

# SP Mini Project

Ivik Lau Dalgas Hostrup  
*ihostr16@student.aau.dk*

June 4, 2023

## Contents

<b>1</b>	<b>Setup</b>	<b>2</b>
<b>2</b>	<b>Presenting the results</b>	<b>2</b>
2.1	Pretty printing . . . . .	2
2.1.1	Circadian reactions . . . . .	2
2.1.2	SEIHR covid 19 reactions . . . . .	2
2.1.3	Simple reaction . . . . .	3
2.2	Simulation results . . . . .	3
2.2.1	Simple reaction . . . . .	3
2.2.2	Covid 19 simulation north Jutland . . . . .	3
2.3	Benchmark results . . . . .	4
2.4	src . . . . .	5
2.4.1	src/ChemicalSystem.h . . . . .	5
2.4.2	src/CircadianSimulator.h . . . . .	7
2.4.3	src/CombinedElements.h . . . . .	9
2.4.4	src/CovidSimulator.h . . . . .	10
2.4.5	src/CsvWriter.h . . . . .	11
2.4.6	src/Monitor.h . . . . .	12
2.4.7	src/MonitorCallBack.h . . . . .	13
2.4.8	src/plot.hpp . . . . .	14
2.4.9	src/Reaction.h . . . . .	15
2.4.10	src/SimpleSimulator.h . . . . .	16
2.4.11	src/SimulationMethods.h . . . . .	18
2.4.12	src/Species.h . . . . .	19
2.4.13	src/SpeciesQuantityMonitorCallBack.h . . . . .	20
2.4.14	src/SymbolTable.h . . . . .	21
2.5	benchmarks . . . . .	22
2.5.1	benchmarks/benchmark_multithreaded_covid.cpp . . . . .	22
2.6	src . . . . .	23
2.6.1	src/ChemicalSystem.cpp . . . . .	23
2.6.2	src/CombinedElements.cpp . . . . .	25
2.6.3	src/CsvWriter.cpp . . . . .	26
2.6.4	src/main.cpp . . . . .	27
2.6.5	src/plot.cpp . . . . .	28
2.6.6	src/Reaction.cpp . . . . .	30
2.6.7	src/SimulationMethods.cpp . . . . .	33
2.6.8	src/Species.cpp . . . . .	35
2.6.9	src/SpeciesQuantityMonitorCallBack.cpp . . . . .	36
2.7	tests . . . . .	38
2.7.1	tests/doctest.cpp . . . . .	38
2.7.2	tests/test_reaction.cpp . . . . .	39
2.7.3	tests/test_symbolTable.cpp . . . . .	40
2.8	benchmarks . . . . .	41
2.8.1	benchmarks/CMakeLists.txt . . . . .	41
2.9	src . . . . .	42
2.9.1	CMakeLists.txt . . . . .	42
2.10	tests . . . . .	43

This report shows the overall results of several different reaction simulations, as well as the implementation to run these simulations and unit tests for some of the code. The simulations include a simple simulation of a reaction, a simulation of a circadian system, a simulation of the spread of covid 19 without multithreading and one with. Additionally, the report contains 4 figures of simulations that I have run and plotted the results for. Conclusions on these 4 simulations will be given as they are presented. Benchmarking has also been done to test the performance of using different threads with the covid 19 multithreading simulation.

## 1 Setup

The project uses C++17, and the compiler is g++ contained within Msys2/mingw64. The compilation options can be seen in my Cmake file seen in section 2.9.1.

## 2 Presenting the results

Within this section my results will be presented.

### 2.1 Pretty printing

When printing out reactions the form will be of  $A + B \rightarrow C$ . This is done through overloading the *operator«* for the reaction class as can be seen in section 2.6.6.

Below are the results.

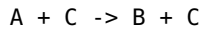
#### 2.1.1 Circadian reactions

```
A + DA -> D_A
D_A -> DA + A
A + DR -> D_R
D_R -> DR + A
D_A -> MA + D_A
DA -> MA + DA
D_R -> MR + D_R
DR -> MR + DR
MA -> MA + A
MR -> MR + R
A + R -> C
C -> R
A -> env
R -> env
MA -> env
MR -> env
```

#### 2.1.2 SEIHR covid 19 reactions

```
A + DA -> D_A
D_A -> DA + A
A + DR -> D_R
D_R -> DR + A
D_A -> MA + D_A
DA -> MA + DA
D_R -> MR + D_R
DR -> MR + DR
MA -> MA + A
MR -> MR + R
A + R -> C
C -> R
A -> env
R -> env
MA -> env
MR -> env
```

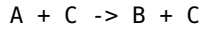
### 2.1.3 Simple reaction



## 2.2 Simulation results

### 2.2.1 Simple reaction

Figure 1 shows a simulation of a simple reaction using a reaction rate lambda of 0.001 and the reaction rule:



With the initial amounts of  $A$  being 100,  $B$  being 0 and  $C$  being 1.

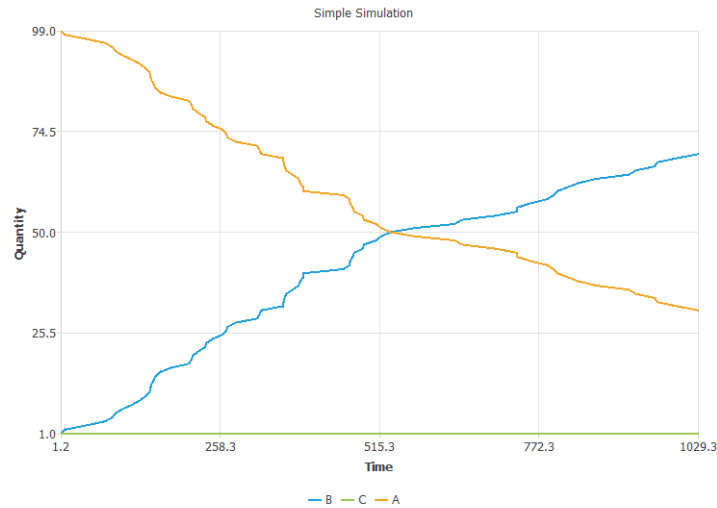


Figure 1: Simple Simulation

As can be seen from the simulation the catalyst affects the reaction linearly such that the amounts of reactant  $A$  and the product  $B$  uniformly decrease and increase respectively.

### 2.2.2 Covid 19 simulation north Jutland

Figure 2 illustrates the the simulation of the spread of covid 19 in north Jutland over a 100 day period. The amounts of hospitalized have been scaled by 1000, such that they are more visible on the graph.

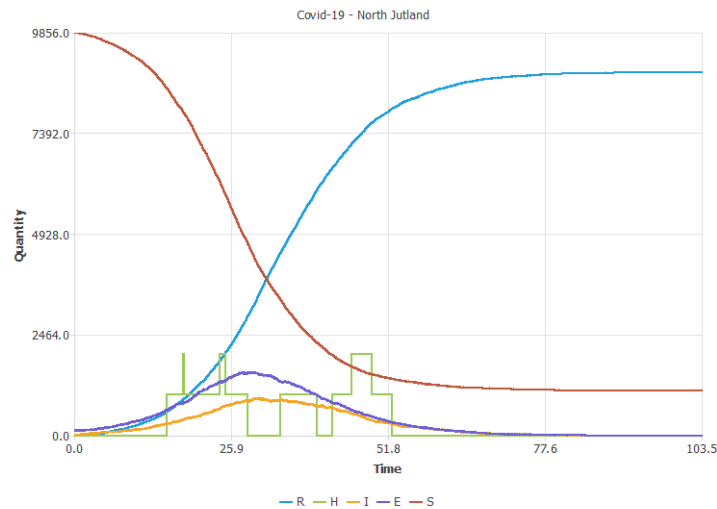


Figure 2: Covid 19 Simulation - North Jutland

The amounts of hospitalized during the 100 day period can be seen to increase in spikes and likewise decrease in spikes, eventually fading off after 52 days. This result seems in line with the infectiousness of covid, causing bursts of outbreaks as it is spread between people, which in turn causes spikes of hospitalizations. The results also seem to be in line with the effects of herd immunity after a while. As would be expected when people are infected they build up immunity, and over time, the virus has no ability to spread.

The implementation calculating the peak hospitalizations and mean hospitalizations can be seen in section 2.6.9. Using this the results show that the peak hospitalizations are 2 and an average 0.673805 adjusting back down from the scaling.

Similarly, figure 3 shows a similar trend as the previous example, but with a more gradual curve of hospitalizations, which seems in line with the amount of people that this simulation included.

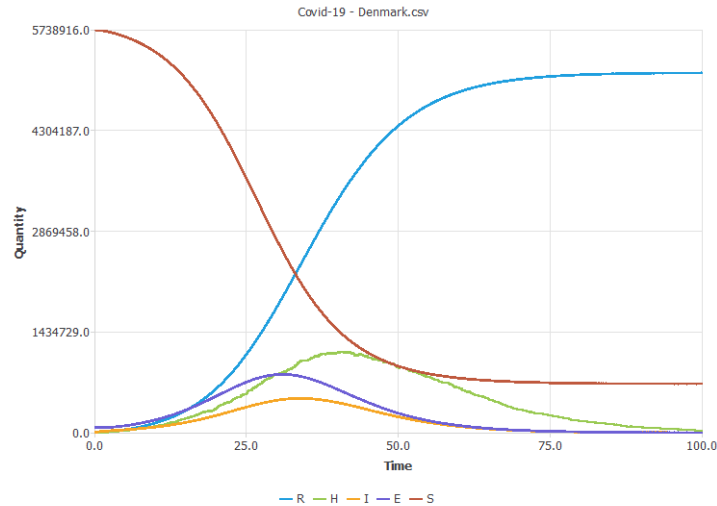


Figure 3: Covid 19 Simulation - Denmark

Here the peak hospitalized were 1222 and the average was 729.595, also adjusting back down from the scaling.

## 2.3 Benchmark results

In order to test the benefit of multithreading, a benchmark was conducted where 20 simulations were run, 10 times for each benchmark in order to get a fitting average. The benchmarks were done using Google Benchmarks. Each benchmark used different amounts of threads to see which performed better. The setup can be seen in section 2.5.1.

Benchmark	Time	CPU	Iterations
BM_multithreadedCovid/8	851990460 ns	0.000 ns	10
BM_multithreadedCovid/1	402505320 ns	0.000 ns	10
BM_multithreadedCovid/5	1899686790 ns	0.000 ns	10
BM_multithreadedCovid/10	976361980 ns	1562500 ns	10
BM_multithreadedCovid/20	851900060 ns	1562500 ns	10

Figure 4: Benchmarks

As the benchmarks clearly illustrate that the use of a single thread beats all others. Even when aligning the amounts of threads, with the threads available on the cpu. This could be due to several factors, such as the overhead from context switching, threads trying to access and modify different variables on the same cache line or thread creation overhead.

## 2.4 src

### 2.4.1 src/ChemicalSystem.h

Listing 1: src/ChemicalSystem.h

```
1  //
2  // Created by Ivik Hostrup.
3  // Chemical system class that represents the stochastic simulation algorithm
4  // Requirement 4.
5  //
6
7  #ifndef STOCHASTICSIMULATION_CHEMICALSYSTEM_H
8  #define STOCHASTICSIMULATION_CHEMICALSYSTEM_H
9
10 #include <chrono>
11 #include <vector>
12 #include "Reaction.h"
13 #include "SymbolTable.h"
14 #include "Monitor.h"
15
16 class ChemicalSystem {
17 public:
18     ChemicalSystem() : m_gen(m_rd()) {};
19
20     template <typename CallBackType>
21     void Simulate(size_t endTime, Monitor<CallBackType>& monitor) {
22         double startTime = 0.0;
23
24         while (startTime <= endTime){
25             ComputeDelay();
26
27             auto reaction_map = m_symbolTable_reactions.GetAllSymbols();
28             auto reaction_with_min_delay = reaction_map.begin()->second;
29
30             for (const auto& [_ , reaction] : reaction_map) {
31                 if (reaction->GetDelay() < reaction_with_min_delay->GetDelay()) {
32                     reaction_with_min_delay = reaction;
33                 }
34             }
35
36             startTime += reaction_with_min_delay->GetDelay();
37             //std::cout << startTime << std::endl;
38             auto combinedSpecies = reaction_with_min_delay->GetReactants().GetCombinedSpecies();
39
40             bool reactantsSufficient = true;
41             for (const auto& reactant : combinedSpecies) {
42                 if(reactant->GetQuantity() < 1) { // If amount agents is less than 1, then the ↵
43                     ↵reaction cannot proceed
44                     reactantsSufficient = false;
45                     break;
46                 }
47             }
48
49             if(reactantsSufficient) {
50                 for (const auto& reactant : combinedSpecies) {
51                     reactant->SetQuantity(reactant->GetQuantity() - 1);
52                 }
53
54                 for (const auto& product : ↵
55                     ↵reaction_with_min_delay->GetProducts().GetCombinedSpecies()) {
56                     product->SetQuantity(product->GetQuantity() + 1);
57                 }
58             }
59         }
60     }
61 }
```

```

55         }
56     }
57
58     monitor.OnStateChange(startTime, *this);
59 }
60
61
62 void ComputeDelay();
63 void Reset();
64
65 std::shared_ptr<Species> AddSpecies(const std::string& name, const size_t& initial_amount);
66 std::shared_ptr<Species> GetSpecies(const std::string& name) const;
67 std::shared_ptr<Reaction> AddReaction(const Reaction& reaction, const double& rate_constant);
68 [[nodiscard]] std::vector<std::shared_ptr<Reaction>> GetReactions() const;
69
70
71 friend std::ostream& operator<<(std::ostream& os, const ChemicalSystem& system);
72 private:
73     std::vector<std::shared_ptr<Species>> m_species;
74     std::vector<std::shared_ptr<Reaction>> m_reactions;
75     SymbolTable<Species> m_symbolTable_species;
76     SymbolTable<Reaction> m_symbolTable_reactions;
77     std::unordered_map<std::string, size_t> m_initial_quantities;
78     std::random_device m_rd;
79     std::mt19937 m_gen;
80 };
81
82
83
84
85 #endif //STOCHASTICSIMULATION_CHEMICALSYSTEM_H

```

---

## 2.4.2 src/CircadianSimulator.h

Listing 2: src/CircadianSimulator.h

```
1  //
2  // Created by Ivik Hostrup.
3  // Circadian simulator class that simulates the circadian rhythm of a cell
4  // Subtask A.1
5  //
6
7  #ifndef STOCHASTICSIMULATION_CIRCADIANSIMULATOR_H
8  #define STOCHASTICSIMULATION_CIRCADIANSIMULATOR_H
9
10
11 #include "ChemicalSystem.h"
12 #include "plot.hpp"
13
14
15 class CircadianSimulator {
16 public:
17     CircadianSimulator(size_t endTime) : m_endTime(endTime) {}
18
19     template<typename CallBackType>
20     void RunSimulation(Monitor<CallBackType>& monitor) {
21         ChemicalSystem system;
22
23         auto alphaA = 50.0;
24         auto alpha_A = 500.0;
25         auto alphaR = 0.01;
26         auto alpha_R = 50.0;
27         auto betaA = 50.0;
28         auto betaR = 5.0;
29         auto gammaA = 1.0;
30         auto gammaR = 1.0;
31         auto gammaC = 2.0;
32         auto deltaA = 1.0;
33         auto deltaR = 0.2;
34         auto deltaMA = 10.0;
35         auto deltaMR = 0.5;
36         auto thetaA = 50.0;
37         auto thetaR = 100.0;
38
39         auto DA = system.AddSpecies("DA", 1);
40         auto D_A = system.AddSpecies("D_A", 0);
41         auto DR = system.AddSpecies("DR", 1);
42         auto D_R = system.AddSpecies("D_R", 0);
43         auto MA = system.AddSpecies("MA", 0);
44         auto MR = system.AddSpecies("MR", 0);
45         auto A = system.AddSpecies("A", 0);
46         auto R = system.AddSpecies("R", 0);
47         auto C = system.AddSpecies("C", 0);
48         auto env = system.AddSpecies("env", 0);
49
50         system.AddReaction(A + DA >=> D_A, gammaA);
51         system.AddReaction(D_A >=> DA + A, thetaA);
52         system.AddReaction(A + DR >=> D_R, gammaR);
53         system.AddReaction(D_R >=> DR + A, thetaR);
54         system.AddReaction(D_A >=> MA + D_A, alpha_A);
55         system.AddReaction(DA >=> MA + DA, alphaA);
56         system.AddReaction(D_R >=> MR + D_R, alpha_R);
57         system.AddReaction(DR >=> MR + DR, alphaR);
58         system.AddReaction(MA >=> MA + A, betaA);
```

```

59     system.AddReaction(MR >=> MR + R, betaR);
60     system.AddReaction(A + R >=> C, gammaC);
61     system.AddReaction(C >=> R, deltaA);
62     system.AddReaction(A >=> env, deltaA);
63     system.AddReaction(R >=> env, deltaR);
64     system.AddReaction(MA >=> env, deltaMA);
65     system.AddReaction(MR >=> env, deltaMR);
66
67     system.Simulate(m_endTime, monitor); // Run simulation
68 }
69
70 private:
71     size_t m_endTime;
72     std::vector<std::vector<double>> m_signals;
73 };
74
75
76 #endif //STOCHASTICSIMULATION_CIRCADIANSIMULATOR_H

```

---



### 2.4.3 src/CombinedElements.h

Listing 3: src/CombinedElements.h

```
1  //
2  // Created by Ivik Hostrup.
3  // Combined elements are used to represent the reactants and products of a reaction.
4  // Baseline to satisfy all requirements.
5  //
6
7  #ifndef STOCHASTICSIMULATION_COMBINEDELEMENTS_H
8  #define STOCHASTICSIMULATION_COMBINEDELEMENTS_H
9
10
11
12 #include <vector>
13 #include <memory>
14 #include "Species.h"
15 #include <iostream>
16
17 class CombinedElements {
18 public:
19     void Add(const std::shared_ptr<Species>& species);
20     [[nodiscard]] std::vector<std::shared_ptr<Species>> GetCombinedSpecies() const;
21
22     friend std::ostream& operator<<(std::ostream& os, const CombinedElements& combinedReactants);
23 private:
24     std::vector<std::shared_ptr<Species>> m_combined_species;
25 };
26
27 CombinedElements operator+(const std::shared_ptr<Species>& leftElement, const ↵
↵std::shared_ptr<Species>& rightElement);
28
29
30 #endif //STOCHASTICSIMULATION_COMBINEDELEMENTS_H
```

## 2.4.4 src/CovidSimulator.h

Listing 4: src/CovidSimulator.h

```
1  //
2  // Created by Ivik Hostrup.
3  // Covid simulator class that can be used to simulate the spread of covid-19
4  // Subtask A.2
5  //
6
7  #ifndef STOCHASTICSIMULATION_COVIDSIMULATOR_H
8  #define STOCHASTICSIMULATION_COVIDSIMULATOR_H
9
10
11 #include <cstdint>
12 #include "Monitor.h"
13 #include "ChemicalSystem.h"
14 #include "CsvWriter.h"
15 #include "plot.hpp"
16
17 class CovidSimulator {
18 public:
19     CovidSimulator(size_t N, size_t endTime) : m_N(N), m_endTime(endTime) {}
20
21     template<typename CallbackType>
22     void RunCovidSimulator(Monitor<CallbackType>& monitor){
23         ChemicalSystem system;
24
25         const auto eps = 0.0009;
26         const auto I0 = size_t(std::round(eps*m_N));
27         const auto E0 = size_t(std::round(eps*m_N*15));
28         const auto S0 = m_N-I0-E0;
29         const auto R0 = 2.4;
30         const auto alpha = 1.0 / 5.1;
31         const auto gamma = 1.0 / 3.1;
32         const auto beta = R0 * gamma;
33         const auto P_H = 0.9e-3;
34         const auto kappa = gamma * P_H*(1.0-P_H);
35         const auto tau = 1.0/10.12;
36
37         auto S = system.AddSpecies("S", S0);
38         auto E = system.AddSpecies("E", E0);
39         auto I = system.AddSpecies("I", I0);
40         auto H = system.AddSpecies("H", 0);
41         auto R = system.AddSpecies("R", 0);
42
43         system.AddReaction(S+I >=> E+I, beta/m_N);
44         system.AddReaction(E >=> I, alpha);
45         system.AddReaction(I >=> R, gamma);
46         system.AddReaction(I >=> H, kappa);
47         system.AddReaction(H >=> R, tau);
48
49         system.Simulate(m_endTime, monitor);
50     };
51 private:
52     size_t m_N;
53     size_t m_endTime;
54     std::vector<std::vector<double>> m_signals;
55 };
56
57
58 #endif //STOCHASTICSIMULATION_COVIDSIMULATOR_H
```

## 2.4.5 src/CsvWriter.h

Listing 5: src/CsvWriter.h

```
1  //
2  // Created by Ivik Hostrup.
3  // CSV writer class that can be used to write the results of a simulation to a csv file
4  // Early implementation to satisfy requirement 6.
5  //
6
7  #ifndef STOCHASTICSIMULATION_CSVWRITER_H
8  #define STOCHASTICSIMULATION_CSVWRITER_H
9
10
11 #include <string>
12 #include <vector>
13
14 class CsvWriter {
15 public:
16     CsvWriter(const std::string& filename, const std::vector<std::string>& speciesNames)
17         : m_filename(filename), m_species_names(speciesNames) {}
18
19     void WriteToCsv(const std::vector<double>& timepoints, const ↵
20 ↵std::vector<std::vector<double>>& signals) const;
21 private:
22     const std::string m_filename;
23     const std::vector<std::string> m_species_names;
24 };
25
26 #endif //STOCHASTICSIMULATION_CSVWRITER_H
```

## 2.4.6 src/Monitor.h

Listing 6: src/Monitor.h

```
1  //
2  // Created by Ivik Hostrup.
3  // Generic monitor class that can be used to monitor the state of a chemical system
4  // Part of requirement 7.
5  //
6
7  #ifndef STOCHASTICSIMULATION_MONITOR_H
8  #define STOCHASTICSIMULATION_MONITOR_H
9
10
11 #include <utility>
12
13 //forward declaration
14 class ChemicalSystem;
15
16 template<typename CallBackType>
17 class Monitor {
18 public:
19     explicit Monitor(CallBackType& callback) : m_callback(callback) {}
20
21     void OnStateChange(double time, const ChemicalSystem& chemicalSystem) {
22         m_callback(time, chemicalSystem);
23     }
24
25     CallBackType& GetCallback() {
26         return m_callback;
27     }
28 private:
29     CallBackType& m_callback;
30 };
31
32
33 #endif //STOCHASTICSIMULATION_MONITOR_H
```

## 2.4.7 src/MonitorCallBack.h

Listing 7: src/MonitorCallBack.h

```
1  //
2  // Created by Ivik Hostrup.
3  // Monitor callback interface
4  // Part of requirement 7
5  //
6
7  #include "ChemicalSystem.h"
8
9  #ifndef STOCHASTICSIMULATION_MONITORCALLBACK_H
10 #define STOCHASTICSIMULATION_MONITORCALLBACK_H
11
12 #endif //STOCHASTICSIMULATION_MONITORCALLBACK_H
13
14 struct MonitorCallBack {
15     virtual void operator()(double time, const ChemicalSystem& chemicalSystem) = 0;
16 };
```

## 2.4.8 src/plot.hpp

Listing 8: src/plot.hpp

```
1  // Plotting class to create graphs of the simulation results
2  // Taken from class lectures
3  // Requirement 6.
4
5  #ifndef PLOT_HPP
6  #define PLOT_HPP
7
8  #include <memory>
9  #include <vector>
10 #include <unordered_map>
11 #include <string>
12
13 class Plot
14 {
15     std::string title;
16     struct app_t;
17     struct chart_t;
18     std::unique_ptr<app_t> app;      // pimpl of the application
19     std::unique_ptr<chart_t> chart; // pimpl of the chart
20     std::string x_axis_label;
21     std::string y_axis_label;
22
23 public:
24     Plot(std::string title, std::string x_axis_label, std::string y_axis_label, int width, int ↵
↵height);
25     Plot(const Plot&) = delete;
26     Plot& operator=(const Plot&) = delete;
27     Plot(Plot&&) noexcept = default;
28     Plot& operator=(Plot&&) = default;
29     ~Plot() noexcept;
30     void save_to_png(const std::string &filename);
31     void scatter(const std::string& name, const std::vector<double>& x, const ↵
↵std::vector<double>& y);
32     void lines(const std::string& name, const std::vector<double>& x, const std::vector<double>& y);
33     void plot_data(const std::vector<double>& time, const std::unordered_map<std::string, ↵
↵std::vector<double>>& species_quantities);
34
35     void process();
36 };
37
38 #endif //PLOT_HPP
```

## 2.4.9 src/Reaction.h

Listing 9: src/Reaction.h

```
1  //
2  // Created by Ivik Hostrup.
3  // Reaction class is used to represent a chemical reaction.
4  // Baseline to satisfy all requirements.
5  //
6
7  #ifndef STOCHASTICSIMULATION_REACTION_H
8  #define STOCHASTICSIMULATION_REACTION_H
9
10
11
12 #include <memory>
13 #include <vector>
14 #include "Species.h"
15 #include "CombinedElements.h"
16 #include <iostream>
17 #include <random>
18
19 class Reaction {
20 public:
21     void AddReactant(const std::shared_ptr<Species>& species);
22     void AddProduct(const std::shared_ptr<Species>& species);
23     void SetRateConstant(const double& rate_constant);
24     [[nodiscard]] double GetDelay() const;
25     void ComputeDelay(std::mt19937& gen);
26     [[nodiscard]] const CombinedElements& GetReactants() const;
27     [[nodiscard]] const CombinedElements& GetProducts() const;
28     std::string to_string() const;
29
30     friend std::ostream& operator<<(std::ostream& os, const Reaction& reaction);
31 private:
32     CombinedElements m_reactants;
33     CombinedElements m_products;
34     double m_lambda;
35     double m_delay = std::numeric_limits<double>::max();
36
37     void PrintReaction(std::ostream& os) const;
38 };
39
40 Reaction operator>>=(const CombinedElements& combinedReactants, const std::shared_ptr<Species>&
    ↪ product);
41 Reaction operator>>=(const std::shared_ptr<Species>& reactant, const CombinedElements&
    ↪ combinedProducts);
42 Reaction operator>>=(const CombinedElements& combinedReactants, const CombinedElements&
    ↪ combinedProducts);
43 Reaction operator>>=(const std::shared_ptr<Species>& reactant, const std::shared_ptr<Species>&
    ↪ product);
44
45 #endif //STOCHASTICSIMULATION_REACTION_H
```

## 2.4.10 src/SimpleSimulator.h

Listing 10: src/SimpleSimulator.h

```
1  //
2  // Created by Ivik Hostrup.
3  // Simple simulation class that can be used to run a simulation with a simple chemical system.
4  // Requirement 4.
5  //
6
7  #ifndef STOCHASTICSIMULATION_SIMPLESIMULATOR_H
8  #define STOCHASTICSIMULATION_SIMPLESIMULATOR_H
9
10 #include <cstdint>
11 #include "Monitor.h"
12 #include "ChemicalSystem.h"
13 #include "CsvWriter.h"
14 #include "plot.hpp"
15
16 class SimpleSimulator {
17 public:
18     SimpleSimulator(size_t endTime) : m_endTime(endTime) {}
19
20     template<typename CallBackType>
21     void RunFirstSimulation(Monitor<CallBackType>& monitor) {
22         ChemicalSystem system;
23
24         auto A = system.AddSpecies("A", 100);
25         auto B = system.AddSpecies("B", 0);
26         auto C = system.AddSpecies("C", 1);
27
28         system.AddReaction(A + C >= B + C, 0.001);
29
30         std::vector<std::string> speciesToMonitor = monitor.GetCallback().GetMonitoredSpecies();
31         system.Simulate(m_endTime, monitor);
32     }
33
34     template<typename CallBackType>
35     void RunSecondSimulation(Monitor<CallBackType>& monitor) {
36         ChemicalSystem system;
37
38         auto A = system.AddSpecies("A", 100);
39         auto B = system.AddSpecies("B", 0);
40         auto C = system.AddSpecies("C", 2);
41
42         system.AddReaction(A + C >= B + C, 0.001);
43
44         system.Simulate(m_endTime, monitor, false);
45     }
46
47     template<typename CallBackType>
48     void RunThirdSimulation(Monitor<CallBackType>& monitor) {
49         ChemicalSystem system;
50
51         auto A = system.AddSpecies("A", 50);
52         auto B = system.AddSpecies("B", 50);
53         auto C = system.AddSpecies("C", 1);
54
55         system.AddReaction(A + C >= B + C, 0.001);
56
57         system.Simulate(m_endTime, monitor, false);
58     }
59 }
```



```
59
60 private:
61     size_t m_endTime;
62     std::vector<std::vector<double>> m_signals;
63 };
64
65
66 #endif //STOCHASTICSIMULATION_SIMPLESIMULATOR_H
```

---

## 2.4.11 src/SimulationMethods.h

Listing 11: src/SimulationMethods.h

```
1  //
2  // Created by Ivik Hostrup.
3  // Container for all simulation methods
4  // Part of all requirements
5  //
6
7  #ifndef STOCHASTICSIMULATION_SIMULATIONMETHODS_H
8  #define STOCHASTICSIMULATION_SIMULATIONMETHODS_H
9
10 #include <string>
11 #include <vector>
12 #include <thread>
13 #include "SpeciesQuantityMonitorCallBack.h"
14 #include "CircadianSimulator.h"
15 #include "SimpleSimulator.h"
16 #include "CovidSimulator.h"
17 #include "plot.hpp"
18
19 void PlotSimple();
20 void PlotCircadian();
21 void PlotCovid(double N = 10000, const std::string& filename = "covid.csv");
22 void MultithreadedCovid(size_t numSimulations = 20, size_t numThreads = ↵
↵std::thread::hardware_concurrency());
23
24 #endif //STOCHASTICSIMULATION_SIMULATIONMETHODS_H
```

## 2.4.12 src/Species.h

Listing 12: src/Species.h

```
1  //
2  // Created by Ivik Hostrup.
3  // Species class for representing a species in a chemical reaction
4  // Baseline to satisfy all requirements.
5  //
6
7  #ifndef STOCHASTICSIMULATION_SPECIES_H
8  #define STOCHASTICSIMULATION_SPECIES_H
9
10
11 #include <string>
12 #include <iostream>
13
14 class Species {
15 public:
16     Species(std::string name, int initialQuantity):
17         m_name(std::move(name)), m_quantity(initialQuantity) {}
18
19     const std::string& GetName() const;
20     const size_t& GetQuantity() const;
21     void SetQuantity(const size_t& quantity);
22
23     friend std::ostream& operator<<(std::ostream& os, const Species& species);
24 private:
25     std::string m_name;
26     size_t m_quantity;
27 };
28
29
30 #endif //STOCHASTICSIMULATION_SPECIES_H
```

## 2.4.13 src/SpeciesQuantityMonitorCallBack.h

Listing 13: src/SpeciesQuantityMonitorCallBack.h

```
1  //
2  // Created by Ivik Hostrup.
3  // Specific monitor class that can be used to monitor the quantity of a species in a chemical system
4  // Part of requirement 7.
5  //
6
7  #ifndef STOCHASTICSIMULATION_SPECIESQUANTITYMONITORCALLBACK_H
8  #define STOCHASTICSIMULATION_SPECIESQUANTITYMONITORCALLBACK_H
9
10
11 #include <vector>
12 #include <mutex>
13 #include "Monitor.h"
14 #include "Species.h"
15 #include "ChemicalSystem.h"
16 #include "MonitorCallBack.h"
17
18 class SpeciesQuantityMonitorCallBack : public MonitorCallBack {
19 public:
20     explicit SpeciesQuantityMonitorCallBack(const std::vector<std::string>& speciesName)
21         : m_species_names(speciesName), m_signals_monitor(speciesName.size()), m_timepoints() {}
22
23     void operator()(double time, const ChemicalSystem& chemicalSystem) override;
24
25     const std::vector<std::string>& GetMonitoredSpecies() const;
26     void CreatePlot(const std::string& plotName = "Covid Simulation",
27                    const std::string& xAxisLabel = "Time",
28                    const std::string& yAxisLabel = "Quantity",
29                    int width = 800, int height = 600) const;
30     std::pair<double, double> GetPeakAndMean(const std::string& speciesName) const;
31 private:
32     std::vector<std::string> m_species_names;
33     std::vector<std::vector<double>> m_signals_monitor;
34     std::vector<double> m_timepoints;
35     std::mutex m_mutex;
36 };
37
38
39 #endif //STOCHASTICSIMULATION_SPECIESQUANTITYMONITORCALLBACK_H
```

## 2.4.14 src/SymbolTable.h

Listing 14: src/SymbolTable.h

```
1  //
2  // Created by Ivik Hostrup.
3  // Generic symbol table class that can be used to store any type of object
4  // Requirement 3.
5  //
6
7  #ifndef STOCHASTICSIMULATION_SYMBOLTABLE_H
8  #define STOCHASTICSIMULATION_SYMBOLTABLE_H
9
10 #include <string>
11 #include <memory>
12 #include <unordered_map>
13 #include <stdexcept>
14
15 template <typename T>
16 class SymbolTable {
17 public:
18     void AddSymbol(const std::string& name, const std::shared_ptr<T>& object);
19     std::shared_ptr<T> GetSymbol(const std::string& name) const;
20     const std::unordered_map<std::string, std::shared_ptr<T>>& GetAllSymbols() const;
21
22 private:
23     std::unordered_map<std::string, std::shared_ptr<T>> m_symbol_table;
24 };
25
26 template<typename T>
27 std::shared_ptr<T> SymbolTable<T>::GetSymbol(const std::string &name) const {
28     auto item = m_symbol_table.find(name);
29     if (item != m_symbol_table.end()) {
30         return item->second;
31     } else {
32         throw std::runtime_error("Symbol not found");
33     }
34 }
35
36 template<typename T>
37 const std::unordered_map<std::string, std::shared_ptr<T>>& SymbolTable<T>::GetAllSymbols() const {
38     return m_symbol_table;
39 }
40
41 template<typename T>
42 void SymbolTable<T>::AddSymbol(const std::string &name, const std::shared_ptr<T> &object) {
43     // Add check to see if symbol already exists
44     if(m_symbol_table.find(name) != m_symbol_table.end())
45         throw std::runtime_error("Symbol already exists");
46
47     m_symbol_table[name] = object;
48 }
49
50
51
52 #endif //STOCHASTICSIMULATION_SYMBOLTABLE_H
```

## 2.5 benchmarks

### 2.5.1 benchmarks/benchmark\_multithreaded\_covid.cpp

Listing 15: benchmarks/benchmark\_multithreaded\_covid.cpp

```
1 //
2 // Created by Ivik Hostrup on 6/3/2023.
3 //
4 #include <benchmark/benchmark.h>
5 #include "../src/SimulationMethods.h"
6
7 static void BM_multithreadedCovid(benchmark::State& state) {
8     for (auto _ : state) {
9         MultithreadedCovid(state.range(0));
10    }
11 }
12
13 BENCHMARK(BM_multithreadedCovid)->Arg(std::thread::hardware_concurrency());
14 BENCHMARK(BM_multithreadedCovid)->Arg(1);
15 BENCHMARK(BM_multithreadedCovid)->Arg(5);
16 BENCHMARK(BM_multithreadedCovid)->Arg(10);
17 BENCHMARK(BM_multithreadedCovid)->Arg(20);
```

## 2.6 src

### 2.6.1 src/ChemicalSystem.cpp

Listing 16: src/ChemicalSystem.cpp

```
1 //
2 // Created by Ivik Hostrup.
3 //
4
5 #include <iostream>
6 #include <algorithm>
7 #include <chrono>
8 #include "ChemicalSystem.h"
9 #include "Monitor.h"
10
11
12 std::shared_ptr<Species> ChemicalSystem::AddSpecies(const std::string& name, const size_t& ↗
↪ initial_amount) {
13     auto new_species = std::make_shared<Species>(name, initial_amount);
14     m_symbolTable_species.AddSymbol(name, new_species);
15
16     m_initial_quantities[name] = initial_amount;
17     return new_species;
18 }
19
20 std::shared_ptr<Species> ChemicalSystem::GetSpecies(const std::string& name) const {
21     return m_symbolTable_species.GetSymbol(name);
22 }
23
24
25 std::shared_ptr<Reaction> ChemicalSystem::AddReaction(const Reaction &reaction, const double& ↗
↪ rate_constant) {
26     auto new_reaction = std::make_shared<Reaction>(reaction);
27     new_reaction->SetRateConstant(rate_constant);
28     auto reaction_name = new_reaction->to_string();
29     m_symbolTable_reactions.AddSymbol(reaction_name, new_reaction);
30
31     return new_reaction;
32 }
33
34 std::vector<std::shared_ptr<Reaction>> ChemicalSystem::GetReactions() const {
35     return m_reactions;
36 }
37
38 // Overload << for reactions in system
39 std::ostream& operator<<(std::ostream& os, const ChemicalSystem& system) {
40     for (const auto& reaction : system.GetReactions()) {
41         os << *reaction << std::endl;
42     }
43     return os;
44 }
45
46 void ChemicalSystem::ComputeDelay() {
47     auto reaction_map = m_symbolTable_reactions.GetAllSymbols();
48
49     for(const auto& [name, reaction] : reaction_map){
50         reaction->ComputeDelay(m_gen);
51     }
52 }
53
54 void ChemicalSystem::Reset() {
```

```
55     for (const auto& [name, species] : m_symbolTable_species.GetAllSymbols()) {  
56         species->SetQuantity(m_initial_quantities[name]);  
57     }  
58 }
```

---



## 2.6.2 src/CombinedElements.cpp

Listing 17: src/CombinedElements.cpp

```
1  //
2  // Created by Ivik Hostrup.
3  //
4
5  #include "CombinedElements.h"
6  #include <iostream>
7
8  void CombinedElements::Add(const std::shared_ptr<Species>& species) {
9      if(species){
10         m_combined_species.push_back(species);
11     } else {
12         std::cout << "Species is null" << std::endl;
13     }
14 }
15
16 std::vector<std::shared_ptr<Species>> CombinedElements::GetCombinedSpecies() const{
17     return m_combined_species;
18 }
19
20 std::ostream& operator<<(std::ostream& os, const CombinedElements& combinedReactants) {
21     // Print the Species objects in the CombinedElements object
22     for (const auto& species : combinedReactants.GetCombinedSpecies()) {
23         os << *species;
24     }
25     return os;
26 }
27
28 CombinedElements operator+(const std::shared_ptr<Species>& leftElement, const ↵
↵std::shared_ptr<Species>& rightElement) {
29     CombinedElements combination;
30     combination.Add(leftElement);
31     combination.Add(rightElement);
32
33     return std::move(combination);
34 }
```

### 2.6.3 src/CsvWriter.cpp

Listing 18: src/CsvWriter.cpp

```
1  //
2  // Created by Ivik Hostrup.
3  //
4
5  #include <fstream>
6  #include <iostream>
7  #include "CsvWriter.h"
8
9  void CsvWriter::WriteToCsv(const std::vector<double>& timepoints, const ↵
↵std::vector<std::vector<double>>& signals) const {
10     std::ofstream file;
11
12     file.open(m_filename);
13     if (file.fail()) {
14         std::cerr << "Failed to open file: " << m_filename << "\n";
15         return;
16     }
17     // Write species names as header
18     file << "Time,";
19     for (const auto& name : m_species_names) {
20         file << name;
21         if (&name != &m_species_names.back()) { // Check if not the last element to avoid ↵
↵trailing comma
22             file << ',';
23         }
24     }
25     file << '\n';
26
27     // Write signals
28     for (size_t t = 0; t < signals[0].size(); ++t) {
29         file << timepoints[t] << ',';
30         for (size_t j = 0; j < signals.size(); ++j) {
31             file << signals[j][t];
32             if (j < signals.size() - 1) { // Check if not the last element to avoid trailing comma
33                 file << ',';
34             }
35         }
36         file << '\n';
37     }
38
39     if (file.bad()) {
40         std::cerr << "Failed to write to file: " << m_filename << "\n";
41     }
42     file.close();
43 }
```

## 2.6.4 src/main.cpp

Listing 19: src/main.cpp

```
1
2
3
4 #include "SimulationMethods.h"
5 #include <benchmark/benchmark.h>
6
7 int main() {
8
9     PlotSimple();
10    //PlotCircadian();
11    //PlotCovid(5822763, "Covid-19 - Denmark.csv");
12    //MultithreadedCovid();
13
14    return 0;
15 }
```

## 2.6.5 src/plot.cpp

Listing 20: src/plot.cpp

```
1  //
2  // Taken from class lectures
3  //
4
5  #include "plot.hpp"
6
7  #include <QtWidgets/QApplication>
8  #include <QtWidgets/QMainWindow>
9
10 #include <QtCharts/QChartView>
11 #include <QtCharts/QScatterSeries>
12 #include <QtCharts/QLineSeries>
13 #include <QtCharts/QBoxPlotSeries>
14 #include <QtCharts/QLegendMarker>
15 #include <QtGui/QImage>
16 #include <QtGui/QPainter>
17 #include <QtCore/QtMath>
18
19 #include <random>
20 #include <algorithm>
21
22 QT_CHARTS_USE_NAMESPACE
23
24 struct Plot::app_t
25 {
26     int argc{0};
27     char** args{nullptr};
28     QApplication app{argc, args};
29     QMainWindow window{nullptr};
30 };
31
32 class Plot::chart_t : public QChartView
33 {
34 public:
35     chart_t(): QChartView{new QChart{}, nullptr} {}
36     ~chart_t() noexcept {}
37 };
38
39 Plot::Plot(std::string title, std::string x_axis_label, std::string y_axis_label, int width, int height):
40     title{std::move(title)}, app{std::make_unique<app_t>()},
41     chart{std::make_unique<chart_t>()}, x_axis_label{std::move(x_axis_label)},
42     y_axis_label{std::move(y_axis_label)}
43 {
44     app->window.setCentralWidget(chart.get());
45     app->window.setWindowTitle(this->title.c_str());
46     app->window.resize(width, height);
47     chart->setRenderHint(QPainter::Antialiasing);
48 }
49
50 Plot::~Plot() noexcept = default;
51
52 void Plot::save_to_png(const std::string& filename)
53 {
54     auto* ch = chart->chart();
55     QPixmap pixmap(chart->size());
56     pixmap.fill(Qt::white);
57     QPainter painter(&pixmap);
```

```

56     chart->render(&painter);
57     pixmap.save(QString::fromStdString(filename), "PNG");
58 }
59
60 template <typename Series, typename Chart>
61 void add_xy(Chart& chart, const std::string& name, const std::vector<double>& x, const ↵
↵std::vector<double>& y)
62 {
63     assert(x.size() <= y.size());
64     auto* series = new Series();
65     series->setName(name.c_str());
66     for (auto i = 0; i < x.size(); ++i)
67         series->append(x[i], y[i]);
68     chart.addSeries(series);
69 }
70
71 void Plot::scatter(const std::string& name, const std::vector<double>& x, const ↵
↵std::vector<double>& y)
72 {
73     add_xy<QScatterSeries>(*chart->chart(), name, x, y);
74 }
75
76 void Plot::lines(const std::string& name, const std::vector<double>& x, const ↵
↵std::vector<double>& y)
77 {
78     add_xy<QLineSeries>(*chart->chart(), name, x, y);
79 }
80
81 void Plot::plot_data(const std::vector<double>& time, const std::unordered_map<std::string, ↵
↵std::vector<double>>& species_quantities)
82 {
83     for (const auto& [species, quantities] : species_quantities)
84     {
85         lines(species, time, quantities);
86     }
87 }
88
89 void Plot::process()
90 {
91     auto* ch = chart->chart();
92     ch->setTitle(title.c_str());
93     ch->createDefaultAxes();
94     ch->setDropShadowEnabled(false);
95
96     if (!ch->axes(Qt::Vertical).isEmpty())
97         ch->axes(Qt::Vertical).first()->setTitleText(y_axis_label.c_str());
98     if (!ch->axes(Qt::Horizontal).isEmpty())
99         ch->axes(Qt::Horizontal).first()->setTitleText(x_axis_label.c_str());
100
101     ch->legend()->setMarkerShape(QLegend::MarkerShapeFromSeries);
102     ch->legend()->setAlignment(Qt::AlignBottom);
103
104     app->window.show();
105     app->app.exec();
106 }

```

## 2.6.6 src/Reaction.cpp

Listing 21: src/Reaction.cpp

```
1  //
2  // Created by Ivik Hostrup.
3  //
4
5  #include "Reaction.h"
6  #include <iostream>
7  #include <sstream>
8  #include <random>
9
10 void Reaction::AddReactant(const std::shared_ptr<Species>& species){
11     m_reactants.Add(species);
12 }
13
14 void Reaction::AddProduct(const std::shared_ptr<Species>& species) {
15     m_products.Add(species);
16 }
17
18 [[nodiscard]] const CombinedElements& Reaction::GetReactants() const {
19     return m_reactants;
20 }
21
22 [[nodiscard]] const CombinedElements& Reaction::GetProducts() const {
23     return m_products;
24 }
25
26
27 void Reaction::SetRateConstant(const double& rate_constant){
28     m_lambda = rate_constant;
29 }
30
31 std::ostream& operator<<(std::ostream& os, const Reaction& reaction) {
32     reaction.PrintReaction(os);
33     return os;
34 }
35
36 std::string Reaction::to_string() const {
37     std::ostringstream os;
38     PrintReaction(os);
39     return os.str();
40 }
41
42 void Reaction::PrintReaction(std::ostream &os) const {
43     size_t count = 0;
44     for (const auto& reactant : this->GetReactants().GetCombinedSpecies()) {
45         os << *reactant;
46         if(++count < this->GetReactants().GetCombinedSpecies().size()){
47             os << " + ";
48         }
49     }
50     os << " -> ";
51
52     count = 0;
53     for (const auto& product : this->GetProducts().GetCombinedSpecies()) {
54         os << *product;
55         if(++count < this->GetProducts().GetCombinedSpecies().size()){
56             os << " + ";
57         }
58     }
```

```

59 }
60
61 double Reaction::GetDelay() const {
62     return m_delay;
63 }
64
65 void Reaction::ComputeDelay(std::mt19937& gen) {
66     auto lambda = m_lambda;
67
68     for(const auto& reactant : m_reactants.GetCombinedSpecies()){
69         lambda *= static_cast<double>(reactant->GetQuantity());
70     }
71
72     std::exponential_distribution distribution(lambda);
73
74     m_delay = distribution(gen);
75 }
76
77 // Multiple reactants and single products
78 Reaction operator>>=(const CombinedElements& combinedReactants, const std::shared_ptr<Species>& ↵
    ↪product){
79     Reaction reaction;
80
81     for(auto const& reactant : combinedReactants.GetCombinedSpecies()){
82         reaction.AddReactant(reactant);
83     }
84
85     if(product->GetName() != "env"){
86         reaction.AddProduct(product);
87     }
88
89     return std::move(reaction);
90 }
91
92 // single reactant and multiple products
93 Reaction operator>>=(const std::shared_ptr<Species>& reactant, const CombinedElements& ↵
    ↪combinedProducts){
94     Reaction reaction;
95
96     reaction.AddReactant(reactant);
97
98     for(auto const& product : combinedProducts.GetCombinedSpecies()){
99         if(product->GetName() != "env"){
100             reaction.AddProduct(product);
101         }
102     }
103
104     return std::move(reaction);
105 }
106
107 // Multiple reactants and multiple products
108 Reaction operator>>=(const CombinedElements& combinedReactants, const CombinedElements& ↵
    ↪combinedProducts){
109     Reaction reaction;
110
111     for(auto const& reactant : combinedReactants.GetCombinedSpecies()){
112         reaction.AddReactant(reactant);
113     }
114
115     for(auto const& product : combinedProducts.GetCombinedSpecies()){
116         if(product->GetName() != "env"){

```

```

117         reaction.AddProduct(product);
118     }
119 }
120
121     return std::move(reaction);
122 }
123
124 // single reactant and a single product
125 Reaction operator>>=(const std::shared_ptr<Species>& reactant, const std::shared_ptr<Species>& ↗
↪product){
126     Reaction reaction;
127
128     reaction.AddReactant(reactant);
129     if(product->GetName() != "env"){
130         reaction.AddProduct(product);
131     }
132
133     return std::move(reaction);
134 }

```

---



## 2.6.7 src/SimulationMethods.cpp

Listing 22: src/SimulationMethods.cpp

```
1  //
2  // Created by Ivik Hostrup.
3  //
4
5
6  #include <future>
7  #include "CovidSimulator.h"
8  #include "SpeciesQuantityMonitorCallBack.h"
9  #include "CircadianSimulator.h"
10 #include "SimpleSimulator.h"
11
12 void PlotSimple(){
13     std::vector<std::string> speciesToMonitor = {"A", "B", "C"};
14     SpeciesQuantityMonitorCallBack speciesMonitorCallBack(speciesToMonitor);
15     Monitor monitor(speciesMonitorCallBack);
16
17     SimpleSimulator simulator(1000);
18     simulator.RunFirstSimulation(monitor);
19     monitor.GetCallback().CreatePlot("Simple Simulation");
20 };
21
22 void PlotCircadian(){
23     std::vector<std::string> speciesToMonitor = {"A", "R", "C"};
24     SpeciesQuantityMonitorCallBack speciesMonitorCallBack(speciesToMonitor);
25     Monitor monitor(speciesMonitorCallBack);
26
27     CircadianSimulator simulator(100);
28     simulator.RunSimulation(monitor);
29     monitor.GetCallback().CreatePlot("Circadian Simulation");
30 };
31
32 void PlotCovid(double N = 10000, const std::string& title = "Covid Simulation"){
33     std::vector<std::string> speciesToMonitor = {"S", "E", "I", "H", "R"};
34     SpeciesQuantityMonitorCallBack speciesMonitorCallBack(speciesToMonitor);
35     Monitor monitor(speciesMonitorCallBack);
36
37     CovidSimulator simulator(N, 100);
38     simulator.RunCovidSimulator(monitor);
39
40     auto [peak, mean] = speciesMonitorCallBack.GetPeakAndMean("H");
41
42     std::cout << "Peak hospitalized: " << peak << std::endl;
43     std::cout << "Mean hospitalized: " << mean << std::endl;
44
45     monitor.GetCallback().CreatePlot(title);
46 };
47
48 void MultithreadedCovid(size_t numSimulations = 20, size_t numThreads = ↗
→std::thread::hardware_concurrency()){
49     std::vector<std::string> speciesToMonitor = {"S", "E", "I", "H", "R"};
50
51     std::vector<std::future<std::pair<double, double>>> futures(numSimulations);
52
53     for(size_t i = 0; i < numSimulations; ++i) {
54         futures[i] = std::async(std::launch::async, [&speciesToMonitor] {
55             // Create a new callback and monitor for each thread
56             SpeciesQuantityMonitorCallBack speciesMonitorCallBack(speciesToMonitor);
57             Monitor monitor(speciesMonitorCallBack);
```

```

58
59     // Create a new simulator for each thread
60     CovidSimulator simulator(10000, 100);
61     simulator.RunCovidSimulator(monitor);
62
63     // Compute and return the peak hospitalized number
64     return speciesMonitorCallBack.GetPeakAndMean("H");
65 });
66 }
67
68 std::vector<double> peakValues(numSimulations);
69 for (size_t i = 0; i < numSimulations; ++i) {
70     auto [peaks, _] = futures[i].get();
71     peakValues[i] = peaks;
72 }
73
74 double meanPeakAcrossSimulations = std::accumulate(peakValues.begin(), peakValues.end(), ↗
↪0.0) / numSimulations;
75
76 //std::cout << "Mean peak hospitalized number: " << meanPeakAcrossSimulations << std::endl;
77 }

```

---

## 2.6.8 src/Species.cpp

Listing 23: src/Species.cpp

```
1  //
2  // Created by Ivik Hostrup.
3  //
4
5  #include <stdexcept>
6  #include "Species.h"
7  #include <iostream>
8
9  const std::string& Species::GetName() const {
10     return m_name;
11 }
12
13 std::ostream& operator<<(std::ostream& os, const Species& species) {
14     os << species.GetName();
15     return os;
16 }
17
18 const size_t& Species::GetQuantity() const {
19     return m_quantity;
20 }
21
22 void Species::SetQuantity(const size_t &quantity) {
23     m_quantity = quantity;
24 }
```

## 2.6.9 src/SpeciesQuantityMonitorCallBack.cpp

Listing 24: src/SpeciesQuantityMonitorCallBack.cpp

```
1  //
2  // Created by Ivik Hostrup.
3  //
4
5  #include <algorithm>
6  #include "SpeciesQuantityMonitorCallBack.h"
7  #include "plot.hpp"
8
9
10 void SpeciesQuantityMonitorCallBack::operator()(double time, const ChemicalSystem ↵
    ↪&chemicalSystem) {
11     std::scoped_lock lock(m_mutex);
12
13     m_timepoints.push_back(time);
14     for(size_t i = 0; i < m_species_names.size(); ++i) {
15         m_signals_monitor[i].push_back(chemicalSystem.GetSpecies(m_species_names[i])->GetQuantity());
16     }
17
18     // Mutex unlocks when going out of scope
19 }
20
21 const std::vector<std::string>& SpeciesQuantityMonitorCallBack::GetMonitoredSpecies() const {
22     return m_species_names;
23 }
24
25 void SpeciesQuantityMonitorCallBack::CreatePlot(const std::string &plotName,
26                                                 const std::string &xAxisLabel,
27                                                 const std::string &yAxisLabel,
28                                                 int width, int height) const {
29
30     // Create map for species and their quantities
31     std::unordered_map<std::string, std::vector<double>> speciesQuantities;
32
33     for(size_t i = 0; i < m_species_names.size(); ++i) {
34         std::vector<double> signals = m_signals_monitor[i];
35         if(m_species_names[i] == "H") {
36             for(size_t j = 0; j < signals.size(); ++j) {
37                 signals[j] *= 1000;
38             }
39         }
40         speciesQuantities[m_species_names[i]] = signals;
41     }
42
43     Plot plot(plotName, xAxisLabel, yAxisLabel, width, height);
44
45     plot.plot_data(m_timepoints, speciesQuantities);
46
47     plot.process();
48
49     plot.save_to_png(plotName + ".png");
50 }
51
52
53 std::pair<double, double> SpeciesQuantityMonitorCallBack::GetPeakAndMean(const std::string& ↵
    ↪speciesName) const {
54
55     auto it = std::find(m_species_names.begin(), m_species_names.end(), speciesName);
56 }
```

```

57     if(it != m_species_names.end()) {
58         size_t index = std::distance(m_species_names.begin(), it);
59
60         auto peak = *std::max_element(m_signals_monitor[index].begin(), ↵
↵m_signals_monitor[index].end());
61         auto sum = std::accumulate(m_signals_monitor[index].begin(), ↵
↵m_signals_monitor[index].end(), 0.0);
62         auto mean = sum / m_signals_monitor[index].size();
63
64         return std::make_pair(peak, mean);
65     } else {
66         throw std::invalid_argument("Species name not found");
67     }
68 }

```

---

## 2.7 tests

### 2.7.1 tests/doctest.cpp

Listing 25: tests/doctest.cpp

```
1 //  
2 // Created by Ivik Hostrup on 6/3/2023.  
3 //  
4  
5 #define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN  
6 #include <doctest.h>
```

## 2.7.2 tests/test\_reaction.cpp

Listing 26: tests/test\_reaction.cpp

```
1  //
2  // Created by Ivik Hostrup on 6/2/2023.
3  //
4
5
6  #include "doctest.h"
7  #include "../src/ChemicalSystem.h"
8
9  TEST_CASE("Testing Reaction class") {
10     ChemicalSystem system;
11     auto S = system.AddSpecies("S", 10);
12     auto E = system.AddSpecies("E", 20);
13     auto I = system.AddSpecies("I", 30);
14
15     auto constructedReaction = system.AddReaction(S+E >= I, 0.5);
16
17     SUBCASE("Testing AddReactant and GetReactants") {
18         Reaction reaction;
19         reaction.AddReactant(S);
20         CHECK(reaction.GetReactants().GetCombinedSpecies().size() == 1);
21     }
22
23     SUBCASE("Testing AddProduct and GetProducts") {
24         Reaction reaction;
25         reaction.AddProduct(E);
26         CHECK(reaction.GetProducts().GetCombinedSpecies().size() == 1);
27     }
28
29     SUBCASE("Testing reaction pretty print") {
30         CHECK(constructedReaction->to_string() == "S + E -> I");
31     }
32 }
```

### 2.7.3 tests/test\_symbolTable.cpp

Listing 27: tests/test\_symbolTable.cpp

```
1  //
2  // Created by Ivik Hostrup on 6/2/2023.
3  //
4
5  #include "doctest.h"
6  #include "../src/ChemicalSystem.h"
7
8  TEST_CASE("Testing SymbolTable class") {
9      ChemicalSystem system;
10     auto S = system.AddSpecies("S", 10);
11
12     SUBCASE("Testing AddSymbol and GetSymbol") {
13         SymbolTable<Species> table;
14         table.AddSymbol("S", S);
15         CHECK(table.GetSymbol("S")->GetQuantity() == 10);
16     }
17
18     SUBCASE("Testing AddSymbol with existing symbol") {
19         SymbolTable<Species> table;
20         table.AddSymbol("S", S);
21
22         CHECK_THROWS_AS(table.AddSymbol("S", S), std::runtime_error);
23     }
24
25     SUBCASE("Testing GetAllSymbols") {
26         SymbolTable<Species> table;
27         table.AddSymbol("S", S);
28         CHECK(table.GetAllSymbols().size() == 1);
29     }
30 }
```



## 2.8 benchmarks

### 2.8.1 benchmarks/CMakeLists.txt

Listing 28: benchmarks/CMakeLists.txt

```
1 cmake_minimum_required(VERSION 3.24)
2
3 # Google Benchmark requires at least C++11
4 set(CMAKE_CXX_STANDARD 11)
5
6 # List of files
7 set(BENCHMARK_FILES
8     benchmark_multithreaded_covid.cpp
9 )
10
11 add_executable(Benchmarks ${BENCHMARK_FILES})
12
13 target_link_libraries(Benchmarks benchmark::benchmark_main stochasticsimulation_lib)
```

## 2.9 src

### 2.9.1 CMakeLists.txt

Listing 29: CMakeLists.txt

```
1 cmake_minimum_required(VERSION 3.24)
2 project(StochasticSimulation)
3
4 set(CMAKE_CXX_STANDARD 17)
5
6 # Qt5
7 find_package(Qt5 COMPONENTS Widgets Charts QUIET)
8 if (Qt5_FOUND)
9     message(STATUS "Qt5 found, fantastic!")
10    set(CMAKE_AUTOMOC ON)
11    set(CMAKE_AUTORCC ON)
12    set(CMAKE_AUTOUIC ON)
13 else(Qt5_FOUND)
14    message(WARNING "Qt5 NOT found, test_qt5 will be disabled. Please install qt5charts ↗
    ↪development package.")
15 endif(Qt5_FOUND)
16
17 include_directories(include)
18
19 # Create a library from source files
20 add_library(stochasticsimulation_lib src/Species.cpp src/Reaction.cpp src/ChemicalSystem.cpp ↗
    ↪src/CombinedElements.cpp src/SpeciesQuantityMonitorCallBack.cpp src/CsvWriter.cpp src/plot.cpp ↗
    ↪src/SimulationMethods.cpp src/SimulationMethods.h)
21
22 # Link the library to the required Qt5 components
23 if(Qt5_FOUND)
24    target_link_libraries(stochasticsimulation_lib Qt5::Widgets Qt5::Charts)
25 endif(Qt5_FOUND)
26
27 add_executable(StochasticSimulation src/main.cpp benchmarks/benchmark_multithreaded_covid.cpp)
28
29 target_link_libraries(StochasticSimulation stochasticsimulation_lib benchmark::benchmark)
30
31 set(BENCHMARK_ENABLE_GTEST_TESTS ON CACHE BOOL "Enable benchmark's tests" FORCE)
32 set(BENCHMARK_DOWNLOAD_DEPENDENCIES ON CACHE BOOL "Let benchmark download its dependencies" FORCE)
33 add_subdirectory(benchmark EXCLUDE_FROM_ALL)
34
35 add_subdirectory(tests)
36 add_subdirectory(benchmarks) # different from benchmark
```

## 2.10 tests

### 2.10.1 tests/CMakeLists.txt

Listing 30: tests/CMakeLists.txt

```
1 add_library(doctest OBJECT doctest.cpp)
2
3 add_executable(tests test_reaction.cpp test_symbolTable.cpp $<TARGET_OBJECTS:doctest>)
4 target_link_libraries(tests stochasticsimulation_lib benchmark::benchmark)
5
6 add_test(NAME tests COMMAND tests)
```