sP Exam Mini-project Assingment

Jonas Jerup Svare

June 4, 2023

This project has been developed on Windows 10, version 21H2, with the CLion compiler, version 2023.1.3.
It does not require and specific tool or libraries to run.

Output figures can be found in Section 1.

Listing 1: CMakeLists.txt

```
cmake_minimum_required(VERSION 3.25)
project(Exam)

set(CMAKE_CXX_STANDARD 23)

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -fopenmp")

add_executable(Exam main.cpp lib/lib.cpp lib/lib.h lib/graph.cpp lib/graph.h)
```

Listing 2: lib.cpp

```
//
// Created by jonas on 31-05-2023.
//

#include <random>
#include <iostream>
#include <algorithm>
#include "lib.h"

//using namespace stochastic;
namespace stochastic {

    /** ==================================
     *  AGENT CLASS
     *  ==================================
     */

    /*
     * Req. 1
     * Operator overload (+)
     */
    std::vector<stochastic::Agent> stochastic::Agent::operator+(const stochastic::Agent &rhs) const {
        std::vector<stochastic::Agent> agents;
        agents.push_back(*this);
        agents.push_back(rhs);
        return agents;
    }

    /*
     * Req. 1
     * Operator overload (>>=)
     */
    stochastic::Reaction operator>>=(const Agent &lhs, const Agent &rhs) {
```

```
        auto reaction = stochastic::Reaction();
        std::string out;

        out += lhs.name;
        reaction.agentVec.push_back(lhs);
        reaction.actionVec.emplace_back("increment");

        out += " -> ";

        out += rhs.name;
        reaction.agentVec.push_back(rhs);
        reaction.actionVec.emplace_back("increment");

        reaction.leftHandSide = std::vector<Agent>{lhs};
        reaction.rightHandSide = std::vector<Agent>{rhs};
        reaction.out = out;
        return reaction;
    }

    /*
     * Req. 1
     * Operator overload (>>=)
     */
    stochastic::Reaction operator>>=(const Agent &lhs, const std::vector<Agent> &rhs) {
        auto reaction = stochastic::Reaction();
        std::string out;

        out += lhs.name;
        reaction.agentVec.push_back(lhs);
        reaction.actionVec.emplace_back("increment");

        out += " -> ";

        for (int i = 0; i < rhs.size(); ++i) {
            if (i > 0) {
                out += " + ";
            }
            out += rhs[i].name;
            reaction.agentVec.push_back(rhs[i]);
            reaction.actionVec.emplace_back("increment");
        }

        reaction.leftHandSide = std::vector<Agent>{lhs};
        reaction.rightHandSide = rhs;
        reaction.out = out;
        return reaction;
    }

    /*
     * Req. 1
     * Operator overload (>>=)
     */
    stochastic::Reaction operator>>=(const std::vector<Agent> &lhs, const Agent &rhs) {
        auto reaction = stochastic::Reaction();
        std::string out;
        for (int i = 0; i < lhs.size(); ++i) {
            if (i > 0) {
                out += " + ";
            }
            out += lhs[i].name;
            reaction.agentVec.push_back(lhs[i]);
```

```
95              reaction.actionVec.emplace_back("decrement");
96          }
97
98          out += " -> ";
99
100         out += rhs.name;
101         reaction.agentVec.push_back(rhs);
102         reaction.actionVec.emplace_back("increment");
103
104         reaction.leftHandSide = lhs;
105         reaction.rightHandSide = std::vector<Agent>{rhs};
106         reaction.out = out;
107         return reaction;
108     }
109
110     /*
111      * Req. 1
112      * Operator overload (>>=)
113      */
114     stochastic::Reaction operator>>=(const std::vector<Agent> &lhs, const std::vector<Agent>
    ↪&rhs) {
115         auto reaction = stochastic::Reaction();
116         std::string out;
117         for (int i = 0; i < lhs.size(); ++i) {
118             if (i > 0) {
119                 out += " + ";
120             }
121             out += lhs[i].name;
122             reaction.agentVec.push_back(lhs[i]);
123             reaction.actionVec.emplace_back("decrement");
124         }
125
126         out += " -> ";
127
128         for (int i = 0; i < rhs.size(); ++i) {
129             if (i > 0) {
130                 out += " + ";
131             }
132             out += rhs[i].name;
133             reaction.agentVec.push_back(rhs[i]);
134             reaction.actionVec.emplace_back("increment");
135         }
136
137         reaction.leftHandSide = lhs;
138         reaction.rightHandSide = rhs;
139         reaction.out = out;
140         return reaction;
141     }
142
143     /*
144      * Req. 1
145      * Operator overload (>>=)
146      * Used for reactants that decay into environment
147      */
148     stochastic::Reaction operator>>=(const Agent &lhs, const std::string &env) {
149         auto reaction = stochastic::Reaction();
150         std::string out;
151
152         out += lhs.name;
153         reaction.agentVec.push_back(lhs);
154         reaction.actionVec.emplace_back("increment");
```

```
155
156         out += " -> " + env;
157
158         reaction.leftHandSide = std::vector<Agent>{lhs};
159         reaction.out = out;
160         return reaction;
161     }
162
163     /** ================================
164      *  MONITOR CLASS
165      *  ================================
166      */
167
168     void stochastic::Monitor::insert (double &time, SymbolTable<Agent> &table) {
169         auto it = monitorMap.find(time);
170
171         if (it != monitorMap.end()) {
172             auto tableVec = it -> second;
173             tableVec.push_back(table);
174             monitorMap[time] = tableVec;
175         }
176         else {
177             monitorMap[time] = std::vector<SymbolTable<Agent>> {table};
178         }
179     }
180
181     void stochastic::Monitor::fileStream(const std::string& filePath) {
182         std::ofstream outFile (filePath);
183
184         if (outFile.is_open()) {
185             outFile << "time,agentname,agentamount" << std::endl;
186
187             for (auto &mapping : monitorMap) {
188                 for (auto &table : mapping.second) {
189                     for (auto &agent : table.fetchTable()) {
190                         outFile << std::to_string(mapping.first) + "," + agent.first + "," +
↪std::to_string(agent.second.amount) << std::endl;
191                     }
192                 }
193             }
194             outFile.close();
195         }
196     }
197
198     int stochastic::Monitor::estimatePeak() {
199         auto peak = 0;
200         for (auto &mapping : monitorMap) {
201             for (auto &table: mapping.second) {
202                 auto H = table.get("H");
203                 if (H.amount > peak) {
204                     peak = H.amount;
205                 }
206             }
207         }
208
209         std::cout << "Estimated peak of hospitalized agents: " + std::to_string(peak) << std::endl;
210         return peak;
211     }
212
213     void stochastic::Monitor::multiplyH() {
214         for (auto &mapping : monitorMap) {
```

```
215             for (auto &table: mapping.second) {
216                 auto H = table.get("H");
217                 H.amount *= 1000;
218                 table.insert("H", H);
219             }
220         }
221     }
222 
223     /** ==================================
224      *  ALGORITHM CLASS
225      *  ==================================
226      */
227 
228     double stochastic::Algorithm::computeDelay(stochastic::Reaction &r,
    ↪stochastic::SymbolTable<Agent> &table) {
229         std::random_device rd;
230         std::mt19937 gen(rd());
231 
232         double lambdaK = 1.0;
233         for (auto &agent: r.leftHandSide) {
234             lambdaK *= table.get(agent.name).amount;
235         }
236         lambdaK *= r.lambdaRate;
237 
238         std::exponential_distribution<double> expDist(lambdaK);
239         return expDist(gen);
240     }
241 
242     bool stochastic::Algorithm::amountChecker(const stochastic::Reaction &reaction,
    ↪stochastic::SymbolTable<Agent> &table) {
243         for (auto &agent: reaction.leftHandSide) {
244             if (!(agent.amount <= table.get(agent.name).amount)) {
245                 return false;
246             }
247         }
248         return true;
249     }
250 
251     /**
252      * Req. 4
253      * This is the stochastic simulation
254      * @param reactionVec Used as the set of reactions to compute
255      * @param endTime Defines the end time
256      * @param agentVec Used to create our symbol table for
257      * @param filePath The path to where we want to save the state monitor
258      * @return
259      */
260     stochastic::Monitor stochastic::Algorithm::simulation(
261             std::vector<stochastic::Reaction> &reactionVec,
262             double endTime,
263             std::vector<stochastic::Agent> &agentVec,
264             const std::string& filePath)
265     {
266         double t = 0.0;
267         Monitor monitor;
268         stochastic::SymbolTable<stochastic::Agent> table =
    ↪stochastic::SymbolTable<stochastic::Agent>::generateSymbolTable(agentVec);
269 
270         while (t <= endTime) {
271             stochastic::Reaction minDelayRec = stochastic::Reaction();
272             for (auto &reaction: reactionVec) {
```

```
273                 reaction.delay = stochastic::Algorithm::computeDelay(reaction, table);
274
275                 if (reaction.delay < minDelayRec.delay) {
276                     minDelayRec = reaction;
277                 }
278             }
279
280             t += minDelayRec.delay;
281
282             if (all_of(minDelayRec.leftHandSide.begin(), minDelayRec.leftHandSide.end(),
283                        [&table](const auto &agent) { return table.get(agent.name).amount > 0;   ↙
  ↪})) {
284                 for (stochastic::Agent &r: minDelayRec.leftHandSide) {
285                     auto a = table.get(r.name);
286                     a.amount -= 1;
287                     table.insert(r.name, a);
288                 }
289                 for (stochastic::Agent &p: minDelayRec.rightHandSide) {
290                     auto a = table.get(p.name);
291                     a.amount += 1;
292                     table.insert(p.name, a);
293                 }
294             }
295             //TODO: print/save/monitor state
296             monitor.insert(t, table);
297         }
298
299         monitor.fileStream(filePath);
300         return monitor;
301     }
302
303     /** ==================================
304      *  VISUALIZER CLASS
305      *  ==================================
306      */
307
308     /**
309      * Req. 2
310      * This function pretty prints the reaction network
311      * @param reactionVec
312      */
313     void stochastic::Visualizer::prettyPrintReactions(const std::vector<Reaction> &reactionVec) {
314         std::cout << "=======================================" << std::endl;
315         for (auto &reaction: reactionVec) {
316             std::cout << "Reaction: " + reaction.out + " || Rate: " +   ↙
  ↪std::to_string(reaction.lambdaRate) << std::endl;
317         }
318         std::cout << "=======================================" << std::endl;
319     }
320
321     /**
322      * Req. 2
323      * This function generates a network graph of any given set of reactions
324      * @param reactionVec
325      * @param filePath
326      */
327     void stochastic::Visualizer::generateNetworkGraph(const std::vector<Reaction> &reactionVec,   ↙
  ↪const std::string &filePath) {
328         std::ofstream outfile(filePath);
329
330         std::vector<std::string> nameVec;
```

```
331
332         std::string numNode = "N";
333
334         outfile << "digraph {" << std::endl;
335
336         for (auto &reaction: reactionVec) {
337             for (auto &lhs: reaction.leftHandSide) {
338                 if (!(std::find(nameVec.begin(), nameVec.end(), lhs.name) != nameVec.end())) {
339                     outfile << lhs.name + " [shape=box];" << std::endl;
340                     nameVec.push_back(lhs.name);
341                 }
342                 outfile << lhs.name + " -> " + numNode << std::endl;
343             }
344             for (auto &rhs: reaction.rightHandSide) {
345                 if (!(std::find(nameVec.begin(), nameVec.end(), rhs.name) != nameVec.end())) {
346                     outfile << rhs.name + " [shape=box];" << std::endl;
347                     nameVec.push_back(rhs.name);
348                 }
349                 outfile << numNode + " -> " + rhs.name << std::endl;
350             }
351             outfile << numNode + " [label=\"" + std::to_string(reaction.lambdaRate) + "\"];";
352             numNode += "1";
353         }
354
355         outfile << "}" << std::endl;
356         outfile.close();
357     }
358 }
```

Listing 3: lib.h

```
1  //
2  // Created by jonas on 31-05-2023.
3  //
4
5  #ifndef EXAM_LIB_H
6  #define EXAM_LIB_H
7
8  #include <string>
9  #include <vector>
10 #include <map>
11 #include <cfloat>
12 #include <fstream>
13
14 namespace stochastic {
15     class Reaction;
16
17     class Agent {
18     public:
19         std::string name;
20         int amount = 0;
21
22         Agent() = default;
23
24         Agent(const std::string& name, int amount) {
25             this->name = name;
26             this->amount = amount;
27         }
28
29         //Req. 1
30         std::vector<Agent> operator+(const Agent &other) const;
31         friend stochastic::Reaction operator >>= (const Agent &lhs, const Agent &rhs);
```

```
32          friend stochastic::Reaction operator >>= (const std::vector<Agent> &lhs, const Agent &rhs);
33          friend stochastic::Reaction operator >>= (const Agent &lhs, const std::vector<Agent> &rhs);
34          friend stochastic::Reaction operator >>= (const std::vector<Agent> &lhs, const
    ↪std::vector<Agent> &rhs);
35          friend stochastic::Reaction operator>>=(const Agent &lhs,  const std::string &env);
36
37      };
38
39      class Reaction {
40      public:
41          std::vector<stochastic::Agent> agentVec;
42          std::vector<std::string> actionVec;
43
44          std::vector<Agent> leftHandSide;
45          std::vector<Agent> rightHandSide;
46
47          double lambdaRate{};
48          double delay = DBL_MAX;
49
50          std::string out;
51
52          Reaction() = default;
53
54          Reaction(Reaction const &reaction, double lambdaRate) {
55              *this = reaction;
56              this->lambdaRate = lambdaRate;
57          }
58      };
59
60      /**
61       * Req. 3
62       * This class allows us to store an instance of the reaction network, using Agent objects,
    ↪in a symbol table
63       * @tparam T
64       */
65      template <typename T>
66      class SymbolTable {
67          std::map<std::string, T> table;
68
69      public:
70          void insert (const std::string &key, const T &value) {
71              table[key] = value;
72          }
73
74          void update (const std::string &key, const T &value) {
75              auto previousValue = get(key);
76              auto newValue = previousValue + value;
77              table[key] = newValue;
78          }
79
80          T& get (const std::string &key) {
81              auto it = table.find(key);
82              if (it != table.end()) {
83                  return it->second;
84              }
85              else {
86                  throw std::invalid_argument("Does not exist");
87              }
88          }
89
90          bool contains (const std::string &key) const {
```

```
91              return table.count(key) != 0;
92          }
93
94          void remove (const std::string &key) {
95              table.erase(key);
96          }
97
98          void PrintAll () {
99              for (auto val : table) {
100                 std::cout << "Name: " << val.first << " Amount: " << val.second.amount << std::endl;
101             }
102             std::cout << "-------------------------" << std::endl;
103         }
104
105         std::map<std::string, T> fetchTable() {
106             return table;
107         }
108
109         static SymbolTable<T> generateSymbolTable(const std::vector<T>& inputVec) {
110             auto symbolTable = SymbolTable<T>();
111             for (auto &input : inputVec) {
112                 symbolTable.insert(input.name, input);
113             }
114             return symbolTable;
115         }
116
117     };
118
119     /**
120      * Req. 7
121      * This class lets us instantiate a state monitor consisting of several symbol tables.
122      * This allows us to store our reactions over a timespan.
123      */
124     class Monitor {
125         std::map<double, std::vector<SymbolTable<Agent>>> monitorMap;
126     public:
127         Monitor() = default;
128
129         void insert (double &time, SymbolTable<Agent> &table);
130         void fileStream(const std::string& fileName);
131         int estimatePeak();
132         void multiplyH();
133     };
134
135     class Algorithm {
136     public:
137         static double computeDelay(stochastic::Reaction &r, stochastic::SymbolTable<Agent>& table);
138         static bool amountChecker(const stochastic::Reaction& reaction,
    ↪stochastic::SymbolTable<Agent>& table);
139         static stochastic::Monitor simulation(std::vector<stochastic::Reaction> &reactionVec,
    ↪double endTime, std::vector<stochastic::Agent> &agentVec, const std::string& filePath);
140     };
141
142     class Visualizer {
143     public:
144         //Req. 2
145         static void prettyPrintReactions(const std::vector<Reaction> &reactionVec);
146         static void generateNetworkGraph(const std::vector<Reaction>& reactionVec, const
    ↪std::string &filePath);
147     };
148
```

```
149 }
150
151 #endif //EXAM_LIB_H
```

1 Outputs

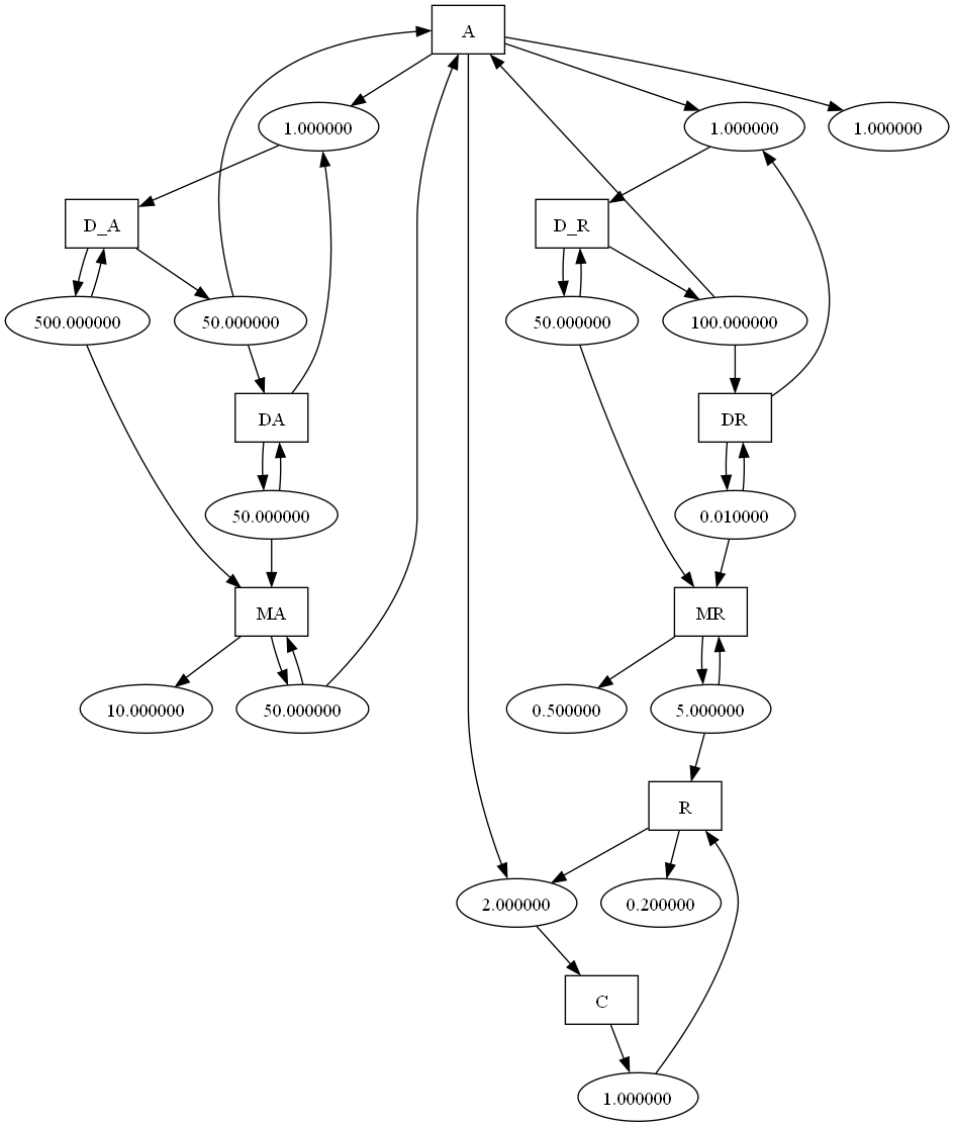

Figure 1: Network graph of the circadian oscillator example

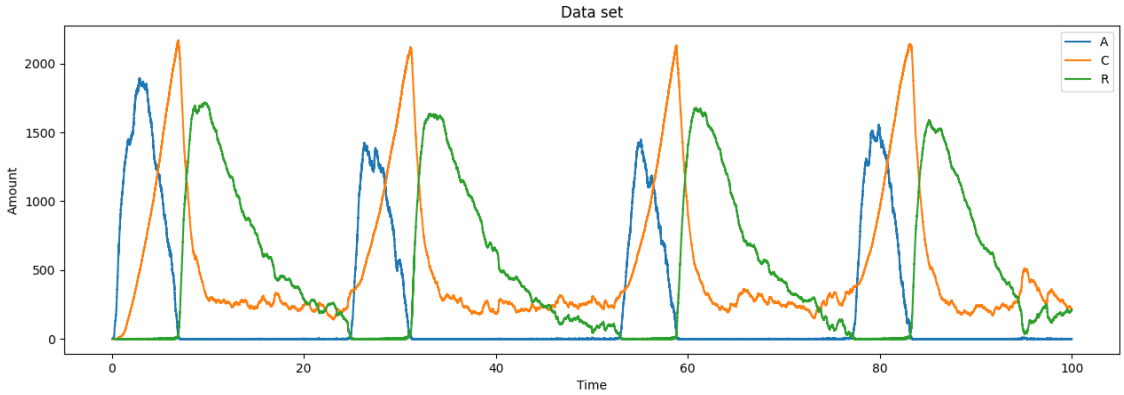

Figure 2: Output plot of the circadian oscillation example

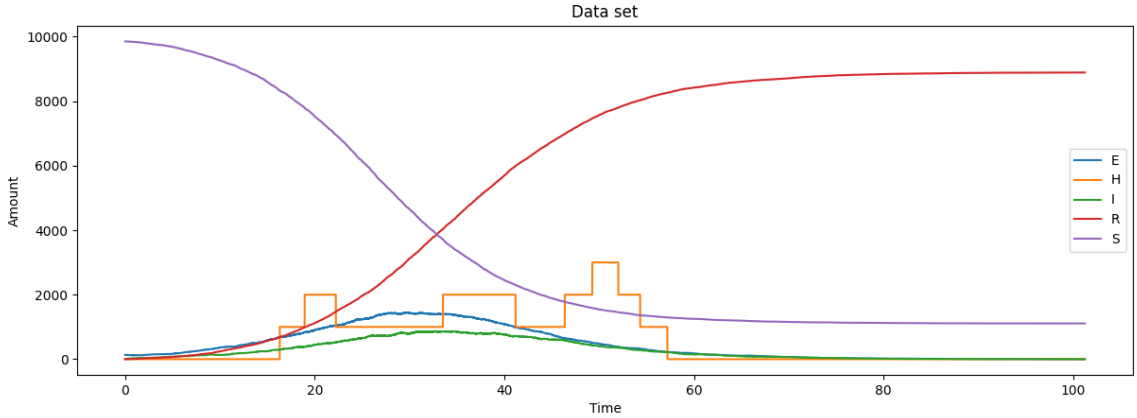

Figure 3: Output of the covid 19 exmaple with N = 10.000

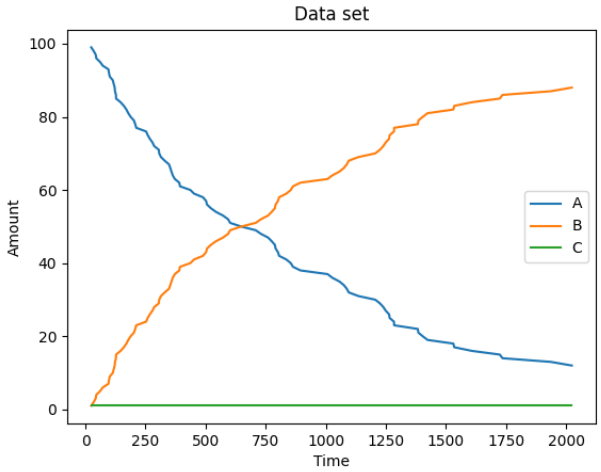

Figure 4: Output of figure 1, where A=100, B=0, C=1

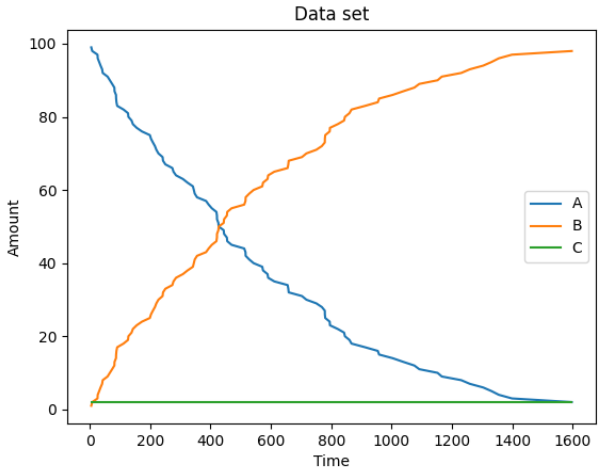

Figure 5: Output of figure 1, where A=100, B=0, C=2

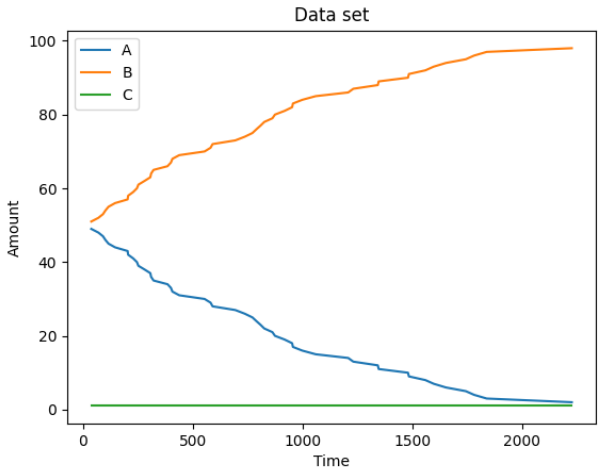

Figure 6: Output of figure 1, where A=50, B=50, C=1