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 Ito для добутку $X_t S_t$ (точніше, багатовимірну формулу Ito для $g = xe^{at+by}$ та $Y_t = (X_t, W_t)$) з урахуванням формули для dX_t

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

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

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

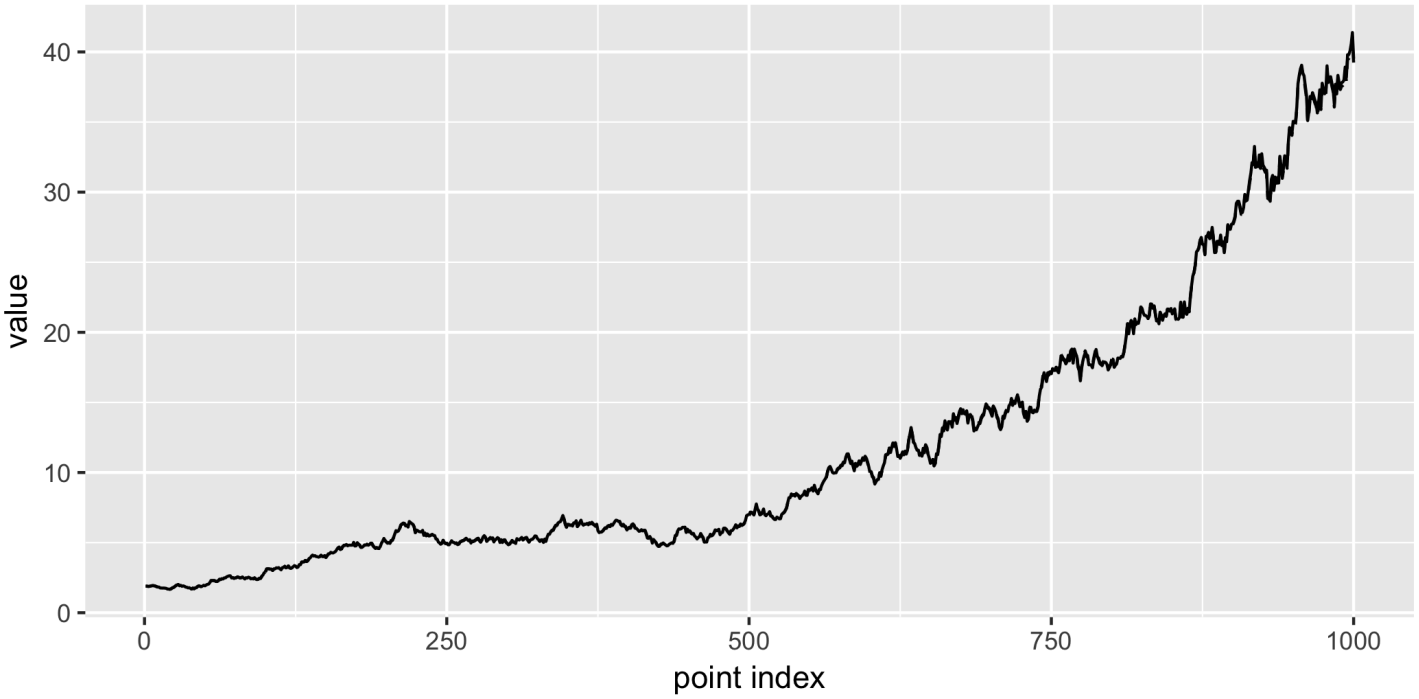
Врахуємо умови варіанту і отримаємо

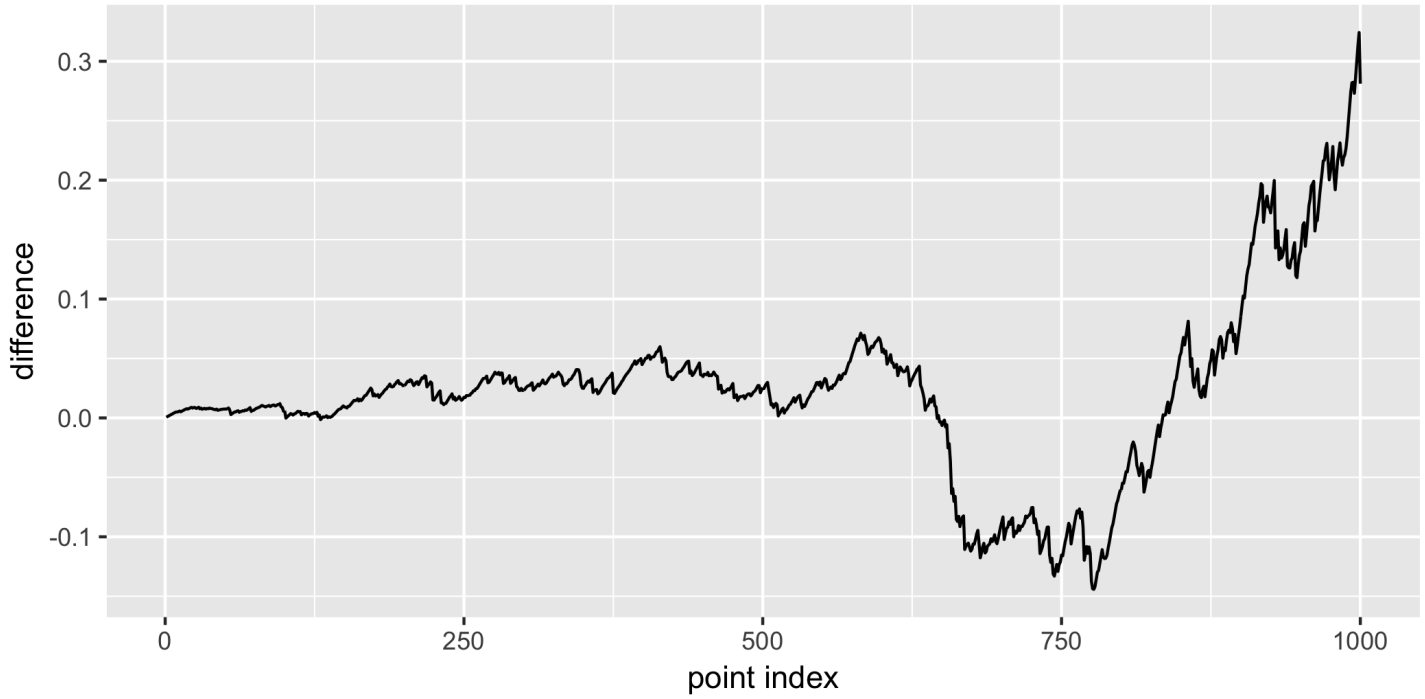
$$X_t = 1.9e^{3.5t + 0.8W_t},$$

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: 24.5 Медіана: 13.6 σ:35.1	Середнє: 4.6 Медіана: 2.05 σ:7.9	Середнє: 0.95 Медіана: 0.55 σ:1.37
	10	Середнє: $3.4 \cdot 10^{15}$ Медіана: $1.19 \cdot 10^{14}$ σ: $3.8 \cdot 10^{16}$	Середнє: $1.4 \cdot 10^{15}$ Медіана: $4.15 \cdot 10^{13}$ σ: $1.72 \cdot 10^{16}$	Середнє: $2.05 \cdot 10^{14}$ Медіана: $5.34 \cdot 10^{12}$ σ: $3.05 \cdot 10^{15}$
	50	Середнє: $8.3 \cdot 10^{77}$ Медіана: $2.19 \cdot 10^{69}$ σ: $8.3 \cdot 10^{79}$	Середнє: $9.3 \cdot 10^{74}$ Медіана: $1.9 \cdot 10^{69}$ σ: $3.77 \cdot 10^{76}$	Середнє: $1.2 \cdot 10^{74}$ Медіана: $3.76 \cdot 10^{68}$ σ: $5.29 \cdot 10^{75}$

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

```
#include<iostream>
#include<fstream>
#include<string>
#include<chrono>

#include "simulator.cu"
#include "RandomProcess.h"

```

```
int main()
{
    Params par;
    int trajectoriesCount, pointsPerUnit, unitCount;
    double start;
    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";
    std::cin >> par.mu0 >> par.mu1 >> par.sigma0 >> par.sigma1 >> start >> pointsPerUnit >> unitCount >> trajectoriesCount;
    std::string fileName;
    std::cout << "Enter output file name\n";
    std::cin >> fileName;

    long long startTime = std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono::steady_clock::now().time_since_epoch()).count();

    RandomProcess *process = new RandomProcess(1 / static_cast<double>(pointsPerUnit), par, start, pointsPerUnit * unitCount);
    std::vector<double> results = SimulateProcess<RandomProcess, double>(process, trajectoriesCount);
    std::ofstream outFile (fileName);
    for (int i = 0; i < results.size(); i++)
    {
        outFile << results[i] << "\n";
    }
    outFile.close();

    std::cout << "elapsed " << std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono::steady_clock::now().time_since_epoch()).count() - startTime << " milliseconds\n";
    delete process;
};
```

RandomProcess header

```
#ifndef __RandomProcess__
#define __RandomProcess__

#include <cuda_runtime.h>

struct Params{
    double mu0, mu1, sigma0, sigma1;
};

class RandomProcess
{
    private:
        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
               << "), so increasing it.\n";
26         trajectoriesCount = (trajectoriesCount / ThreadsPerBlock + 1) * ThreadsPerBlock;
27     }
28
29     TProcess *devProcess;
30     cudaMalloc(reinterpret_cast<void **>(&devProcess), sizeof(TProcess));
31     cudaMemcpy(devProcess, process, sizeof(TProcess), cudaMemcpyHostToDevice);
32
33     std::vector<TResult> hostResult = std::vector<TResult>(trajectoriesCount);
34     TResult *devResult;
35     cudaMalloc(reinterpret_cast<void **>(&devResult), trajectoriesCount * sizeof(TResult));
36
37     kernel_SimulateProcess<<<trajectoriesCount / ThreadsPerBlock,
           ThreadsPerBlock>>>(devProcess, devResult);
38     cudaMemcpy(&hostResult.front(), devResult, trajectoriesCount * sizeof(TResult),
           cudaMemcpyDeviceToHost);
39     cudaError_t code = cudaGetLastError();
40     if (code != cudaSuccess)
41     {
42         std::cerr << cudaGetErrorString(code) << "\n";
43     }
44
45     cudaFree(devProcess);
46     cudaFree(devResult);
47
48     return hostResult;
49 };
```

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 домопогою 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

```
#ifndef __ParamProcess__
#define __ParamProcess__

#include <cuda_runtime.h>

class ParamProcess
{
    private:
        double SDE, theta, stepLength, maxSteps, numerator, denominator;
    public:
        __host__ __device__ ParamProcess(double start, double theta, double stepLength,
            double maxSteps);
        __host__ __device__ int Next(double increment);
        __host__ __device__ double GetValue();
};

#endif
```

ParamProcess

```
#include "ParamProcess.h"

__host__ __device__
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).