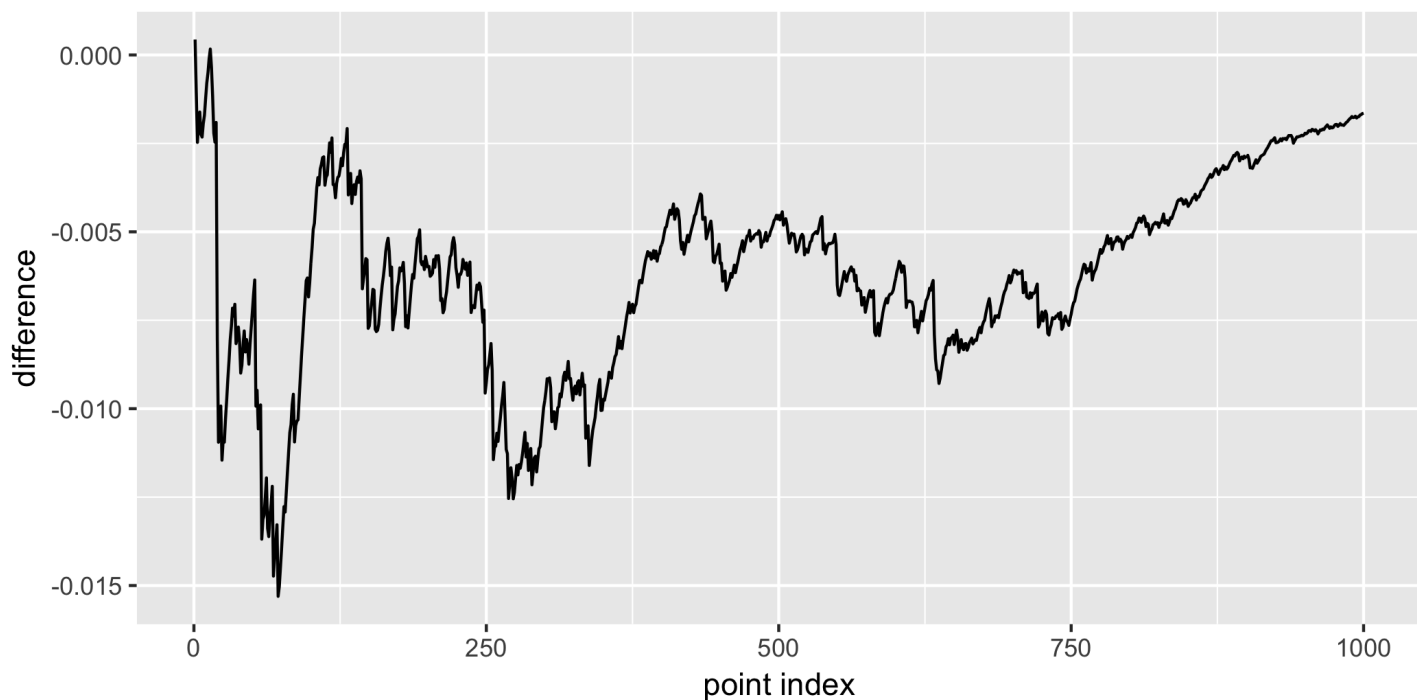
$$X_t = 1.9e^{(-3.5 - \frac{0.8^2}{2})t + 0.8W_t},$$

а наближення Ейлера будуватимуться за правилом

$$X_{t+h} = -3.5X_t h + 0.8X_t(W_{t+h} - W_t), \quad X_0 = 1.9.$$

Просимулюємо наближений та точний розв'язок для різних діаметрів розбиття та часів зупинки T ,





		Діаметр розбиття		
		$\frac{1}{10}$	$\frac{1}{100}$	$\frac{1}{1000}$
T	1	Середнє: 0.22128 Медіана: 0.19125 σ : 0.1247206	Середнє: 0.035534 Медіана: 0.031779 σ : 0.0169989	Середнє: 0.010780 Медіана: 0.009660 σ : 0.005104184
	10	Середнє: 0.21758 Медіана: 0.19064 σ : 0.1220253	Середнє: 0.035676 Медіана: 0.032213 σ : 0.01706393	Середнє: 0.010599 Медіана: 0.009526 σ : 0.004947747
	50	Середнє: 0.21972 Медіана: 0.19166 σ : 0.1221377	Середнє: 0.035943 Медіана: 0.032150 σ : 0.01710764	Середнє: 0.010635 Медіана: 0.009568 σ : 0.004958791

З таблиці можна побачити, що відхилення зменшуються зі зменшенням діаметру розбиття. Це особливо помітно для $T = 1$, а для інших його значень варто згадати, що, наприклад, $EX_{50} = e^{(3.5 - \frac{0.8^2}{2})50} = e^{159} > 10^{69}$, у порівнянні з чим відхилення не неймовірно великі. Обчислення виконані за допомогою такого коду

main

```

1 #include<iostream>
2 #include<fstream>
3 #include<string>
4 #include<chrono>
5
6 #include "simulator.cu"
7 #include "RandomProcess.h"
8

```

```

9  int main()
10 {
11     Params par;
12     int trajectoriesCount, pointsPerUnit, unitCount;
13     double start;
14     std::cout << "Enter_parameters_mu0,_mu1,_sigma0,_sigma1,_starting_value,_number_of_points_
        per_unit,_number_of_units_and_number_of_trajectories_(in_given_order)\n";
15     std::cin >> par.mu0 >> par.mu1 >> par.sigma0 >> par.sigma1 >> start >> pointsPerUnit >>
        unitCount >> trajectoriesCount;
16     std::string fileName;
17     std::cout << "Enter_output_file_name\n";
18     std::cin >> fileName;
19
20     long long startTime = std::chrono::duration_cast<std::chrono::milliseconds>
        (std::chrono::steady_clock::now().time_since_epoch()).count();
21
22     RandomProcess *process = new RandomProcess(1 / static_cast<double>(pointsPerUnit), par,
        start, pointsPerUnit * unitCount);
23     std::vector<double> results = SimulateProcess<RandomProcess, double>(process,
        trajectoriesCount);
24     std::ofstream outFile (fileName);
25     for (int i = 0; i < results.size(); i++)
26     {
27         outFile << results[i] << "\n";
28     }
29     outFile.close();
30
31     std::cout << "elapsed_" << std::chrono::duration_cast<std::chrono::milliseconds>
        (std::chrono::steady_clock::now().time_since_epoch()).count() - startTime << "_
        milliseconds\n";
32     delete process;
33 };

```

RandomProcess header

```

1  #ifndef __RandomProcess__
2  #define __RandomProcess__
3
4  #include <cuda_runtime.h>
5
6  struct Params{
7      double mu0, mu1, sigma0, sigma1;
8  };
9
10 class RandomProcess
11 {
12     private:
13         double SDE, addend, multiplier, maxDifference, stepLength;

```

```

14         int maxSteps;
15         Params par;
16     public:
17         __host__ __device__ RandomProcess(double stepLength, Params par, double start, int
            maxSteps);
18         __host__ __device__ int Next(double increment);
19         __host__ __device__ double GetValue();
20 };
21
22 #endif

```

RandomProcess

```

1  #include "RandomProcess.h"
2
3  __host__ __device__
4  RandomProcess::RandomProcess(double stepLength, Params par, double start, int maxSteps)
5  : stepLength(stepLength), par(par), SDE(start), addend(start), multiplier(1), maxSteps(maxSteps),
    maxDifference(0) {};
6
7  __host__ __device__
8  int RandomProcess::Next(double increment)
9  {
10     SDE += (par.mu0 + par.mu1 * SDE) * stepLength + (par.sigma0 + par.sigma1 * SDE) *
        increment * sqrt(stepLength);
11     addend += (par.mu0 - par.sigma0 * par.sigma1) / multiplier * stepLength + par.sigma0 /
        multiplier * increment * sqrt(stepLength);
12     multiplier *= exp((par.mu1 - pow(par.sigma1, 2) / 2) * stepLength + par.sigma1 * increment *
        sqrt(stepLength));
13     maxDifference = abs(SDE - addend * multiplier) > maxDifference ? abs(SDE - addend *
        multiplier) : maxDifference;
14     return --maxSteps > 0;
15 };
16
17 __host__ __device__
18 double RandomProcess::GetValue()
19 {
20     return maxDifference;
21 };

```

simulator

```

1  #include <vector>
2
3  #include <cuda_runtime.h>
4  #include <curand_kernel.h>
5
6  #define ThreadsPerBlock 512

```

```

7
8 template<typename TProcess, typename TResult>
9 __global__
10 void kernel_SimulateProcess(TProcess * process, TResult * results)
11 {
12     TProcess local = *process;
13     int id = threadIdx.x + blockIdx.x * blockDim.x;
14     curandState state;
15     curand_init(static_cast<unsigned long long>(clock()), id, 0, &state);
16     while(local.Next(curand_normal_double(&state))) {};
17     results[id] = local.GetValue();
18 };
19
20 template<typename TProcess, typename TResult>
21 std::vector<TResult> SimulateProcess(TProcess * process, int trajectoriesCount)
22 {
23     if (trajectoriesCount % ThreadsPerBlock != 0)
24     {
25         std::cout << "Trajectories_count_not_multiple_of_ThreadsPerBlock(" << ThreadsPerBlock
26             << "),_so_increasing_it.\n";
27         trajectoriesCount = (trajectoriesCount / ThreadsPerBlock + 1) * ThreadsPerBlock;
28     }
29
30     TProcess *devProcess;
31     cudaMalloc(reinterpret_cast<void **>(&devProcess), sizeof(TProcess));
32     cudaMemcpy(devProcess, process, sizeof(TProcess), cudaMemcpyHostToDevice);
33
34     std::vector<TResult> hostResult = std::vector<TResult>(trajectoriesCount);
35     TResult *devResult;
36     cudaMalloc(reinterpret_cast<void **>(&devResult), trajectoriesCount * sizeof(TResult));
37
38     kernel_SimulateProcess<<<trajectoriesCount / ThreadsPerBlock,
39         ThreadsPerBlock>>>(devProcess, devResult);
40     cudaMemcpy(&hostResult.front(), devResult, trajectoriesCount * sizeof(TResult),
41         cudaMemcpyDeviceToHost);
42     cudaError_t code = cudaGetLastError();
43     if (code != cudaSuccess)
44     {
45         std::cerr << cudaGetErrorString(code) << "\n";
46     }
47
48     cudaFree(devProcess);
49     cudaFree(devResult);
50
51     return hostResult;
52 };

```

Завдання 2

Розглядаємо стохастичне диференціальне рівняння

$$dX_t = \theta 2 \frac{X_t}{1+X_t^2} dt + (1_3 \sin X_t) dW_t, \quad X_0 = 2,$$

де $\theta = 1$. Побудуємо для нього наближення Ейлера та оцінемо параметр за допомогою оцінки найбільшої вірогідності для різних діаметрів розбиття та довжин відрізка T . Отримуємо такий результат

		Діаметр розбиття		
		$\frac{1}{10}$	$\frac{1}{100}$	$\frac{1}{1000}$
T	10	Середнє: 1.0072 Медіана: 1.0007 σ : 0.074	Середнє: 1.0019 Медіана: 1.0002 σ : 0.042	Середнє: 1 Медіана: 1 σ : 0.016
	20	Середнє: 1.0023 Медіана: 1.0001 σ : 0.037	Середнє: 1.0010 Медіана: 1 σ : 0.029	Середнє: 1.0001 Медіана: 1 σ : 0.01
	50	Середнє: 1.0006 Медіана: 1.0001 σ : 0.015	Середнє: 1 Медіана: 1 σ : 0.018	Середнє: 1 Медіана: 1 σ : 0.006

Симуляцію здійснено за допомогою тієї ж програми, у яку внесено зміни, а саме замінено *RandomProcess* на *ParamProcess*

ParamProcess header

```

1 #ifndef __ParamProcess__
2 #define __ParamProcess__
3
4 #include <cuda_runtime.h>
5
6 class ParamProcess
7 {
8     private:
9         double SDE, theta, stepLength, maxSteps, numerator, denominator;
10    public:
11        __host__ __device__ ParamProcess(double start, double theta, double stepLength,
12            double maxSteps);
13        __host__ __device__ int Next(double increment);
14        __host__ __device__ double GetValue();
15    };
16 #endif

```

ParamProcess

```

1 #include "ParamProcess.h"
2
3 __host__ __device__
4 ParamProcess::ParamProcess(double start, double theta, double stepLength, double maxSteps)

```

```

5      : SDE(start), theta(theta), stepLength(stepLength), maxSteps(maxSteps) {};
6
7  __host__ __device__
8  int ParamProcess::Next(double increment)
9  {
10     double defPart = 2 * SDE / (1 + pow(SDE, 2));
11     double randomPart = (1 - 3 * sin(SDE));
12     double processIncrement = theta * defPart * stepLength + randomPart * increment *
        sqrt(stepLength);
13     numerator += defPart / pow(randomPart, 2) * processIncrement;
14     denominator += pow(defPart / randomPart, 2) * stepLength;
15     SDE += processIncrement;
16     return --maxSteps > 0;
17 };
18
19 double ParamProcess::GetValue()
20 {
21     return numerator / denominator;
22 };

```

Рівняння має єдиний розв'язок, адже параметри (функції) ліпшицеві (бо диференційовні) та лінійного зросту (бо ще й неперервно диференційовні).