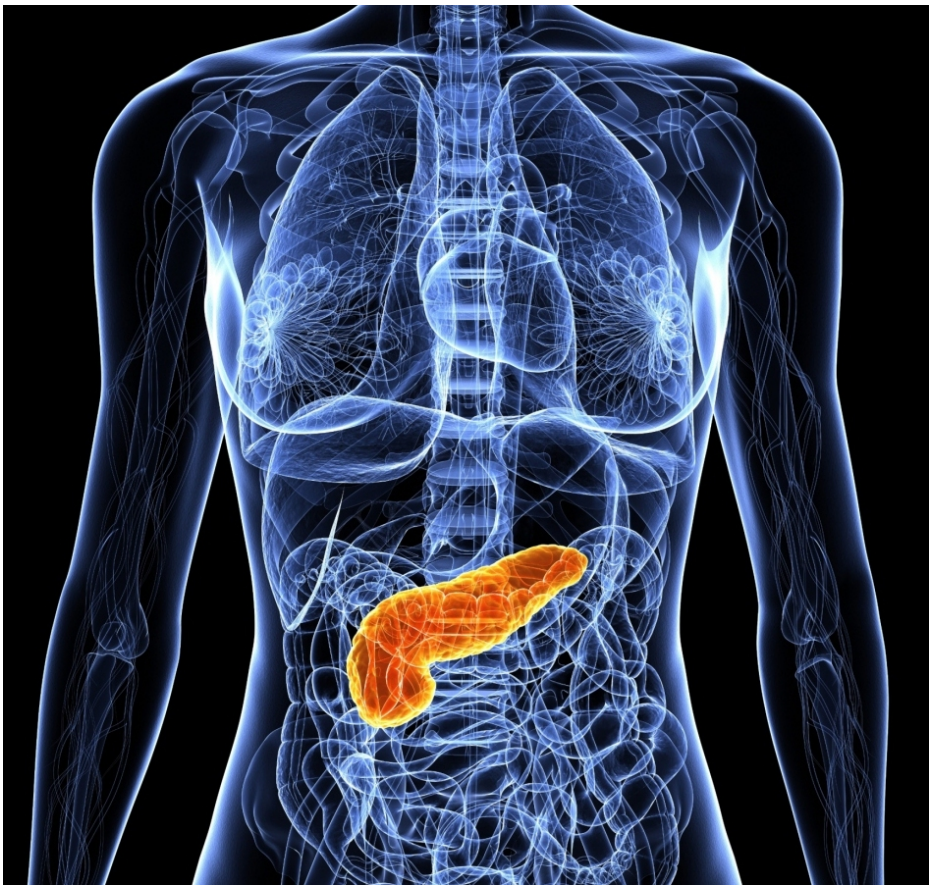
Software for in Silico Testing of an Artificial Pancreas

Master's Thesis



By Stefan Høngaard Andersen

DTU

Student id:	s082878
Supervisor:	John Bagterp Jørgensen
Institute:	DTU COMPUTE
Submission Date:	30-05-2014

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Matematiktorvet, building 303B,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3351
compute@compute.dtu.dk
www.compute.dtu.dk
Frontpage picture: http://healthhub.brighamandwomens.org/

Abstract

Tight glycemic control of people suffering from Type 1 diabetes remains a challenge despite recent developments within diabetes treatment. A fully automated closed-loop Artificial Pancreas (AP) has the potential to become the new standard of care for people with type 1 diabetes. This through a tight glycemic control which resembles the control of a healthy individual. The road towards a commercial-viable AP is long and costly but In silico simulations has the potential to speed up the development of a commercial-viable AP.

The purpose of this thesis is to develop a software and a simulation framework which can become a benchmarking platform and an experimental playground for future in silico testing of the AP. This ensures a consistent testing standard of the AP, while mimicking the setup of clinical trials at a fraction of the cost. The simulation framework developed consist of 6 principal components;

1. A cohort of virtual patients (n = 50 adults) based on actual patient data spanning the observed metabolic variability of key physiological parameters.
2. A mathematical model to describe the glucose-insulin dynamics of a patient suffering from Type 1 Diabetes.
3. A model representing the usage of a Continuous Glucose Monitor.
4. A model to represent the usage of a subcutaneous insulin pump.
5. The implementation of an insulin administration strategy.
6. A set of outcome metrics allowing investigation of the performance of the AP.

These 6 principal components provides a comprehensive in silico simulation framework for closed-loop testing. Thus, a time- and cost-effective tool for performance testing of various APs tested in various user-defined scenarios prior to their clinical implementation.

The software has been implemented in the programming language Java using the open source numerical library Apache Commons Mathematics. The software has been constructed such that execution from Matlab is possible. Additionally, the software has been optimised to ensure fast runtime by the implementation of multi-threading.

The simulation framework would need future clinical validation. However, the findings from this thesis show that realistic computer simulations can provide valuable information about safety and the limitation on closed-loop control systems in a time- and cost-effective way. Hereby, the simulation software developed in this thesis serves as a strong tool in guiding and assessing clinical studies, why this thesis brings an efficient testing tool to the development of the AP.

Preface

This Master's thesis was conducted by Stefan Høngaard Andersen at the department of Applied Mathematics and Computer Science (DTU COMPUTE) at the Technical University of Denmark (DTU) in fulfilment of the requirements for acquiring a Master's degree in Mathematical Modelling and Computation.

The project is written under supervision of Associate Professor John Bagterp Jørgensen. The project was carried out in the period December 2013 to May 2014. The workload of this project is equivalent to 30 ECTS.

Lyngby, 30-05-2014

Stefan Høngaard Andersen

Acknowledgements

First, I would like to thank John Bagterp Jørgensen for proposing this project to me. Without Johns input and support the project would never have been possible.

Bernd Dammands valuable insight in High Performance Computing has been very motivating and I want to thank him for his guidance and interest in the project. In addition, the several employees at DTU High Performance Computing also deserves a thank for the assistance throughout the project.

Ian Jørgensen[1] and Lars Norbert Petersen[2] has both been very helpful in their respective area of knowledge. Ian is an outstanding graphics expert and has given me valuable sparring in the development of plots. Lars is a control guru and has been a valuable help within the area of control systems.

My good friends Søren Olofsson[3] and Nicolai Troensegaard[4] also deserves a thanks. Each has assisted me with insight and discussions in their respective field of expertise. Søren has been a skilled support in the domain of computer science while Nicolai has been a great support in report writing and structuring.

Finally, I would like to thank my girlfriend Stephanie for an invaluable support throughout the project.

Thank you

[1] Linehq.com
[2] PhD-student at DTU
[3] Netcompany A/S
[4] A.T Kearney A/S

Abbreviations

ACM	Apache Commons Mathematics
ADRR	Average Daily Risk Range
AP	Artificial Pancreas
BG	Blood Glucose
BW	Body Weight
CGM	Continous Glucose Monitor
CR	Insulin-to-Carbon Ratio
CSII	Continuous Subcutaneous Insulin Infusion
CSV	Comma Separated Values
HBGI	High Blood Glucose Index
IVP	Initial Value Problem
JDRF	Juvenile Diabetes Research Foundation
JVM	Java Virtual Machine
LBGI	Low Blood Glucose Index
MBG	Mean Blood Glucose
ODE	Ordinary Differential Equation
PID	Proportional Integral Derivative
RK	Runge-Kutta
SMBG	Self Monitoring of Blood Glucose
T1D	Type 1 Diabetes
T1DMS	Type 1 Diabetes Metabolic Simulator
UML	Unified Modelling Language

Bold symbol indicates a vector.
$(\pm SD)$ after a number indicates the standard deviation.

Contents

Chapter 1

Introduction

Diabetes is a disease affecting over 300 million people worldwide of which approximately 10% suffers from Type 1 Diabetes (T1D) [17]. T1D is characterized by the total absence of endogenous insulin production and consequently T1D patients fully rely on external insulin for survival. A potential future treatment solution for T1D patients is an AP. An AP is a system which mimics the behaviours of a human pancreas. This ensures automatic continuous insulin infusions with little or no human interference. The three principal components of the AP are: 1) a glucose monitor which ensures real-time glucose measurements from the subcutaneous tissue; 2) an insulin pump which ensures insulin injections subcutaneously; and 3) a control system that decides when and how much insulin to infuse in the patient suffering from T1D. The AP is still not commercialised. However, trends within the field of the AP points in the direction of better and more accurate Continues Glucose Monitors (CGM), Continuous Subcutaneous Insulin Infusion (CSII) pumps and more efficient control systems. This will potentially lay the foundations for a commercial-viable AP within a few years and consequently a potential new standard of care of T1D.

Pre-Clinical and clinical trials with an AP have already been undertaken and several studies suggests that the AP is effective in managing a patient suffering from T1D [38, 40]. Much progress has been made but more intensive testing is essential before the AP can become commercially-viable. Pre-clinical and clinical trials are very comprehensive with respect to both time and costs though

an essential part of proper evaluation and documentation of any medical device. A cost-effective solution which can speed up the process towards a commercial-viable AP is needed. In silico[1] testing could accelerate the development of the AP in two ways. First, by substituting pre-clinical studies. Second, by predicting the outcome of real life clinical trials and consequently assist in the design of actual clinical studies [19]. Hereby, in silico testing provides valuable information at a fraction of the time and cost compared to conventional pre-clinical and clinical studies. Thus, in silico testing could serve as a tool which can speed up the process towards a commercial-viable AP .

The DIACON Group is a Copenhagen based research consortium consisting of researchers from the Technical University of Denmark, Hvidovre Hospital and diabetes related companies. The DIACON group's focus is to develop technologies that can support the treatment for T1D. This thesis will focus on the development of software specifically for research conducted by the DIACON group. The software can potentially be used as a substitute for pre-clinical trials and as a support in predicting the outcome of actual clinical experiments. The software will be independent such that no interference with intellectual property rights in commercial contexts will appear. Consequently, this software enables experimental testing of user-defined scenarios for in silico testing of the AP.

The subsequent chapters presents the problem statement, the problem scope, a conceptual clarification and finally the contributions of this thesis.

1.1 Problem Statement

The aim of this thesis is to develop a software which can resemble the usage of an Artificial Pancreas and simulate the glucose-insulin dynamics of a virtual cohort. This will be done by implementing a proof-of-concept Artificial Pancreas consisting of a glucose monitor, an insulin infusion pump and a simple control algorithm.

In order to ensure that the simulation will be realistic the virtual cohort must be representative of the target population. Thus, the software should introduce an experimental playground where virtual subjects can be created and modelled on a user-defined basis.

Several performance metrics will be established to enable easy evaluation across tests. The software should be build platform and license independently such

[1] Test executed in a virtual computer environment.

that the software in commercial contexts does not interfere with intellectual property rights. As part of being time-efficient the software must also be able to conduct the simulation in a few minutes.

1.2 Problem Scope

The subject of simulation is broad and includes many areas of research. Therefore, it is important to narrow the subject further. This thesis seeks to develop and implement a simulation framework for in silico testing of an AP consisting of the following 6 principal components:

1. A cohort of virtual patients.
2. A mathematical model to describe the glucose-insulin dynamics of a T1D patient.
3. A model representing the usage of a Continuous Glucose Monitor.
4. A model to represent the usage of a subcutaneous insulin pump.
5. The implementation of an insulin administration strategy, also denoted as a controller.
6. A set of performance metrics to evaluate the performance of the AP such that different insulin administration strategies can be compared.

The software will be developed in Java. Java has been chosen for two reasons. First, Java is a high-level programming language which makes implementation far more simple compared to low-level language such as C. Second, Java is platform independent such that the software can be used across platforms. This means that the software developed can be used on both Windows and Linux [2]. To deviate from interference with intellectual property rights in commercial contexts, the software will be based on open source numerical algorithms.

This thesis will limit its focus on three aspects. First, focus will not be on control systems and the theory related to this field. However, a proof-of-concept scenario with the implementation of a simple PID (Proportional Integral Derivative) controller will be made. Second, this thesis will not focus on creating a Graphical User Interface for the interaction with the software. Both subjects will be entrusted to future research. Third, this thesis will consider the implementation of a single-hormone physiological model.

[2] This includes derivative platforms from linux such as Mac OS or UNIX.

1.3 Structure of the Report

This thesis has been written such that the reader can follow the string of thoughts and arguments in reaching the goal of developing a feasible and usable software for in silico testing of an AP.

The thesis is organized as follows. Chapter 2 presents the basic physiological dysfunctionalities in the human body for people with T1D. Additionally, Chapter 2 gives an overview of the current treatment available and the context of the AP. The same chapter also looks into a selected literature review.

Chapter 3 and Chapter 4 describe the theory and methods used throughout this report. Chapter 3 presents the mathematical model used to describe patients with T1D. Additionally, Chapter 3 defines the virtual cohort and the performance metrics used to evaluate the in silico tests. Chapter 4 explains and justifies the various numerical methods used in the software including the insulin administration strategy implemented in this software.

Chapter 5 focuses on the development of the software. This includes the implementation of the theory and methods from Chapter 3 and Chapter 4 in the programming language Java. Chapter 6 presents and discusses tests of the software and the results obtained using the software. Chapter 7 rounds off this thesis with a perspective on the project and ideas for future work.

In Chapter 8 a conclusion on the thesis findings is presented.

1.4 Conceptual Clarification

This section seeks to give a brief clarification of the concepts used throughout this chapter.

A *system* is a collection of *entities* as proposed by Schmidt and Taylor in [39]. The entities in this system are: The mathematical physiological model, the CGM, the inuslin pump and the controller. The system will receive some *input* or *events*, evaluate changes and return an *output* as shown in Figure 1.1. In this thesis the input are made up of a meal intake, d, the insulin infusion, u, and the *virtual cohort*. The virtual cohort consist of a subset of *virtual patients* each composed by a set of *physiological parameters*, $\boldsymbol{\theta}$. The output of the system is an evaluation of the system performance. The system which we seek to analyse is a *continuous* system, that is, a system which changes continuously with respect

Figure 1.1: *This figure shows the conceptual understanding between input, system and output that will be used throughout this thesis.*

to time.

The *actual* system is one or more patients suffering from T1D currently complying treatment with an AP. Even though testing on the actual system is possible it comes with a large time and economic burden. On the contrary a *model*, which is used to describe the actual system, can test the system in a cost and time efficient manner. The model is composed of *state variables*, which indicates the system states over time. By numerically challenging the model with various input it is possible to *simulate* the output performance of the actual system.
As a final remark is the usage of *commercial viability* and *standard of care*. Standard of care refers to an authority-approved and usable treatment option for T1D. Additionally, the treatment has become the actual preferred standard by professionals in treating patients suffering from T1D. For a treatment to become commercial viable means that the treatment is competing effectively with other existing treatment options *and* are profitable.

1.5 Contributions

This thesis contributes to the DIACON Group and their research by developing a cost- and time-efficient simulation framework, as depicted in Figure 1.2. The simulation framework seeks to predict the outcome of actual clinical studies by:

1. Developing a data management system that enables creation of virtual cohorts which are representative of a target population.
2. Implementing open-source numerical methods for simulation purpose while refraining from conflict with intellectual property rights.
3. Developing an analytical tool package for evaluation of in silico testing of an AP.

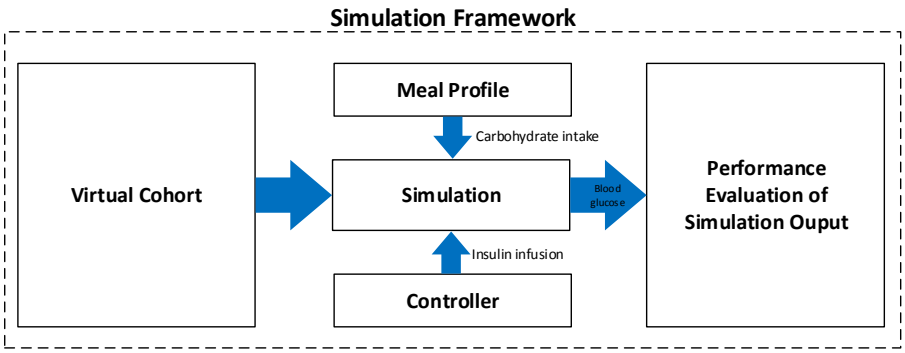


Figure 1.2: *This figure shows the schematic representation of the contribution to the DIACON groups research. The simulation framework is a cost- and time-effective solution for designing and testing components of the AP, hereby predicting the outcome of actual clinical studies.*

Chapter 2

Background

The purpose of this chapter is to give an overview of the problem domain of this thesis. The chapter is organized as follows. Section 2.1 is dedicated to an introduction of the biological dysfunctionalities related to diabetes. Section 2.2 looks into the current treatment options and technologies available. Finally, Section 2.3 presents a selected review on the AP project.

2.1 Diabetes

Diabetes is a dysfunctionality of the body's regulation of blood sugar levels. Diabetes is conventionally divded into two types: Type 1 and Type 2 diabetes (T1D and T2D). T1D is characterized by the total absence of insulin production in the pancreas [8]. Consequently, patients suffering from T1D must administer the infusions of external insulin themselves. In contrast, people suffering from T2D still produce insulin in the pancreas but typically suffers from insulin intolerance, i.e. insensitivity to insulin. This means that the body's natural insulin production is insufficient to regulate the blood sugar level.
T1D will be the focus of this thesis. Figure 2.1 shows the carbohydrate metabolism in the human body. The carbohydrate metabolism has two main functionalities. First, it ensures a constant supply of energy to the cells in the human body. Sec-

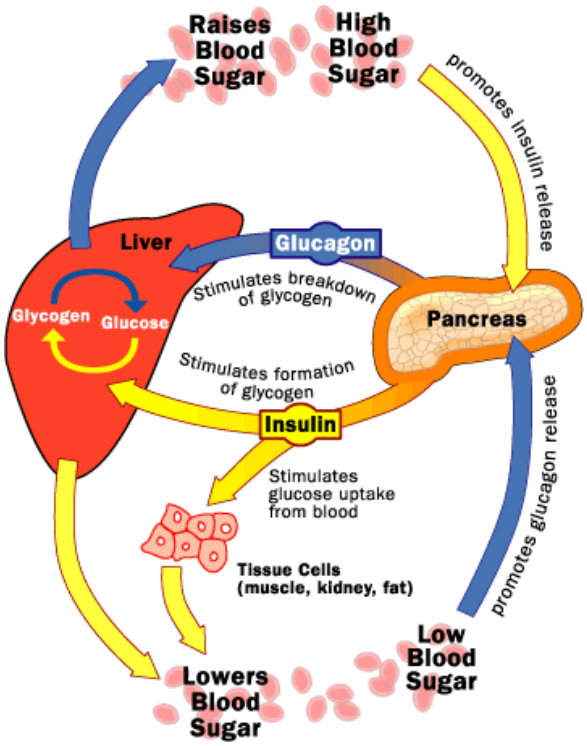


Figure 2.1: *The carbohydrate metabolism in a healthy individual. The disturbances in the glucose level is counteracted by the action of the pancreas [16].*

ond, the carbohydrate metabolism ensures that the blood glucose (BG) levels in the human body stabilizes after external influences such as food intake, fasting or exercise [13]. Insulin and glucagon are the two key hormones regulating this system. They ensure that the BG level is kept within a range between 4-10 mmol/L [13]. If the BG level drops below normal, α-cells in the pancreas release glucagon. This stimulates the release of glucose from the liver to the blood stream, and thus normalises the BG level. If the BG level is too high, β-cells in the pancreas secrete insulin which stimulates the uptake and storage of glucose in several cell types in the body, thus decreasing the BG level. The glucose metabolism hereby works as a physiological feedback mechanism ensuring that the BG levels will stay within a range of 4-10 mmol/L while providing the human body with a constant supply of energy. As the β-cells in T1D patients are destroyed, the patient consequently have no insulin secretion. This leads to a dysfunctional glucose metabolism with highly variable BG levels. Thus, T1D patients are fully dependent on external insulin injections for survival. The Diabetes Control and Complication Trial Research Group (DCCT) has identified

two important factors to minimize long term complications for T1D patients. The first factor is to keep the BG levels within the euglycemic range, i.e. 4-10 mmol/L. The second factor is to lower the glycemic variability, that is, fluctuations between lows and highs [23]. Hence, the DCCT denotes that the tighter BG control a T1D patient can achieve, the fewer complication can be expected. BG levels outside the euglycemic range can lead to severe complications. A too low BG level, called Hypoglycemia, may incur seizures or coma which may lead to brain damage and ultimately death. This is due to a lack of energy supply to the nerve cells in the human body. A too high BG level, called Hyperglycemia, leads to more subtle and long term complications. This includes but is not limited to vascular diseases, vision disorders, nerve degeneration and the like. People suffering from T1D have a three to five fold risk of suffering from such complications compared to healthy people [23].

2.2 Diabetes Treatment

The previous section determined that patients suffering from T1D are in constant need of external insulin to survive. Additionally, compliance with the treatment and tight glycemic control are important elements in reducing the complications related to T1D. This section will look into some of the treatment options available for T1D.

2.2.1 The Current Treatment Solutions

Tradional T1D treatment is done by regular Self Monitoring of Blood Glucose (SMBG) and insulin injections. SMBG is usually done with a finger stick and gives a measurement of the BG level. The SMBG assist the T1D patient in determining the BG level such that a decision on insulin injections can be made. Patients suffering from T1D needs to regulate their BG levels often to mimick the function of their pancreas [8]. This is usually done by combining rapid and slow acting insulin. During morning hours, the slow acting insulin is injected such that a constant basal rate of insulin is ensured throughout the day. During meals the rapid acting insulin is injected to compensate for the carbohydrates in the meal intake. Compliance and tight BG control with this technique can be challenging. As a tight control of BG is important for T1D patients, more patients starts combining Continous Glucose Monitors (CGM) and Continous Subcutaneous Insulin Infusion (CSII) pumps to dose and infuse insulin during the day [42]. The CGM ensures continuous readings of the BG level while the CSII ensures the infusion of insulin to the body. Figure 2.2 gives an example of

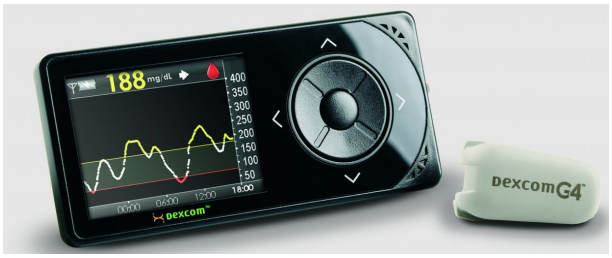


Figure 2.2: *The Dexcom G4® Continous Glucose Monitor assist diabetics monitor their blood glucose in real-time [6].*

Figure 2.3: *The ACCU CHEK® insulin pump which contains the insulin in a small compartment. Insulin dosage will be decided by the patient [2].*

a CGM, the Dexcom G4 CGM. CGMs are less accurate than SMBG and must be calibrated several times a day. For patients with CGMs, the SMBG is used for calibration of the CGM signal. The CGM sensor detects the glucose level in the subcutanoues tissue which leads to a signal lag of approximately 10-15 minutes [32].

Insulin pumps allow the patient to control the dose of insulin to infuse. Many factors influences the dosage of insulin and modern insulin pumps usually considers parameters such as the insulin-to-carbohydrate ratio (CR), insulin sensitivity and recent bolus history to estimate the recommended insulin bolus. Figure 2.3 shows an example of a CSII pump, the Roche CSII pump [2]. The CR is the ratio describing the amount of insulin needed to cover the carbohydrate intake for a meal while the insulin sensitivity is a measure of the patients responsiveness to insulin. Parameters like these are known to vary significantly between patients and they are also known to vary within patients over time.

Figure 2.4: *The Medtronic Minimed® pump with the Enilte CGM. The system monitors blod glucose and previous injections and assist the T1D patient in deciding insulin dosage [11].*

Nevertheless, commercialisation of basic diabetes management systems already exist. These systems can take some decisions on behalf of the T1D patient. An example of such a system can be seen in Figure 2.4 which is the Medtronic Minimed® insulin pump with an Enlite® sensor. The system can assist with basic recommendations and disable the infusion of insulin if the BG level becomes lower than a threshold value. The Medtronic Minimed® automates only some elements of the diabetes management and accordingly the patient still need to be part of the daily treatment [11].

2.2.2 An Artificial Pancreas

An Artificial Pancreas combines the use of a CGM, a CSII pump, and a control algorithm to automatically control the BG levels of a T1D patient. This is illustrated in Figure 2.5. Hereby the AP serves to substitute the functionality of a healthy pancreas. As denoted in Figure 2.1 the pancreas has several important functionalities but it is the absence of insulin production which the AP seek to substitute. Development of two-hormone AP also exist and includes the usage of glucagon as a counterbalance to the effect of insulin [38]. Whether the AP uses one or two hormones the goal with an AP is, first, to improve compliance with diabetes treatment. Second, to ensure glycemic control such that it resembles the one of a healthy individual. Third, to improve the quality of life for patients with T1D.

A key challenge in making an AP successful is to accommodate the differences between the individual patients. Elements such as meals, stress, exercise and

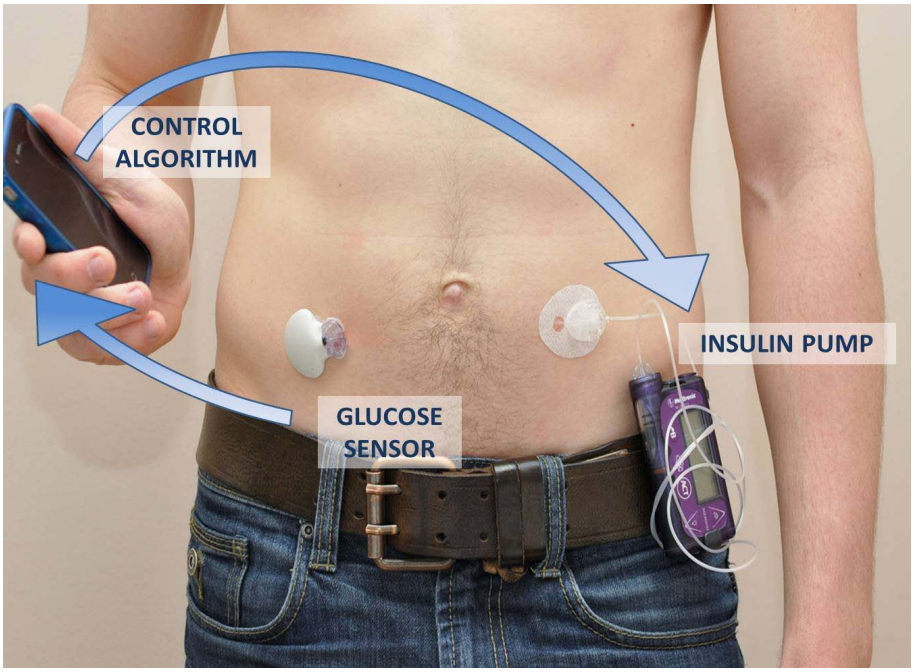


Figure 2.5: *This figure shows the components of the Artificial Pancreas. In this picture a Medtronic device is used as glucose sensor and insulin pump. An iphone 4s is schematised to contain the control algorithm. It could also have been embedded in the pump [29].*

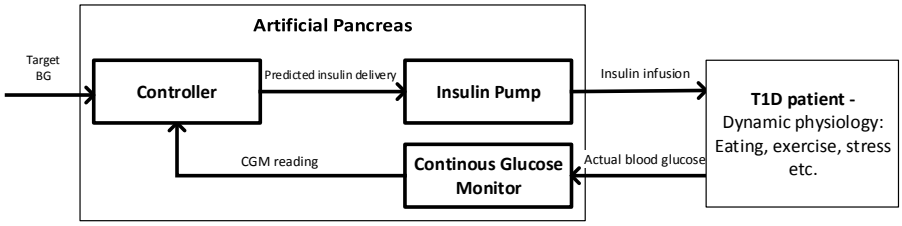


Figure 2.6: *This figure represent the closed-loop system of the Artficial Pancreas.*

other life events are known to impact the BG levels very differently across individuals [18, 28]. The AP must take this variability into consideration by a continuous evaluation of the patient. Figure 2.6 identifies the main loop of the AP. The system is said to be closed-loop. This indicates that the system receives continuous feedback from the patient and continuously adjust the system output accordingly. The system output in the AP is the insulin to be delivered.

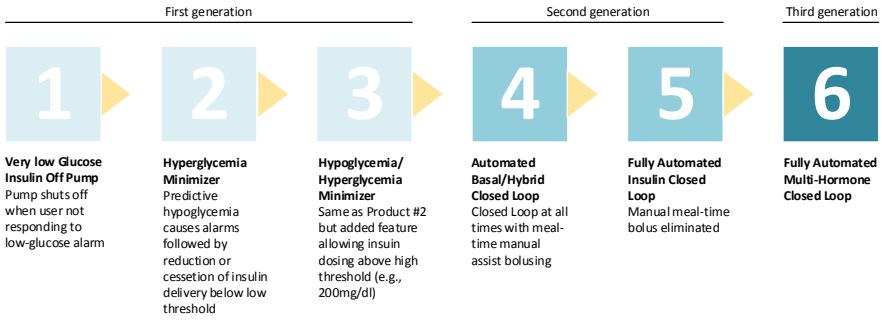


Figure 2.7: *The 6 step plan proposed by the JDRF [8].*

2.3 The AP progress and related work

The Juvenile Diabetes Research Foundation (JDRF) has recently been very focused in their funding of research and development within the to field of the AP. In 2006 the JDRF launched the Artificial Pancreas Project to [8]:

"*accelerate the development of a commercially-viable artificial pancreas as a system to ultimately mimic the biological function of the pancreas for patients with type 1 diabetes.*"

The JDRF denoted a 6 step plan towards a commercial-viable AP. The 6 step plan can be seen in Figure 2.7. The 6 step plan proposed a first generation AP which would prevent hyper- or hypo- glycemia by causing alarms or ultimately shutting off the insulin pump. The first generation AP is largely scoped by the Medtronic Minimed® as reviewed in Section 2.2. The second and third generation AP propose the step towards fully automated solutions. Clinical trials are being conducted these years with focus on second and third generation AP [38, 40]. A lot of this research is conducted in the JDRF established consortium of university centres[1] to conduct closed-loop glucose research [8, 7]. This consortium drives a large part of the research currently being conducted in the research topic of the AP.

The JDRF has identified that one of the main challenges with the AP project is that the current CGM accuracy and reliability is insufficient for ensuring a fully automated closed-loop AP [8]. A recent review by Damiano et al. in [20] compared three commercially available CGMs and concludes a CGM mean

[1] The JDRF Consortium consist of, e.g., University of Virginia, University of Boston, and University of Cambridge.

absolute relative error between $10 - 15\%$. The CGM inaccuracy is considered a major challenge in the AP development and consequently a lot of attention has been brought to in silico testing of the inaccuracy of the CGM. Kovatchev and Breton gives a mathematical model which can simulate the inaccuracy derived from the CGM sensor noise, thus, testing the controller using noised CGM readings [32]. Dalla Man et al. compare this model with a real sensor noise from a FreeStyle Navigator® CGM and concludes that no significant difference exist between the simulated and the real distribution of the sensor error [19]. Conducting experiments in silico is becoming frequently more used. Cobelli et al., Hovorka et al. advocate that the introduction and development of in silico methods will speed up the commercialisation of the AP [18, 28]. The current physiological models available is feasible to simulate T1D patients and to asses the performance of control algorithms. However, they are not sufficient to describe inter individual variability. For this reason further research within the field of in silico simulation systems is essential for further development of control algorithms [28].

2.3.1 In Silico Simulation Systems

In this section we review a selected list of in silico simulators currently available on-line.

AIDA Diabetes Simulator

An early attempt to create an AP simulator is the interactive simulator AIDA[2] [3]. AIDA is a patient simulator which makes type 1 diabetes simulation possible. The AIDA simulator is simple and has a user-interface which makes the modification of meal input and insulin types (slow vs. rapid acting insulin) easy. However, AIDA neither enables the implementation of various insulin administration strategies nor does it allow variations in patient physiology. This means that AIDA is very usable for illustrative and educational purposes but does not serve as a testing facility for control systems. The software is freely available from AIDA's homepage.

[2] URL: `http://www.2aida.org/`

Maxsim2

A recent development from the University of Chalmers is the Maxsim2[3] [10]. Maxsim2 has a very specific purpose, which is to simulate pharmaco-kinetic and -dynamics in various testing environments such as animals and humans. This software can be used to compare different scenarios of new pharmaceuticals but cannot be used in a full AP simulation study.

Type 1 Diabetes Metabolic Simulator

In a recent contribution from the Epsilon Group, the development of the Type 1 Diabetes Metabolic Simulator[4] (T1DMS) enables testing of insulin administration strategies and various patient variabilities [12]. The T1DMS has been approved by the Food and Drug Administration as a substitute for pre-clinical animal studies for glucose control scenarios. For this reason the T1DMS serves as a strong tool to asses and evaluate proposed insulin administration strategies considering the inter individual variability. As expressed by Dalla Man et al. in [19] software of this kind sets a precedent for future preclinical studies thus leading to the paradigm that [19]:

1. *"In silico modelling could produce credible preclinical results that could substitute certain animal trails, and*

2. *in silico testing yields these results in a fraction of the time required for animal trials."*

The software is developed with the main focus to reduce the need for animal trials such that new AP systems can be tested in a much faster and more convenient way. T1DMS describes much of the inter individual variability which exist between patients [12]. The T1DMS is not freely available. Furthermore, T1DMS builds on Matlab's Simulink® network making the installation of Matlab® a prerequisite.

[3] URL: `http://www.maxsim2.com/`
[4] URL: `http://tegvirginia.com/solutions/t1dms/`

2.4 Summary

In this chapter we have provided a brief background introduction to the dysfunctionality in the human body in people suffering from diabetes. It was found that T1D patients needed continuous insulin administration for survival and that continuous BG monitoring was necessary to obtain proper treatment. Compliance and tight glycemic control were the most important elements in diabetes treatment. One way of obtaining such treatment would be through the AP. The AP would completely substitute human intervention in the treatment of T1D. Lastly, a literature review confirmed that a lot of attention exist in the development of the AP and that a simulation tool is considered very usable in the road towards a commercial-viable AP. The development and testing of such a simulation tool will be the focus of the remaining part of this thesis.

Chapter 3

Physiological Model

The objective of this chapter is twofold. First, we introduce the physiological model used to simulate a virtual patient. This includes the implementation of a CSII pump, a CGM and a definition of a virtual cohort. Second, the objective is to define a set of performance metrics to evaluate simulations. This chapter is organised as follows. Section 3.1 presents the physiological model used to represent the glucose-insulin dynamics in a T1D patient. Additionally, Section 3.1 describes the insulin absorption by using a CSII pump and also establish a virtual cohort. Section 3.2 presents the CGM model used. Finally, Section 3.3 explains the performance metrics applied to evaluate simulations.

3.1 The Hovorka Model

In recent years, many physiological models have been developed with the purpose of describing the glucose-insulin dynamics of a human body. In this section, we present the model developed by Hovorka and Wilinska and revised by Boiroux [16, 25]. The model is used in this thesis to determine the glucose-insulin dynamics of a virtual patient suffering from T1D. The model will be referred to as "the Hovorka model".

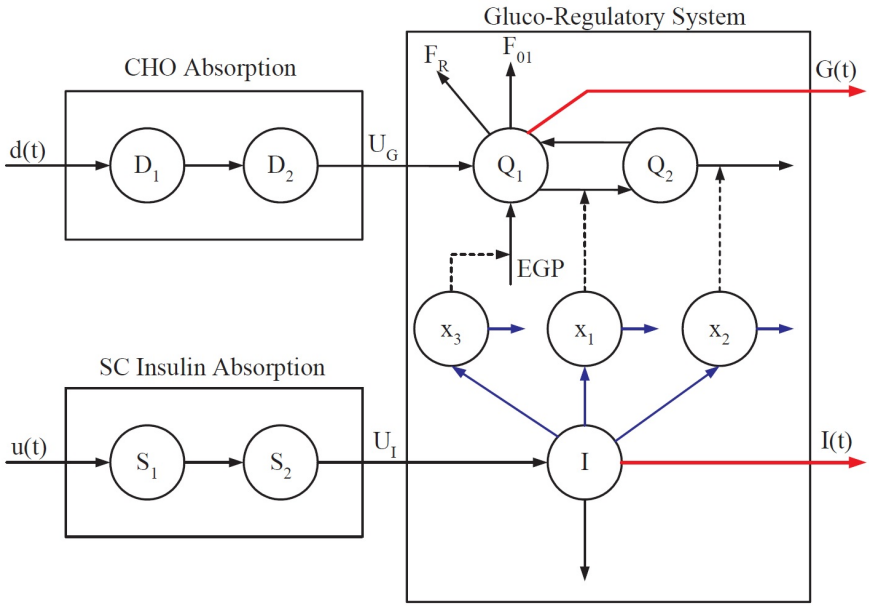


Figure 3.1: *A compartment diagram of the Hovorka model [16].*

The model is a nonlinear system of differential equations and can be summarized by the compartment diagram in Figure 3.1. The model consist of three subsystems; the carbohydrate absorption, the subcutaneous insulin and the glucose-insulin regulatory system. There are two external inputs: The oral intake of carbohydrate and the subcutaneous insulin injections. The model has two external outputs: The BG level and the plasma insulin level. The model describes accurately the glucose-insulin dynamics of the human biological system [25].

3.1.1 Carbohydrate Absorption

The meal input, $d(t)$ [g/min], indicates the intake of carbohydrates per unit of time. The absorption and conversion of the carbohydrate from the oral food intake to the glucose is modelled by a two-compartment model:

$$\frac{dD_1}{dt}(t) = AgD(t) - \frac{D_1(t)}{\tau_D} \tag{3.1}$$

$$\frac{dD_2}{dt}(t) = \frac{D_1(t)}{\tau_D} - \frac{D_2(t)}{\tau_D} \tag{3.2}$$

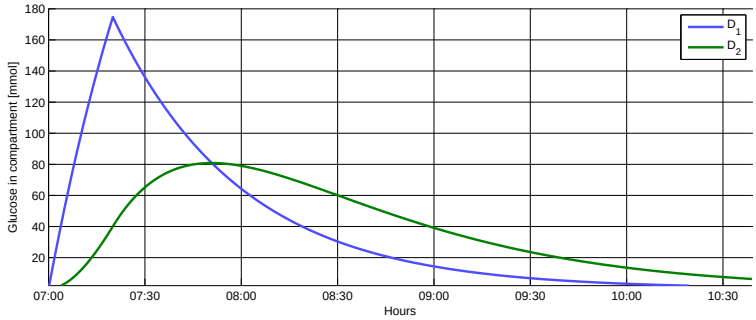


Figure 3.2: *This figure show the glucose response of 25g of carbohydrate ingested between 07:00 and 07:20.*

$D(t)$ [mmol/min], denotes the oral intake of carbohydrate in glucose equivalents and is defined as $D(t) = \frac{1000}{M_{wg}} \cdot d(t)$, where $M_{wg} = 180.16$ is the molecular weight of glucose. $D_1(t)$ [mmol] and $D_2(t)$ [mmol] denote the amount of glucose in compartment one and compartment two at time t, respectively. A_g is a utilization factor of the absorption of carbohydrate to glucose and τ_D [min] is a time constant. The representation of the absorption of carbohydrate in the two compartments can be seen in Figure 3.2. The rate of appearance of the absorption of glucose in the blood stream, U_G [mmol/min] is defined as:

$$U_G(t) = \frac{D_2(t)}{\tau_D} \tag{3.3}$$

Figure 3.3 shows the absorption rate from the gut for three different meal profiles ingested at 07:00-07:20.

3.1.2 Subcutaneous Insulin Absorption

The external insulin injected by a CSII pump, $u(t)$ [mU/min], is modelled by absorption through two compartments, $S_1(t)$ [mU] and $S_2(t)$ [mU]:

$$\frac{dS_1}{dt}(t) = u(t) - \frac{S_1(t)}{\tau_S} \tag{3.4}$$

$$\frac{dS_2}{dt}(t) = \frac{S_1(t)}{\tau_S} - \frac{S_2(t)}{\tau_S} \tag{3.5}$$

$$\tag{3.6}$$

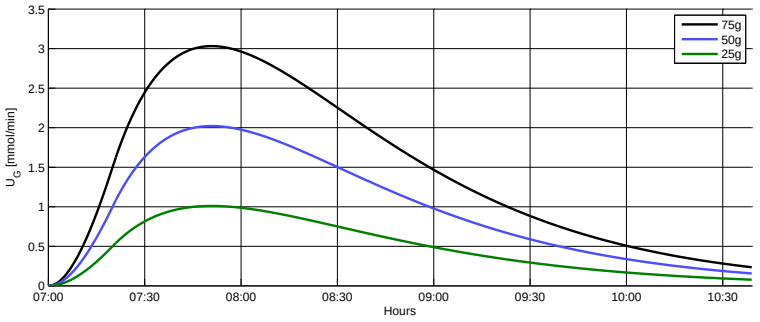


Figure 3.3: *This figure show the glucose absorption from the gut with 25g, 50, and 75g of carbohydrate ingested between 07:00 and 07:20.*

τ_S [min], is a time constant. The absorption rate of insulin in the blood stream, $U_I(t)$ [mU/min], is given by:

$$U_I(t) = \frac{S_2(t)}{\tau_s} \tag{3.7}$$

The dynamics for the plasma insulin concentration, $I(t)$ [mU/L], is given by:

$$\frac{dI}{dt}(t) = \frac{U_I(t)}{V_I} - k_e I(T) \tag{3.8}$$

V_I [L], is the insulin distribution volume and k_e [min^{-1}], is an insulin elimination rate. Figure 3.4 illustrates the response of three subcutaneous insulin boluses on top of a basal injection in the human body.

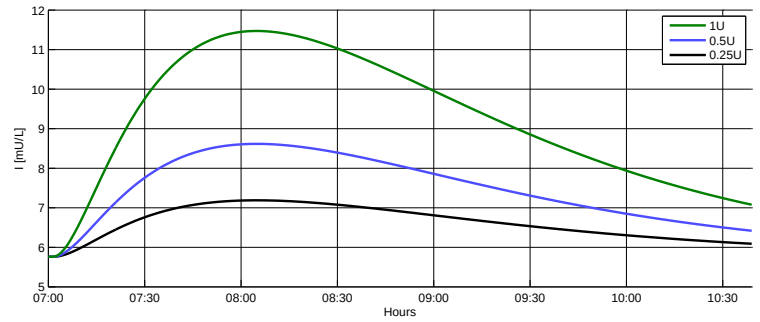


Figure 3.4: *This figure show plasma insulin concentration for a insulin injection at 07:00 of* $1U$, $0.5U$ *and* $0.25U$ *respectively on top of a basal injection.*

3.1.3 Gluco-Regulatroy System

The absorbed glucose distributes through the gluco-regulatory system as follows. The two state variables $Q_1(t)$ [mmol] and $Q_2(t)$ [mmol] indicate the glucose in the main bloodstream and the peripheral tissue, respectively. The rate of change for compartment one is defined as follows:

$$\frac{dQ_1}{dt}(t) = U_G(t) - (F_{01}^c(t) + F_R(t) + x_1(t)Q_1(t)) + k_{12}Q_2(t) + EGP \tag{3.9}$$

3.9 includes the glucose absorption from the gut, $U_G(t)$, the consumption of glucose by the central nervous system, F_{01}^c [mmol/min], the renal glucose excretion, $F_R(t)$ [mmol/min], the uptake of glucose to the peripheral tissue, $x_1(t)Q_1(t)$ [mmol/min], the transfer of glucose from the peripheral tissue to the bloodstream, $k_{12}Q_2(t)$ [mmol/min], and finally the endogenous release of glucose by the liver, $EGP = EGP_0(1 - x_3(t))$, extrapolated from the zero insulin release of glucose. $x_1(t)$ and $x_3(t)$ are remote effects on the peripheral tissue and the endogenous glucose production, respectively. As the glucose concentration increases in the main bloodstream, the glucose distributes into the peripheral tissue as well. The dynamic system for the peripheral tissue is represented as follows:

$$\frac{dQ_2}{dt}(t) = x_1(t)Q_1(t) - (k_{12} + x_2(t))Q_2(t) \tag{3.10}$$

$x_1(t)Q_1(t)$ [mmol/min], is the uptake of glucose to the peripheral tissue from the bloodstream and, $k_{12}Q_2(t)$ [mmol/min], is the transfer of glucose from the peripheral tissue back to the bloodstream. $x_2(t)Q_2(t)$ [mmol/min] is the movement of glucose out of the gluco-regulatory system. $x_2(t)$ is a remote effect on the disposal of glucose.

In reference to (3.9), the consumption of glucose by the central nervous systems, F_{01}^c, is defined as:

$$F_{01}^c(t) = \begin{cases} F_{01} & \text{if } G(t) \geq 4.5\text{mmol/L} \\ F_{01}\frac{G(t)}{4.5} & \text{otherwise} \end{cases} \tag{3.11}$$

The renal glucose excretion $F_R(t)$ is only present with a glucose level higher than 9 mmol/L:

$$F_R(t) = \begin{cases} 0.003(G(t) - 9)V_G & \text{if } G(t) \geq 9\text{mmol/L} \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

The linkage to the glucose concentration, $G(t)$ [mmol/L], is given by:

$$G(t) = \frac{Q_1(t)}{V_G} \tag{3.13}$$

Table 3.1: *The parameters for the Hovorka model.*

Parameter	Unit	Description
EGP_0	$mmol/min$	Liver production of glucose at zero insulin
F_{01}	$mmol/min$	Non-insulin dependent glucose consumption
k_{12}	min^{-1}	Fractional transfer rate parameter
$k_{a,1}$	min^{-1}	Deactivation rate parameter
$k_{a,2}$	min^{-1}	Deactivation rate parameter
$k_{a,3}$	min^{-1}	Deactivation rate parameter
$S_{I,1}$	$min^{-1}/mU/L$	Transport insulin sensitivity
$S_{I,2}$	$min^{-1}/mU/L$	Disposal insulin sensitivity
$S_{I,3}$	L/mU	EGP insulin sensitivity
k_e	min^{-1}	Elimination rate of insulin
V_G	L	Distribution volume of glucose
V_I	L	Distribution volume of insulin
τ_S	min	Insulin absorption constant
τ_D	min	Carbohydrate time constant
Ag	–	Carbohydrate absorption constant

$Q_1(t)$ is the glucose in the main bloodstream and V_G [L] is the glucose distribution volume. The glucose distribution volume depends on the individual body weight, BW [Kg].

As briefly mentioned in (3.9) and (3.10), the glucose insulin system has three external impacts. These are; the glucose distribution, $x_1(t)$, the glucose disposal, $x_2(t)$, and the endogenous glucose production $x_3(t)$:

$$\frac{dx_1}{dt}(t) = -k_{a1}x_1(t) + k_{b1}I(t) \tag{3.14}$$

$$\frac{dx_2}{dt}(t) = -k_{a2}x_2(t) + k_{b2}I(t) \tag{3.15}$$

$$\frac{dx_3}{dt}(t) = -k_{a3}x_3(t) + k_{b3}I(t) \tag{3.16}$$

$k_{b1} = k_{a1}S_{I1}$, $k_{b2} = k_{a2}S_{I2}$, and $k_{b3} = k_{a1}S_{I3}$. All the parameters in the Hovorka model are summarized in Table 3.1. The entire set of parameters will be refereed to as the vector $\boldsymbol{\theta}(t)$. The vector is time dependent as the physiological parameters will vary over time.

3.1.4 The Hovorka Model as an Initial Value Problem

The state variables in the Hovorka Model are:

$$\boldsymbol{x}(t) = [D_1(t)\ D_2(t)\ S_1(t)\ S_2(t)\ Q_1(t)\ Q_2(t)\ I(t)\ x_1(t)\ x_2(t)\ x_3(t)]' \tag{3.17}$$

Furthermore, the physiological parameters is given by $\boldsymbol{\theta}(t)$, the insulin dosage by $u(t)$ and the meal intake by $d(t)$. The notation for the entire system of

differential equations can be defined as an Initial Value Problem (IVP) as seen in (3.18):

$$\begin{aligned} \boldsymbol{x}(t_0) &= \boldsymbol{x_0} \\ \dot{\boldsymbol{x}}(t) &= f(t, \boldsymbol{x}(t), u(t), d(t), \boldsymbol{\theta}(t)) \end{aligned} \tag{3.18}$$

$\boldsymbol{x} \in \mathbb{R}^{n_x}$, and the manipulated variables; $u \in \mathbb{R}^{n_u}$, $d \in \mathbb{R}^{n_d}$, and $\boldsymbol{\theta} \in \mathbb{R}^{n_\theta}$. The system in (3.18) is solved numerically by discretization and thus must be formulated as an IVP in discretized form. In this thesis the usage of zero-order hold piecewise constant discretization is used, where the time interval $[t_0, t_{max}]$ is divided into K intervals with a constant length of T_s. This means that the manipulated variables is constant between time intervals t_n and t_{n+1}, $n = 0, 1, .., K-2, K-1$. Therefore, (3.18) can be rewritten to the expression given in (3.19).

$$\begin{aligned} \boldsymbol{x}(t_0) &= \boldsymbol{x_0} \\ \dot{\boldsymbol{x}}(t) &= f(\boldsymbol{x}(t), u_n, d_n, \boldsymbol{\theta}_n), \qquad t_n \leq t < t_{n+1} \end{aligned} \tag{3.19}$$

3.1.5 Steady State

A property of the Hovorka model is its steady state, that is, an equilibrium where the states variables are unchanged over time. To obtain a steady state for the model the state vector from (3.17) must be constant. This means that the derivative vector must remain zero:

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), u_n, d_n, \boldsymbol{\theta}_n) = \boldsymbol{0} \tag{3.20}$$

In addition the BG level would remain steady around a fixed level. Typically this would vary between individuals denoted r. The BG level is calculated from (3.21):

$$\frac{Q_1(t)}{V_G} = r \leftrightarrow \frac{Q_1(t)}{V_G} - r = 0 \tag{3.21}$$

From here it is assumed that there will be no food intake, $d = 0$, and the parameters will be constant. A system of n_x equations with $n_x + 1$ variables can be established by combining (3.21) and (3.20):

$$\begin{bmatrix} f(t, \boldsymbol{x}(t), u_k, d_k, \boldsymbol{\theta}_k) \\ \frac{Q_1(t)}{V_G} - r \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ 0 \end{bmatrix} \tag{3.22}$$

The system defined in (3.22) consists of $n_x + 1$ equations and $n_x + 1$ variables. This system can be reformulated to a univariate expression of the unknown variable u_k. Solving the univariate expression of (3.22) gives the basal insulin rate u_{ss}. The derivation of (3.22) to a univariate function can be found in Appendix A.

Table 3.2: *The extended parameters for the Hovorka model.*

Parameter	Unit	Distribution	Variability (z, U)
EGP_0	$mmol/min$	$\sim \mathbf{N}(0.0169, 0.0039^2)$	5%, $\mathbf{U}(0, 3$ hours$)$
F_{01}	$mmol/min$	$\sim \mathbf{N}(0.0111, 0.0007^2)$	5%, $\mathbf{U}(0, 3$ hours$)$
k_{12}	min^{-1}	$\sim \mathbf{N}(0.00649, 0.00282^2)$	5%, $\mathbf{U}(0, 3$ hours$)$
$k_{a,1}$	min^{-1}	$\sim \mathbf{N}(0.0055, 0.0056^2)$	Stationary
$k_{a,2}$	min^{-1}	$\sim \mathbf{N}(0.0683, 0.0507^2)$	Stationary
$k_{a,3}$	min^{-1}	$\sim \mathbf{N}(0.0304, 0.0235^2)$	Stationary
$S_{I,1}$	$min^{-1}/mU/L$	$\sim \mathbf{N}(0.00512, 0.00131^2)$	5%, $\mathbf{U}(0, 3$ hours$)$
$S_{I,2}$	$min^{-1}/mU/L$	$\sim \mathbf{N}(0.00082, 0.00032^2)$	5%, $\mathbf{U}(0, 3$ hours$)$
$S_{I,3}$	L/mU	$\sim \mathbf{N}(0.052, 0.0125^2)$	5%, $\mathbf{U}(0, 3$ hours$)$
k_e	min^{-1}	$\sim \mathbf{N}(0.14, 0.035^2)$	5%, $\mathbf{U}(0, 3$ hours$)$
V_G	L	$exp(V_G) \sim \mathbf{N}(1.16, 0.23^2)$	Stationary
V_I	L	$\sim \mathbf{N}(0.12, 0.012^2)$	Stationary
τ_S	min	$\frac{1}{T_S} \sim \mathbf{N}(0.018, 0.0045^2)$	Stationary
τ_D	min	$ln\left(\frac{1}{T_D}\right) \sim \mathbf{N}(-3.689, 0.025^2)$	Stationary
Ag	,	0.8	20%, $\mathbf{U}(0, 24$ hours$)$
BW	kg	$\mathbf{N}(74.9, 14.4)$	Stationary

3.1.6 Virtual Cohort of people with T1D

This section defines a virtual cohort of people with T1D. The sample size of the cohort is 50 ($n = 50$). The virtual cohort consist of virtual patients individually defined by a set of physiological parameters given by $\boldsymbol{\theta}(t)$. In practice the physiological parameters are experimentally collected during studies with T1D patients. This done by fitting the physiological model to the individual patients. The model parameters estimated by Hovorka et al. are used to estimate the population in this thesis [26, 28]. Hereby, the virtual population approximately models an actual population. The physiological parameters are estimated from probability distributions as shown in Table 3.2. In addition to being randomly distributed interpersonally the parameters are added with intra-individual variability such that selected parameters oscillates over time. This biological phenomenon is usually refereed to as the Circadian Rhythm. Here, a parameter value, p_n, oscillates by a sinusoidal oscillation as seen in (3.23).

$$p_n = p_0 + \sin\left(\pi \cdot \frac{t}{60 \cdot U}\right) A \tag{3.23}$$

Here the wave length is defined by the parameter, U, and the amplitude $A = p_0 \cdot z$, where z denotes the percentage size of the amplitude in relation to the initial parameter size, p_0. All physiological parameters and their respective inter and intra variability are given in Table 3.2. Figure 3.5 illustrates how the parameter variations looks for two arbitrary individuals. Figure 3.5 shows how the variation in the insulin parameters S_{I1}, S_{I2}, S_{I3} during a 5 hour period. Note how the thickness of the line symoblises the relative size of the parameter S_{I3}. As shown the parameters varies across individuals but also within individuals.

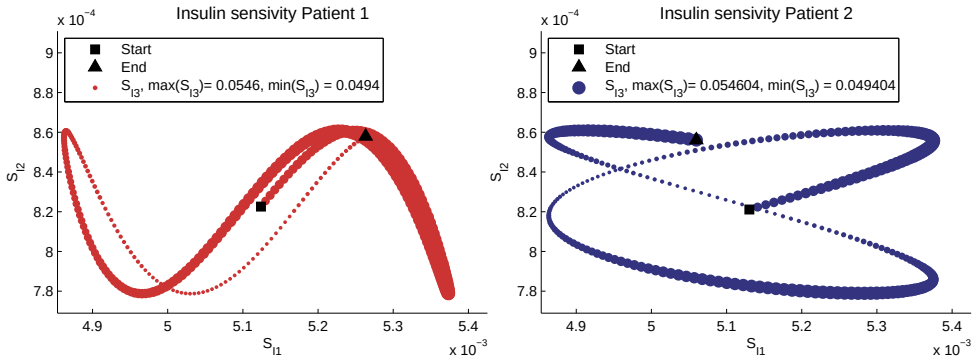


Figure 3.5: *An example of the intra variability which exist between the virtual patients. The insulin sensitivity S_{I1}, S_{I2}, S_{I3} in a 5 hour period for two individuals is shown. Notice how the thickness of the line displays the relative size of S_{I3}.*

3.2 Continous Glucose Monitor Model

To simulate the real life usage of a CGM sensor noise a CGM model proposed by Kovatchev and Breton adds an additional state to the Hovorka model [32]. The CGM tracks the glucose level in the interstitial fluid to asses the value of the BG. This creates a lag modelled by (3.24)

$$\frac{dG_I}{dt}(t) = -\frac{(G_I(t) - G(t))}{\tau_I} \tag{3.24}$$

τ_I [min] represents the time lag between the plasma and interstitial fluid and $G_I(t)$ is the interstitial glucose and $G(t)$ is the BG. From here Kovatchev and Breton proposes a non-Gaussian noise using an autoregressive moving average process.

$$e_n = \begin{cases} v_n & \text{if } n = 1 \\ 0.7 \cdot (e_{n-1} + v_n) & \textbf{otherwise} \end{cases} \tag{3.25}$$

Where $v_n \sim N(0, 1)$. The final definition of the noise becomes:

$$\varepsilon_n = \left(\xi + \lambda \cdot \sinh\left(\frac{e_n - \gamma}{\delta}\right)\right) \tag{3.26}$$

Combining the CGM signal lag modelled by (3.24) and the CGM signal noise modelled by (3.26) the simulated CGM signal is defined by (3.27).

$$G_{CGM}(t) = G_I(t) + \varepsilon_n \tag{3.27}$$

Table 3.3: *CGM model parameters*

Parameter	Value
λ	15.96
ξ	-5.471
δ	1.6898
γ	-0.5444

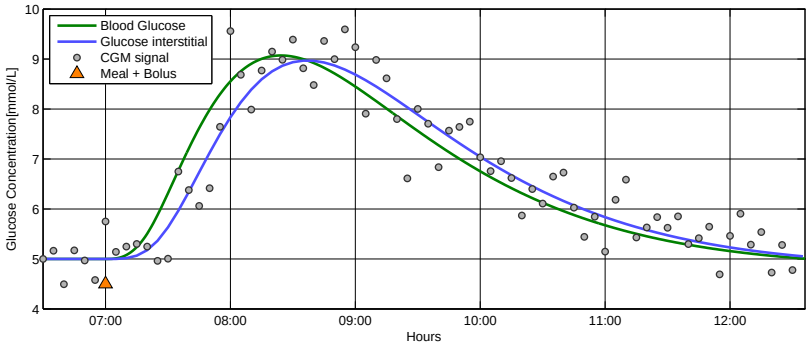


Figure 3.6: *This figure shows the Continous Clucose Monitor Model with noise. It shows the lag to interstitial fluid and the sensor noise for a single* 50 *g meals and* 1 *U meal bolus.*

Kovatchev and Breton also gives a set of parameters displayed in Table 3.3 which they consider realistic to simulate the implementation of a CGM. Figure 3.6 shows the BG and interstitial glucose plotted with the noised CGM signal in a 70 kg virtual patient with one meal and one insulin bolus at 07:00.

3.3 Performance Metrics

The physiological model with the noised CGM signal has been defined. Thereby it is now possible to simulate a virtual patient. This leads to the need for methods to evaluate how the simulated patients handle their T1D. The evaluation of a run and how well the treatment of the T1D patient is conducted will be denoted as the performance of a run. Hence, a performance of a run indicates whether a T1D patient is treated well or bad. The performance metrics has been selected to reflect the main factors which explains much of the complications related to T1D. As described in Section 2.1 minimization of complications is related to hypo- and hyper-glycemia and glycemic variability. Hence, the

Table 3.4: *The performance metrics that will be used to evaluate simulations.*

Abbreviation	Definition
TIZ	Time In Zone
TA_{T_x}	Time Above Threshold T_x
TB_{T_y}	Time Below Threshold T_y
MAGE	Mean Amplitude of Glycemic Excursion
ADRR	Average Daily Risk Range
LBGI	Low Blood Glucose Index
HBGI	High Blood Glucose Index
HbA1c	Glycated hemoglobin

metrics seeks to show either hypoglycemia, hyperglycemia, glycemic variability or a combination of these factors. In Table 3.4 the list of performance metrics are shown.

TIZ is an expression of the time the BG level is between 4-10 mmol/L. TA_{Tx} denotes the time spend above a threshold, Tx, while TB_{Ty} indicates the time spend below a threshold Ty. TIZ, TA_{Tx} and TB_{Ty} are used to identify if hypo- or hyper-glycemia is present in a T1D patient. MAGE indicates an average value of the fluctuations of the BG level. MAGE is calculated as a mean value of the distance between the highest and lowest BG value in 6 hours intervals [14]. Additionally standard statistical measurements are available, such as; Mean Blood Glucose, median and standard deviation.

3.3.0.1 Blood Glucose Risk Space (ADRR, LBGI, and HBGI)

The Average Daily Risk Range (ADRR) is a measure of variability. The ADRR is in practice computed from SMBG data and developed by Kovatchev et al. in [31]. The development of ADRR has been done in a clinical environment and the ADRR showed a superior balance in predicting hypo- and hyper-glycemia for people with T1D. This means that a T1D patient can use ADRR to indicate a risk category of either hyper- or hypo-glycemia [31].

The idea behind the ADRR is to transform the BG readings to depict a risk measure. The transformation of the BG to a risk measure is done through 3.28 which is an asymmetric function that will penalize low BG values more relative to high BG:

$$f(BG) = 1.509 \cdot \left[\ln(BG \cdot 18.0182)^{1.084} - 5.381\right] \tag{3.28}$$

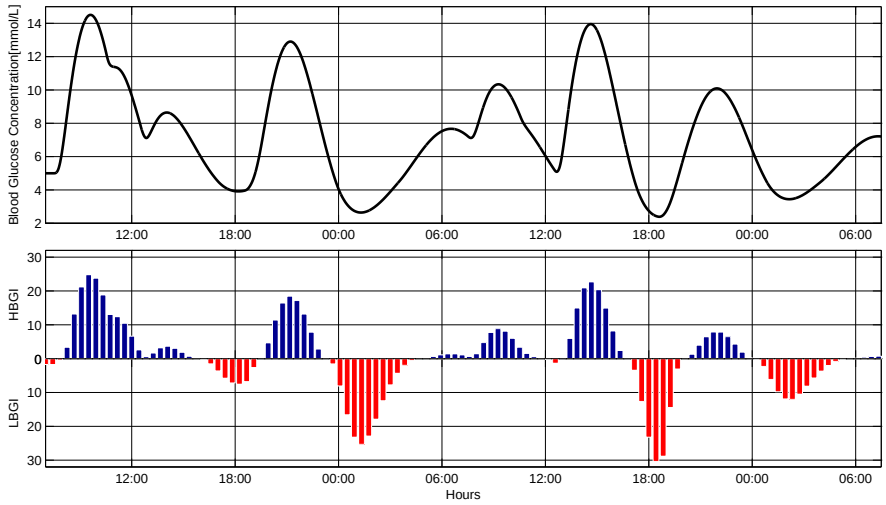


Figure 3.7: *The Risk Space LBGI and HBGI (bottom) as a function of the blood glucose levels (top) over a 2 day period with a total of 7 meals. Note how the asymmetric risk space gives more weight on low BG levels especially around 01:30 during night.*

From here the rl and rh can be calculated:

$$rl(BG) = \begin{cases} 10 \cdot (f(BG))^2 & \text{if } f(BG) < 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.29}$$

$$rh(BG) = \begin{cases} 10 \cdot (f(BG))^2 & \text{if } f(BG) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.30}$$

$$\tag{3.31}$$

rl and rh represent the elements of $f(BG)$ below and above zero respectively. Hereby, rl and rh gives a metric for the low and high BG levels respectively. Low and High Blood Glucose Index ($LBGI$ and $HBGI$) is denoted as the average value of rl and rh respectively. Figure 3.7 graphically show the risk space as a function of the BG level. Note that lower BG level gives a higher penalty compared to higher BG level. This is particular noticeable around night time where the BG level becomes below 4 mmol/L. This is in correspondence with the fact that hypoglycemia is considered more dangerous compared to hyperglycemia. Let x_{jl} be a BG reading from a CGM where $j \in J$ is the reading number on day $l \in L$. Kovatchev et al. notes that a minimum of 3 measurements per day over a minimum of 14 days is required for a valid use of

Table 3.5: *The translation of risk for ADRR.*

Range	Risk of hypo/hyper-Glycemia
$ADRR < 20$	Low
$20 \leq ADRR \leq 40$	Medium
$40 < ADRR$	High

this method. The ADRR can be found by (3.32)

$$ADRR = \frac{1}{M}\sum_{l=1}^{M}(LR_l + HR_l) \tag{3.32}$$

Where M is the cardinality of L, that is $M = |L|$ and

$$LR_l = \max_{\forall j \in J}(rl(x_{jl})) \qquad \forall l \in L \tag{3.33}$$

$$HR_l = \max_{\forall j \in J}(rh(x_{ll})) \qquad \forall l \in L \tag{3.34}$$

The translation of ADRR to a risk-measure is evaluated in Table 3.5

3.3.0.2 HbA1c

HbA1c is a measure used to identify the historical average BG level by measuring the glycated hemoglobin [37]. In assessing glycemic control, HbA1c has been identified by the DCCT to be an important measure is assessing the patients risk of complications. This means that HbA1c is a commonly used metric in the evaluation of T1D patients health. HbA1c is calculated using a theoretical estimation based on the Mean Blood Glucose (MBG) levels. A comprehensive analysis of the relationship between the estimated HbA1c level and the MBG using data from 1.441 T1D subjects during a 5 week period has been proposed by Rohlfing et al. based on a linear regression analysis ($r = 0.82$)[37]. At the time of publication this study was the first to introduce this many data points for the estimation of HbA1c. The estimation became:

$$HbA1c = \frac{MBG + 4.29}{1.98} \tag{3.35}$$

Kilpatrick et al. noted that the results from Rohlfing et al. was inconsistent between patient groups and that the relationship between the MBG and Hb1Ac was depended on the glycemic control of the population being studied [30].

Hence they proposed a new calculations depending on the treatment type:

$$HbA1c = \begin{cases} \frac{MBG+0.10}{1.54} & \text{Conventional therapy} \\ \frac{MBG-0.47}{1.23} & \text{Intensive therapy} \\ \frac{MBG+3.61}{1.87} & \text{Combined} \end{cases} \tag{3.36}$$

All four evaluation methods will be available. However an average of the four methods will be used going forward. A slight drawback with this estimation method is that interpersonal variability will be eliminated as the method is an estimation across many patients. The HbA1c nonon-diabetics is typically in the range of 5.7%. Diabetics typically has a HbA1ac in the range between 6.5% - 7%. For people suffering from T1D a HbA1c value below 7% is considered within reasonable glycemic control.

3.4 Summary

In this chapter the Hovorka model from Boiroux and Hovorka and Wilinska for people with T1D has been presented. The model described the glucose-insulin dynamics in people with T1D and consisted of three subsystems. The carbohydrate absorption, the insulin absorption from a CSII pump, and the glucose-insulin regulatory system. Additionally, a CGM model was added. The Hovorka model can be formulated as an IVP:

$$\boldsymbol{x}(t_0) = \boldsymbol{x}_0 \tag{3.37}$$

$$\dot{\boldsymbol{x}}(t) = f(t, \boldsymbol{x}(t), d(t), \boldsymbol{\theta}(t)) \tag{3.38}$$

$$f(t, \boldsymbol{x}(t), d(t), \boldsymbol{\theta}(t)) : (\mathbb{R}, \mathbb{R}^{n_x}, \mathbb{R}, \mathbb{R}^{n_\theta}) \mapsto \mathbb{R}^{n_x} \tag{3.39}$$

Several performance metrics are presented to evaluate simulations. This included standard statistical measurements and other risk parameters such as ADRR and HbA1c.

Chapter 4

Methodology

The objective of this chapter is to deal with three numerical challenges. First, to solve the non-stiff system of differential equations presented in the previous chapter. Second, to solve the steady state problem using an iterative root finding method. Third, to present the PID controller used in this software. This chapter is organized as follows. Section 4.1 explains the numerical integration algorithm. Section 4.2 describes an iterative root finding method. Section 4.3 describes the PID controller. Finally, Section 4.4 looks into the implementation of these methods in the programming language Java using the Apache Commons Mathematics Library.

4.1 Explicit Runge-Kutta Methods

The Dormand Prince 54 (DoPri54) method, belongs to the family of Runge-Kutta (RK) methods, and is chosen as the default ODE solver in this thesis. It were developed by Dormand and Prince in 1980 [21] and are currently part of the standard ODE solver library in e.g. Matlab®.

The general idea behind any numerical integration method is to compute an approximate solution x_n of the true value $x(t_n)$ for some point t_n, $n \in N$ given

an initial condition, thus, denoted an IVP in the form:

$$\begin{aligned} \boldsymbol{x}(t_0) &= \boldsymbol{x}_0 \\ \dot{\boldsymbol{x}}(t) &= f(t, \boldsymbol{x}(\boldsymbol{t})) \\ f(t, \boldsymbol{x}(t)) &: (\mathbb{R}, \mathbb{R}^{n_x}) \mapsto \mathbb{R}^{n_x} \end{aligned}$$

Where $\boldsymbol{x}_0$ is the initial vector. The basics of the Explicit RK methods, and for the DoPri54 for that fact, is given by the next step calculation in (4.1) to (4.5).

$$T_i = t_n + c_i h_n \qquad i = 2, .., s \tag{4.1}$$
$$T_1 = t_n$$

$$\boldsymbol{X}_i = \boldsymbol{x}_n + h_n \cdot \sum_{j=1}^{i-1} a_{ij} \cdot f(T_j, \boldsymbol{X}_j) \qquad i = 2, .., s \tag{4.2}$$
$$\boldsymbol{X}_1 = \boldsymbol{x}_n$$

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + h_n \cdot \sum_{j=1}^{s} b_j \cdot f(T_j, \boldsymbol{X}_j) \tag{4.3}$$

$$\hat{\boldsymbol{x}}_{n+1} = \boldsymbol{x}_n + h_n \cdot \sum_{j=1}^{s} \hat{b}_j \cdot f(T_j, \boldsymbol{X}_j) \tag{4.4}$$

$$\boldsymbol{e}_{n+1} = \boldsymbol{x}_{n+1} - \hat{\boldsymbol{x}}_{n+1} = h_n \cdot \sum_{j=1}^{s} d_j f(T_j, \boldsymbol{X}_j), \qquad d_j = b_j - \hat{b_j} \tag{4.5}$$

The step size is denoted as: $t_{n+1} - t_n = h_n$. The coefficients, a_{ij}, b_j, $\hat{b}_j$, c_i, and d_j are given by the Buthcer's Tableau. The explicit methods that will be used here have a Butcher's tableau that is a strictly lower triangle matrix as shown in Table 4.1. The classical RK method uses only (4.1) to (4.3) to estimate the solution of the IVP since the classical RK method does not estimate the truncation error which is given by (4.4) to (4.5). The classical RK method is explicit and has $s = 4$ stages. The method has a fixed step size such that $h_n = h$. The coefficients for the classical RK method is given by the coefficients in Table

Table 4.1: *Butcher's tableau for the Explicit RK method.*

0	0			
c_2	$a_{2,1}$			
$\vdots$	$\vdots$	$\ddots$		
c_s	$a_{s,1}$	$\dots$	$a_{s,s-1}$	0
x	b_1	$\dots$	$\dots$	b_s
$\hat{x}$	$\hat{b}_1$	$\dots$	$\dots$	$\hat{b}_s$
$\hat{e}$	$\hat{d}_1$	$\dots$	$\dots$	$\hat{d}_s$

Table 4.2: *Butcher's tableau for the Classic Runge Kutta method*

0	0	0	0	0
$\frac{1}{2}$	$\frac{1}{2}$	0	0	0
$\frac{1}{2}$	0	$\frac{1}{2}$	0	0
1	0	0	1	0
x	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

Table 4.3: *Butcher's tableau for DoPri 54 method*

0	0	0	0	0	0	0	0
$\frac{1}{5}$	$\frac{1}{5}$	0	0	0	0	0	0
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$	0	0	0	0	0
$\frac{4}{5}$	$\frac{44}{45}$	$\frac{-56}{15}$	$\frac{32}{9}$	0	0	0	0
$\frac{8}{9}$	$\frac{19372}{6561}$	$\frac{-25360}{2187}$	$\frac{64448}{6561}$	$\frac{-212}{729}$	0	0	0
1	$\frac{9017}{3168}$	$\frac{-355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$\frac{-5103}{18656}$	0	0
1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$\frac{-2187}{6784}$	$\frac{11}{84}$	0
x	$\frac{5179}{57600}$	0	$\frac{7571}{16695}$	$\frac{393}{640}$	$\frac{-92097}{339200}$	$\frac{187}{2100}$	$\frac{1}{40}$
$\tilde{x}$	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$\frac{-2187}{6784}$	$\frac{11}{84}$	0

4.2. The DoPri54 method differs from the classical RK method in several ways. The DoPri54 method uses the Butcher's Tableau given by Figure 4.3 and uses an adaptive step size [21]. The DoPri54 method advances the solution with a fifth order accuracy and creates the error estimate with a fourth order accuracy as denoted in (4.5). The method is explicit and has $s = 7$ stages. An example of an IVP is given in (4.6).

$$\begin{aligned} x(0) &= 0, \\ \frac{d}{dt}x(t) &= (x(t)-1)^2 \cdot (t-1)^2 \end{aligned} \tag{4.6}$$

with the analytical solution as found in Maple®:

$$x(t) = \frac{t \cdot \left(t^2 - 3t - 3\right)}{t^3 - 3t^2 + 3t + 3} \tag{4.7}$$

Figure 4.1 identifies the global truncation error for the classical RK and DoPri54 method respectively for the problem in (4.6) in comparisment with the analytical solution from (4.7). Note from the figure that the slope represent the fourth and

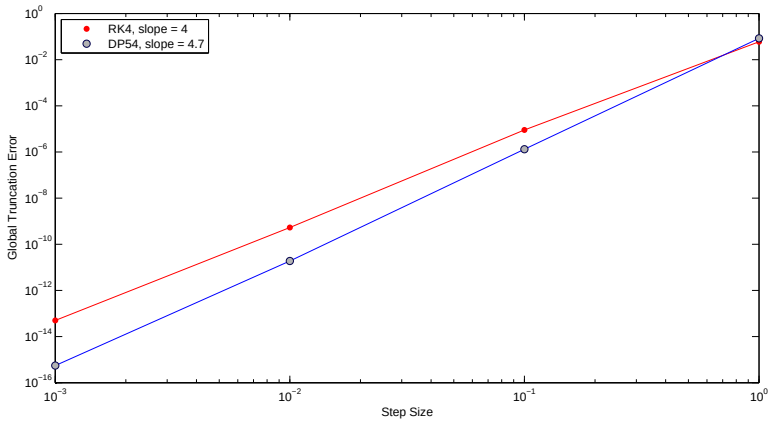


Figure 4.1: *This figure shows convergence for the classical RK and DoPri54 method respectively. Note how the global truncation error is fourth and fifth order accurate.*

fifth order accuracy of the two methods. The DoPri54 uses an adaptive step size potentially changing the step size h_n at each iteration. The adaptive step size has two advantages. First, it helps increase the solution accuracy since it will take shorter and more accurate steps when needed. Second, it reduces the computation time as it allows for larger step size when possible. As the analytical solution is unknown the truncation error is estimated as the difference between the fifth and fourth order estimate as shown in (4.5). The step size is chosen such that the norm of the estimated error, e_{n+1} is smaller than some threshold tolerance. In (4.8) the tolerance term is defined. Note that i indicates the i'th element of the state vector.

$$tol_i = absTol + resTol \cdot \max((x_n)_i, (x_{n+1})_i) \tag{4.8}$$

Then the error ratio, r is defined by (4.9).

$$r_{n+1} = \sqrt{\frac{1}{n_x} \cdot \sum_{i=1}^{n_x} \frac{(e_{n+1})_i}{tol_i}} \tag{4.9}$$

Where $(e_{n+1})_i$ denotes the value of the error from (4.5) and n_x is the dimension of the state variable $\boldsymbol{x}(t)$. tol_i is a combination of the absolute and the relative tolerance term. If the previous solution, $(x_n)_i$ or the current solution, $(x_{n+1})_i$ is large, the relative tolerance will be the dominating part and the absolute tolerance otherwise. Hence, the error estimation $(e_{n+1})_i$ will be seen relative to the size of the solutions generated. (4.9) can be seen as a ratio between the

Algorithm 1 Dormand Prince 54 - Step size control

1: Input: $\boldsymbol{x}_n$, $\boldsymbol{e}_{n+1}$, h_n, $maxStep$, $minStep$, $relTol$, $absTol$
2: Evaluate:

$$r_{n+1} \leftarrow \sqrt{\frac{1}{n_x} \cdot \sum_{i=1}^{n_x} \frac{(e_{n+1})_i}{tol_i}}$$

3: **if** $r_{n+1} > 1$ **then**
4: Reject step, reduce step size and try again.
5: Evaluate:

$$\gamma \leftarrow \left(\frac{\epsilon}{r_{n+1}}\right)^{1/p}$$
$$factor \leftarrow \min\left(maxStep, \max\left(minStep, \gamma\right)\right)$$
$$h_n \leftarrow factor \cdot h_n$$

6: **else**
7: Accept step
8: Update solution: $\boldsymbol{x}_{n+1}$, and time: $t_{n+1} = t_n + h_n$
9: **end if**

actual error and the tolerance. The error ratio, r_{n+1}, defines whether a step should be accepted or rejected. If $r_{n+1} > 1$ the step will be rejected. This means that the solution will be computed again with a new reduced step size. On the other hand, if $r_{n+1} \leq 1$ the step size will be accepted and the algorithm will continue the next iteration.

In the case where the estimated error ratio r_{n+1} is larger than 1, the value of h_n will be reduced by a factor γ as denoted in (4.10).

$$\gamma = \left(\frac{\epsilon}{r_{n+1}}\right)^{1/p} \tag{4.10}$$

Here $\epsilon = 0.9$ is a safety factor and p is the order of the method as defined in [33]. The step size control can be see in Algorithm 1.

The DoPri54 method is the standard ODE solver in many mathematical software's as e.g. Matlab®, GNU Octave and Simulink®, though with a slightly different step size calculation. As seen in Figure 4.1 the DoPri54 method has a global truncation error of $O(h^5)$ which means that reducing the step size with h reduces the error with h^5. This is superior to the classical RK method for comparison. One of the drawbacks with the DoPri54 method is that the number of steps can become unpredicted and therefore the computational time can become unpredicted as well. For this reason it is important to state a reasonable absolute and relative tolerance such that the method will execute with a reasonable running time. The full algorithm of the DoPri54 method as implemented in the numerical library used in this thesis can be seen in Algorithm

Listing 4.1: *Definition of ODE in Java*

```
1  private static class ExampleODE implements ...
       FirstOrderDifferentialEquations {
2
3      private double c;
4
5      // Constructor of ODE
6      public ExampleODE(double c) {
7          this.c      = c;
8      }
9
10     // Dimension of state variable
11     public int getDimension() {
12         return 1;
13     }
14
15     // Here goes ODE such that yDot = f(t,y)
16     public void computeDerivatives(double t, double[] y, ...
           double[] yDot) {
17         yDot[0] = Math.pow( (y[0]-c),2.0) * Math.pow((t-c),2.0);
18
19     }
20
21 }
```

Listing 4.2: *Instantiate ODE function.*

```
1  ExampleODE ode = new ExampleODE(1);
```

2. The implementation of the Hovorka model in Java is extensive. Instead the example from (4.6) is used to show the implementation of an ODE and the subsequent usage of the DoPri54 solver in Java. Please refer to Appendix F for the Java implementation of the Hovorka model. First, we define the ODE using the `FirstOrderDifferentialEquations` interface as shown in Listing 4.1.

Then, the function must be instantiated as shown in Listing 4.2. From this point the algorithm can be executed by the command in Listing 4.3. Notice that the solver will simply update the state variable x.

Listing 4.3: *Solving the ODE.*

```
// ...
// First instantiate the integrator:
FirstOrderIntegrator dp = new
 DormandPrince54Integrator([minstep],
 [maxstep],[abstol],[reltol]));

// Solve from tzero to tmax. x will be updated.
dp.integrate(ode, tzero, x, tmax, x);
// ...
```

Algorithm 2 Dormand Prince 54 - Explicit Runge-Kutta Method

1: Input: $\boldsymbol{x}_n$, t_n, t_{max} (final time)
2: **while** $t_n < t_{max}$ **do**
3: Evaluate:

$$T_1 \leftarrow t_n$$
$$X_1 \leftarrow x_n$$

4: **for** $i = 2 \rightarrow s$ **do**
5: Evaluate:

$$T_i \leftarrow t_n + c_i h_n$$
$$\boldsymbol{X}_i \leftarrow \boldsymbol{x}_n + h_n \cdot \sum_{j=1}^{i-1} a_{ij} \cdot f(T_j, \boldsymbol{X}_j)$$

6: **end for**
7: Evaluate:

$$\boldsymbol{x}_{n+1} \leftarrow \boldsymbol{x}_n + h_n \cdot \sum_{j=1}^{s} b_j \cdot f(T_j, \boldsymbol{X}_j)$$
$$\hat{\boldsymbol{x}}_{n+1} \leftarrow \boldsymbol{x}_n + h_n \cdot \sum_{j=1}^{s} \hat{b}_j \cdot f(T_j, \boldsymbol{X}_j)$$
$$\boldsymbol{e}_{n+1} \leftarrow h_n \cdot \sum_{j=1}^{s} d_j f(T_j, \boldsymbol{X}_j)$$

8: Evaluate step size control, Algorithm 1
9: **end while**
10: **return** $\boldsymbol{x}_{t_{max}}$

4.2 Newton-Raphson's Method

This section deals with the implementation of the Newton-Rhapson's method to find the steady state of the Hovorka model in Java. The Newton-Rhapson's

method is an iterative root finding method for non-linear equations. In this thesis the Newton-Rhaphson's method is used to compute the steady state for the system of nonlinear equations defined in (3.22) from Section 3.1.5:

$$\begin{bmatrix} f(\boldsymbol{x}(t), u_k, d_k, \boldsymbol{\theta}_k) \\ \frac{Q_1(t)}{V_G} - r \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix} \tag{4.11}$$

This system has $n_x + 1$ equations with $n_x + 1$ variables. This system could be solved using the multivariate Newton-Raphson's method, where the general definition is:

$$f(\boldsymbol{x}) = \mathbf{0} \tag{4.12}$$

$$f : \mathbb{R}^{n_x} \mapsto \mathbb{R}^{n_x} \tag{4.13}$$

The multivariate iterative root finding method is based on following approximation:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - J^{-1}(\boldsymbol{x}) f(\boldsymbol{x}) \tag{4.14}$$

However, since the univariate case of the Newton-Rhaphons method is more simple to implement in Java, (4.11) must be revised to a univariate expression. The full transformation can be seen in Appendix A. The univariate function, $q(u)$, derived from (4.11) becomes:

$$\begin{aligned} q(u) = &- F_{01}^{c} - F_R - \frac{r \cdot V_G \cdot k_{b1} \cdot u_k}{V_I \cdot k_e \cdot k_{a1}} \\ &+ k_{12} \cdot \frac{r \cdot V_G \cdot k_{b1} \cdot u_k}{k_{a1} V_I k_e \left(k_{12} + k_{b2} \frac{u}{k_{a2} V_I k_e} \right)} \\ &+ EGP_0 \cdot \left(1 - \frac{k_{b3} \cdot u}{V_I \cdot k_e \cdot k_{a3}} \right) \end{aligned} \tag{4.15}$$

Where the derivative of $q(u)$ becomes.

$$\begin{aligned} \frac{dq(u)}{du} = &- \frac{r \cdot V_G \cdot k_{b1}}{V_I \cdot k_e \cdot k_{a1}} \\ &+ \frac{k_{12} \cdot r \cdot V_G \cdot k_{b1}}{k_{a1} V_I k_e \left(k_{12} + k_{b2} \frac{u_k}{k_{a2} V_I k_e} \right)} \\ &+ \left(\frac{-k_{b2} \cdot k_{12} \cdot r \cdot V_G \cdot k_{b1} \cdot u}{k_{a2} k_{a1} \left(V_I k_e k_{12}^{-1} + k_{b2} k_{a2}^{-1} u \right)^2} \right) \\ &- \frac{EGP_0 \cdot k_{b3}}{V_I \cdot k_e \cdot k_{a3}} \end{aligned} \tag{4.16}$$

Algorithm 3 Newton-Rhaphson's method to find the approximate steady state bolus, u_{ss}

1: READ $\epsilon \leftarrow$ tolerance, $k_{max} \leftarrow$ max evaluations, $u_0 \leftarrow$ Initial guess .
2: Evaluate:

$$q(u_0) \leftarrow \text{Equation (4.15)}$$
$$q'(u_0) \leftarrow \text{Equation (4.16)}$$
$$k \leftarrow 0, h \leftarrow \frac{q}{q'}, u \leftarrow u_0$$

3: **while** $k \leq k_{max}$ or $|h| \geq \epsilon$ **do**
4: Evaluate:

$$u \leftarrow u - h$$
$$q(u) \leftarrow \text{Equation (4.15)}$$
$$q'(u) \leftarrow \text{Equation (4.16)}$$
$$h \leftarrow \frac{q}{q'}$$
$$k \leftarrow k + 1$$

5: **end while**
6: **return** u

The problem which we seek to implement and solve in Java can then be defined as a univariate case:

$$q(u) = 0 \tag{4.17}$$
$$q(u) : \mathbb{R} \mapsto \mathbb{R}$$

Essentially this will return the basal insulin rate, u_{ss}. Solving (4.17) by Newton-Rhapson's method can be done from definition 4.3.1 in Lars Eldén and Nielsen the general formula for the approximation of the root is given by the iterative Newton method defined as follows [33]:

$$u_{k+1} = u_k - \frac{q(u_k)}{q'(u_k)} \tag{4.18}$$

u_k is the current approximation and we seek to find a step h such that $q(u_k+h) = 0$. Since h can be expected to be small the Taylor approximation gives a good estimate of the function value $q(u_k+h) \approx q(u_k)+q'(u_k)h$. Setting the left hand side to zero allow us to find that $h = \frac{q(u_k)}{q'(u_k)}$ and hence u_{k+1} will become the new approximate solution to the root. The algorithm for the method is described in Algorithm 3. One challenge with the Newton-Rhapson's method, is its usage of the first order derivative. However, in this thesis the derivative was possible to find making it easy to implement. The implementation in Java are done in two steps. First, is the definition of the function and its derivative as shown in Listing 4.4 Second step is the actual execution of the solver initially by defining the function, followed by the solver as showed in Listing 4.5.

Listing 4.4: *Implementing basal function in Java.*

```
// Class which implements the UnivariateDifferentiableFunction ... interface
public class HovorkaBasalState implements ... UnivariateDifferentiableFunction {


    public double value(double u) {
        // here goes q(u)
    }

    public DerivativeStructure value(DerivativeStructure t) {
        // here goes q'(u)
    }

}
```

Listing 4.5: *Execution of Newton-Rhapson's method in Java.*

```
// ...
// Instantiate function to solve
HovorkaBasalState f = new HovorkaBasalState([ode ... parameters],[target BG]);

// Instantiate solver
NewtonRaphsonSolver newton = new NewtonRaphsonSolver();

// Solve
double uss = newton.solve(maxeval,f,min,max,x0);
// ...
```

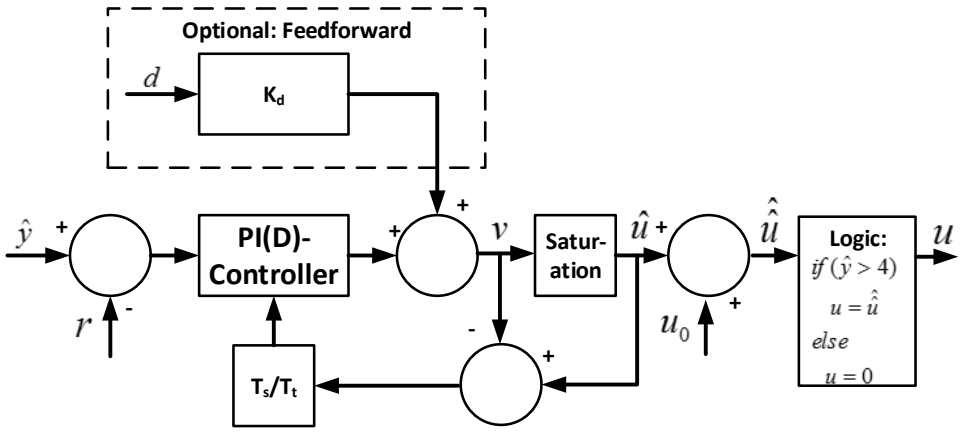


Figure 4.2: *This figure shows the insulin administration strategy used the Proof-Of-Concept AP. It consist of a PID controller, a saturation and some logic. An optional Feed Forward mechanism can be used.*

4.3 Insulin Administration Strategy

As part of the aim of this thesis, as described in Section 1.1, is the implementation of a PID controller which are part of the Proof-Of-Concept AP implemented.

The control algorithm used is based on the PID controller described by Wittenmark et al. in [43]. Some improvements to this PID controller has been made and this section seeks to give a brief overview of the controller implemented and the improvements made shown by Figure 4.2.

- **Input:** As described in Section 2.2.2 Figure 2.6 the control system receives the CGM signal, $\hat{y}$ from the simulated T1D patient and potentially a meal input d. The input, $\hat{y}$, is subtracted the set point value r to calculate the error. The error term is given as input to the PID controller.

- **PID-controller:** The PID controller consist of three elements. The proportional term, P, the Integral term, I, and the derivative term, D. In this example the derivative term, D, has been disabled, in practice making this a PI-controller. The goal with the PI controller is to calculate an output value based on the difference between r and $\hat{y}$. From a computational

standpoint the PI-controller looks as shown in (4.19) to (4.22).

$$P = K \cdot (r - \hat{y}) \tag{4.19}$$

$$I = I_{old} + \frac{K \cdot h}{T_i}(r - \hat{y}) + \frac{h}{T_t} \cdot (\hat{u} - v) \tag{4.20}$$

$$FF = K_d \cdot CR \cdot d \tag{4.21}$$

$$v = P + I + FF \tag{4.22}$$

Where, K, is the gain, h, is the sampling period, T_i, is the integral time, T_t, is the reset time, CR, is the insulin-to-carbon ratio, and K_d, is the Feed Forward gain. The CR is an estimation of the insulin dosing to eliminate a 50 g meal and indicates the ratio between the meal intake and the size of the bolus.

- **Feed Forward:** The Feed Forward mechanism is optional and simply adds an input based on the meal input as shown in (4.21) and (4.22). The meal bolus will be estimated by calculating CR for each virtual patient. Calculation of the CR has been done by developing a heuristic for estimation of the CR which can be seen in Algorithm 4. The algorithm estimates the CR given a 50 g meal. After 180 minutes the BG levels are evaluated. If the BG levels are within acceptable range the CR will be estimated, otherwise the algorithm will continue until tolerance is satisfied.

- **Saturation:** The saturation determines a limit for the received signal i.e. a limit on the actual output value. In this case the limit is given by the physical dosing limit by the pump such that 0 mU $\leq \hat{u} \leq$ 300 mU.

- **Anti wind-up:** The anti wind-up mechanism reduces the oscillation of the controller. Here the signal v and $\hat{u}$ is used to determine the anti wind-up.

- **Logic:** The logic is a method to shut off the insulin infusion pump if the CGM value, $\hat{y}$, is lower than 4 mmol/L.

- **Output:** The output value, u, indicates the insulin dosage calculated by the insulin administration system measured in mU/min.

The controller has been tuned by visually inspecting the ADRR performance of an average individual adjusting the parameters K, T_i, T_t and K_d.

Algorithm 4 Algorithm for estimation of optimal Insulin-to-Carbon Ratio

Input: u_{basal}, BG_{basal}
$d \leftarrow 50g$, $u \leftarrow 1U$, $t_0 \leftarrow 0$, $t_{max} \leftarrow 180$
$BG \leftarrow$ array for storage
$A_1, A_2, A_3 \leftarrow$ **false**
while $A_1 \& A_2 \& A_3$ **not true do**
    $BG \leftarrow$ Simulate patient with d, u from $t_0 \rightarrow t_{max}$
    if $0.85 \cdot BG_{basal} \leq BG_{end} \leq 1.1 \cdot BG_{basal}$ **then**
        $A_1 \leftarrow$ **true**
    end if
    if $\min BG < 5$ **then**
        $u \leftarrow u - 0.01$
    else
        $A_2 \leftarrow$ **true**
    end if
    if $\max BG > BG_0 + 4$ **then**
        $u \leftarrow u + 0.01$
    else
        $A_3 \leftarrow$ **true**
    end if
end while
$CR \leftarrow \frac{d}{u}$
return CR

4.4 Apache Commons Mathematics Library

The described numerical methods for solving the IVP and the univariate root finding problem from Section 4.1 and Section 4.2 have been implemented in Java using the open source Apache Commons Mathematics library (ACM) [4]. The library is a formerly Fortran based library that has been translated, expanded and developed especially for Java purposes. The library is well supported and, of the authors understanding, the best known practice within the field of numerical methods, solvers, optimization tools and more for Java. The motivation behind choosing the ACM Library is threefold:

- **Documentation:** The first and foremost reason for choosing the ACM lies in the extensive documentation available. This ensures easy education, reduction of potential bugs and ease in implementation. In addition Apache Commons is part of the Apache license which in practice makes it free of charge [5]. One of the benefits of the Apache License is that it allows work or derivative work of Apache license products to be redistribute under the condition that the Apache license is quoted in the license itself.
- **Implementation:** The methods in the ACM library is relatively easy to implement for an experienced Java programmer. This is especially due to the JavaDoc documentation available at their homepage.

- **Future development:** The ACM library is updated at a regular basis and the source code can be assessed directly which means that contributions to the ACM library can be made and submitted online. The submissions will be verified by the ACM team to ensure consistency in every update.

Based on above motivational points it appears that the ACM library is a well documented, updated and free library. Therefore, the ACM library has been considered very suitable for this thesis. The entire source for the ACM library can be found in Appendix G.

4.5 Summary

In this chapter the numerical method for solving the IVP is presented. The IVP will be solved with the DoPri54 method with adaptive step size. The steady state for the IVP is found using the Newton-Rhapsons iterative root finding method. Additionally, a PID controller with Feed Forward was introduced as part of the proof-of-concept AP. The numerical methods is implemented using the open-source ACM library to ensure proper implementation, documentation and future reliability. The next chapter will initiate the development of the software for in silico testing for an AP.

Chapter 5

Developing a Software for In Silico Testing for an Artificial Pancreas

The objective of this chapter is to present the development of the software for in silico testing for an AP. The conceptual clarification from Section 1.4 is a prerequisite for reading this chapter. The chapter is organised as follows. Section 5.1 analyse the requirements of the software. Section 5.2 implements the requirements in Java. Section 5.3 implements multi-threading and finally Section 5.4 introduces a configuration-system.

5.1 Requirements and Functionalities

The overall purpose of this software is to mimic the clinical setup of a closed-loop study such that the software imitates the actual flow of data between the virtual patient, the CGM device, the insulin administration strategy and the insulin pump. The software framework developed consist of following 6 principal components:

1. The virtual cohort as described in Subsection 3.1.6.

2. The mathematical model described in Section 3.1 to simulate the glucose-insulin dynamics of patients suffering from T1D.
3. The Insulin infusion pump defined by the subcutaneous insulin infusion described in Subsection 3.1.2.
4. The CGM model as defined in Section 3.2.
5. The Insulin Administration Strategy as described in Section 4.3.
6. Performance Metrics to evaluate simulation of the system as described in Section 3.3.

As the Graphical User Interface (GUI) is out of scope of this thesis Matlab® will be used as a substitute. This means that Java will be called from Matlab. Additionally, all plotting is done in Matlab.

Throughout the remaining part of this chapter the description will be divded into three key areas. The input - containing principal component 1. The system - containing Principal compoent 2-5 and finally the output - containing principal component 6.

5.1.1 Input - Principal component 1

The first principal component is the virtual cohort which consist of the following elements:

- **Virtual Patient:** As a central part of the AP system is the ability to define a patient. A patient is a definition of two key elements:
  - **Physiological Parameters:** A set of individual physiological parameters that can vary over time as the physiological composition of the patient change. This also includes the bodyweight of an individual.
  - **Meal Profiles:** One or more set of meals with an associated start time of meal, an amount of carbohydrate in the meal $[g]$, and a duration of the meal $[min]$.

The virtual patient can be summarised by a domain model. Figure 5.1 shows the domain model of the software and the data structures used. Note that for one patient, many variations in meal profiles and continuous changes in the

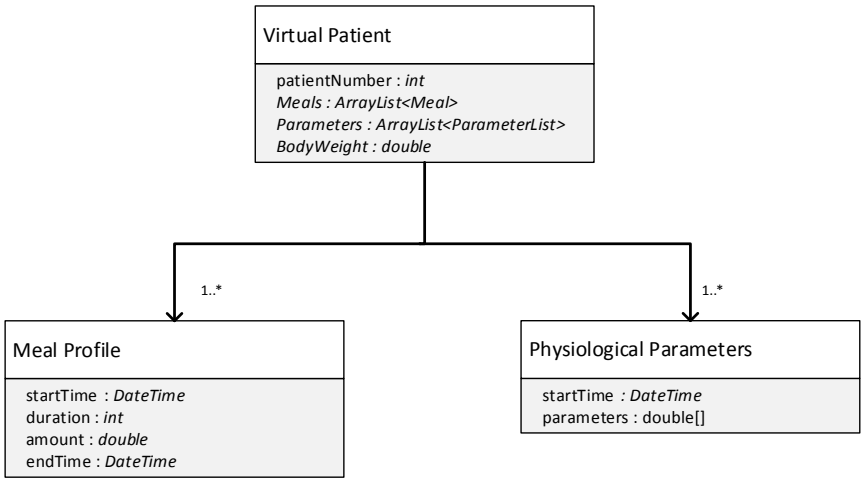


Figure 5.1: *This figure show the domain model for the software and the datastructures used. Note that a patient can consist of many meals and many parameters as denoted by* $1..^{\star}$.

Table 5.1: *Meal profile input*

Time	Duration [min]	CHO [g]
07:00	**20**	**60**
10:00	10	10
12:00	30	50
18:00	25	55

Table 5.2: *Parameter input*

Time	EGP0	F01	$\cdots$	Ag
07:00	0.0161	0.0097	$\cdots$	0.8
07:15	0.0161	0.0097	$\cdots$	0.78
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$

physiological parameters exist. This is natural in the sense that meal profiles across individuals are very different. Additionally, the physiological parameters changes over time due to the variation as a consequence of lifestyle changes or biological events. The user must be able to make the decision on how the virtual T1D patient is composed such that modelling of the cohort can be done easily. An intuitive way to retrieve this input is by allowing the user to create files in Comma Separated Values (CSV) format as depicted in Table 5.1 for meal profiles and as in Table 5.2 for the parameter definitions. Table 5.1 is the layout for the meal profiles for a single day containing information on when a meal is consumed, the length and the size of the meal. Figure 5.2 shows the parameter values for specific times during a day. This would be possible to load into the domain model and the respective data structures. The data structure `DateTime` is for calender calculations. The drawback with the domain model proposed is that it need to be transformed to be usable in the numerical methods. Thus, after loading the data into the domain model the data is transformed. The transformation simply ignores the calender time and instead uses the zero-order

Table 5.3: *Transformed CSV file. One file indicates one virtual patient suffering from T1D.*

Start time	t	CHO [g]	EGP0	F01	···	Ag
12-02-2014 **07:00**	0	**15**	0.0161	0.0097	···	0.8
	5	**15**	0.0161	0.0097	···	0.8
	10	**15**	0.0161	0.0097	···	0.8
	15	**15**	0.0161	0.0097	···	0.78
	20	0	0.0161	0.0097	···	0.78
	⋮	⋮	⋮	⋮	⋱	⋮

hold piece wise constant principle as depicted in Table 5.3. Notice from Table 5.3 how the breakfast meal from Table 5.1 is divided into equal amounts over the duration of the meal. In practice it is now possible to create as many virtual patients as possible to form a cohort. However, it is rather comprehensive to do this for, lets say, 500 virtual patients, that is, defining CSV files for each individual. Thus, some additions to the previously described requirements must be considered:

- **Reuse meal profiles:** It should be possible to simply create one day of meals and reuse this meal profile for several days across individuals. In addition it should be possible for several individuals to use the same meal profiles, hereby the user needs only to make the meal profiles once.

- **An easy way of changing parameters:** Creating individual physiological parameters for 500 virtual patients with the CSV method above is very time consuming. Instead the program should be able to handle a vector input with the parameter values as a function of time.

- **Study protocol:** The user should be able to initiate trials composing of several patients. A trial is specified by having a start time, an end time and a time between measurements (step size), and finally the number of patients that participate in the trial.

A transformed file in the form presented in Table 5.3 corresponds to a single virtual individual from the trials starts to the trial ends. The transformed file thus contains data on meal input and the physiological parameters for each time step, T_s.

5.1.2 System - Principal Component 2-5

Input from each virtual patient for each time step T_s is the input to the system. The system simulates the T1D patient by simulation of the glucose-insulin dynamics considering the input files. The system should take following elements into consideration:

- **Physiological model:** Simulate a patients glucose-insulin dynamics by implementing the simulation models described in Section 3.1.
- **CGM model:** Consider the irregularities of a CGM such as lag and noise by implementing the CGM model proposed in Section 3.2.
- **Insulin Administration:** Be able to make decision on insulin dosage by implementing the insulin administration strategy proposed in Section 4.3.

At this point the simulation should be conducted and the simulated BG stored. However, the user requires to also have:

- **Software configurability:** The software should be easy to configure. This could for instance be the tolerance of the numerical integrator or the directory for which the files will be exported to.
- **High performance:** To increase the usability of the software the software must execute within a few minutes.

5.1.3 Output - Principal Component 6

As a the system has been conducted the software should analyse and evaluate the system performance:

- **Outcome Evaluation:** As the trial is conducted it must be evaluated on individual patient basis. A variety of statistical measures and risk metrics as proposed in Section 3.3 should be implemented.

The proposed requirements creates a platform in which closed-loop control systems can be tested. The next section will consider the implementation of the described requirements and functionalities in Java.

5.2 Implementation

Unified Modelling Language (UML) Sequence Diagrams will be used to illustrate the series of steps which the program will execute. Steps in this context can be functionalities, work flow or messages. Appendix B gives a brief overview of the notation of the UML Sequence Diagrams. This Section has been divided into three areas corresponding to the three different functionalities of the program. The three areas can be summarized as follows:

1. **Input:** As described in Section 5.1 the user should be able to enter one or more input files for one or more virtual patients. In addition the user defines a trial that gathers all individual patients under one test. This step should transform Table 5.1 and Table 5.2 to the output depicted in figure 5.3.
2. **System:** This functionality models the system behaviour which is the dynamics of one or more T1D patients based on the input received. The system should handle the inputs and model the physiological behaviour for each T1D patient. As the system will receive various input the system will simulate one or more patients and handle the system entities. The system simulation will be stored in a csv file or in memory depending on the user needs.
3. **Output:** The final functionality gathers the output from the system simulation into relevant performance metrics for each patient in the trial.

Each of the above mentioned functionalities will be described in detail in the next three subsections. This will be done in coherence by the current best practice in the UML notation as described by Bell in [15]. **Please note** for easy reading each subsection is numbered in correspondence with the figures.

5.2.1 Input

Figure 5.2 shows the execution process of this part of the software. The parameters for this call should be:

- The number of virtual patients
- The sampling time (step size)
- The body weight for each patient

- The number of days that the trial should last
- A start and end time of the trial.

This functionality also propose an easy way to handle continuous input such as the physiological parameters which varies over time. This is done by including an optional vector with physiological parameters.

5.2.1.1 1 through 2 - Figure 5.2

Upon execution of the input program the program will loop over all participating patients. Initially a patient is instantiated. As a patient is associated by one or more meals and by one or more set of parameters each patient is associated with all available meals and parameters. The meals and parameters will be loaded from a predefined directory. If the number of meals or parameters available are zero the program will throw an `InsufficuentInputException` and terminate. The `InsufficuentInputException` is thrown specifically if bad input is received. In the case that the user specifies to few or to many meals than the number of days the trial runs the program will not terminate but instead take this into consideration by either reusing or skipping meals. In the same manner if the number of patients defined is lower than the number of available meal profiles and parameters, the program will simply duplicate the existing patient meals and parameters and reuse them for other patients. This can be very useful if one seeks to test the same meal profile for many different patients. Since every meal profile and every parameter is associated with a time it is possible to point each meal and parameters to a calender date using the datastructure `DateTime`. Since the time interval N and the step size T_s is known the idea is to define the trial length as a calender interval. Hereby it is possible to find the number of minutes between the start and end time such that a time interval $[t_0, t_{max}]$ can be defined. Thanks to the Java library Joda time [9] it is possible to point a location in the calender for each meal and each set of parameters. Each information will be stored for each virtual patient in the data structure as depicted in the domain model in Figure 5.1 in Section 5.1.

5.2.1.2 3 through 4 - Figure 5.2

Once all the virtual patients has been associated with the content loaded from the CSV files the next step will be to consider whether the user have specified an optional vector with parameters. This is shown as the logical alternative condition. `parametersToChange:addToArray` is a the reference to instance of

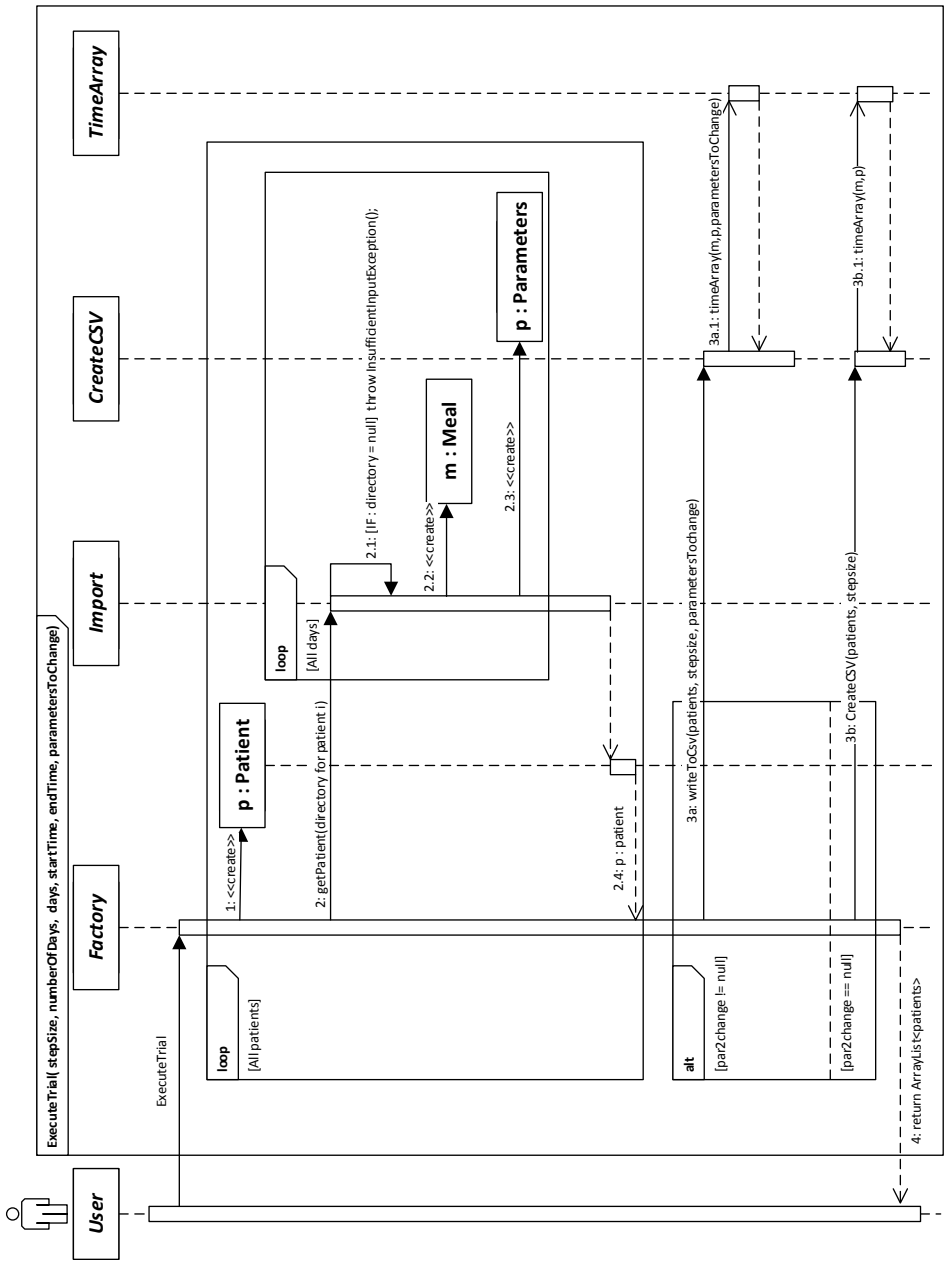


Figure 5.2: *Sequence diagram for the input handle.*

an `addToArray`. `addToArray` is a very useful object that ensures that if the user specifies one or more parameters to be changed these will be incorporated in the output file. Method overload has been introduced with the calls `timeArray(m:ArrayList<Meals>, p:ArrayList<Parameters>, pararameters:double[][])` and `timeArray(m:ArrayList<Meals>,p:ArrayList<Parameters>)` which then transforms each `Patient` into a transformed output. The calender system makes it possible to discretize the time interval into K intervals with a constant length of T_s such that $n = [0, 1, \ldots, K-2, K-1]$. In practice this means that the n elements can be stored such that a time array T can be generated: $T = [t_0, t_0 + T_s, \ldots, t_{max} - T_s, t_{max}]$. As exporting starts each meal and parameters are simply associated with a location in the vector T and printed to that location in the csv file.

This hereby ensures that the system, described in the next section, receive input which have been piecewise constant descritized. The reasoning behind this discretization is elaborated in details in Section 3.1.5. The proceeding Subsection 5.2.2 will describe how the input will be handled in the system.

5.2.2 System

Before execution of the simulator it is assumed that the input has already been discretized as described in Subsection 5.2.1. The input must be placed in a csv file format in a specified folder which the system will gather data from. Figure 5.3 depicts the sequence diagram for the system. The system is the most computational heavy of this entire software. This is due to the fact that the numerical methods will be conducted here.

5.2.2.1 1 through 3 - Figure 5.3 and Figure 5.4

Upon execution the `startText()` will print all relevant information to console. From here a loop will run through each patient. The function `run()` initiates the simulation of each patient. Each patient is executed as a individual thread. Each thread has the same code tot execute and Figure 5.4 indicates the sequence diagram for an individual thread. Since each patient is executed in a concurrent[1] manner the methods accessing shared variables in `run()` must be thread-safe. Section 5.3 goes into details on this matter. From here a system of ODEs are created for each patient and the steady state, as described in Section 3.1.5, is computed. From here all elements in the discretized input time array T will be

[1]Concurrent will be used to state the simultaneous execution of one or more threads. Concurrency must be supported by the processor to be executed parallel.

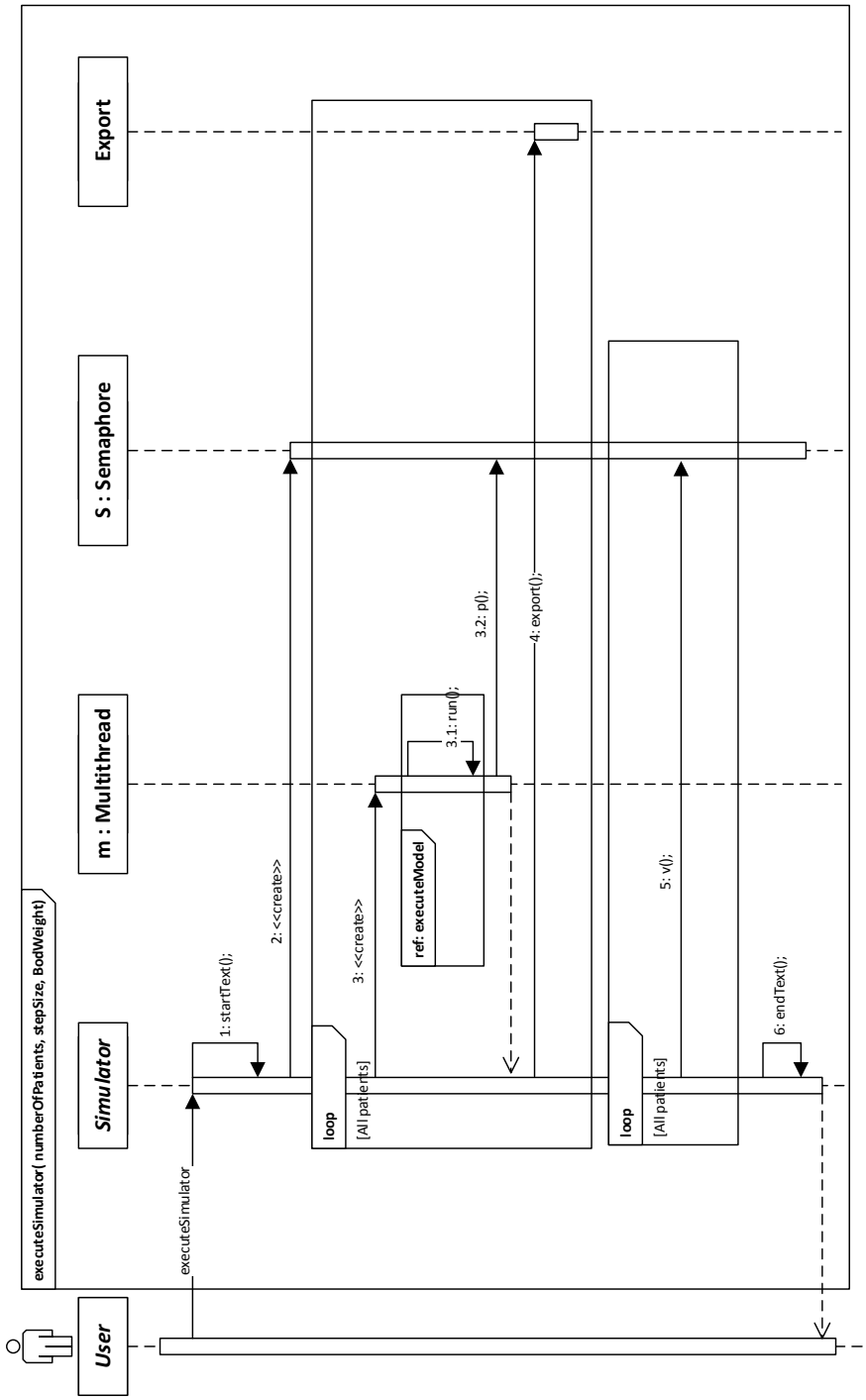


Figure 5.3: Sequence diagram for the system.

looped through to conduct the simulation corresponding to each time for the CGM to make a reading. At each step the insulin controller will be evaluated such that a decision on insulin infusion can be made.

5.2.2.2 4 through 6 - Figure 5.3

Even though each element of `run()` is concurrent the function `export()` will export each patient information once each thread terminates. Fortunately each print is associated with a patient number such that the virtual patients remain in the correct order. It is possible to change the program to save the simulations to the temporary memory instead of writing to a csv file. This should only be done if the user is certain that the memory would fit the data. The configuration-system will be explained in details in Section 5.4. However, the program must synchronize threads in order to terminate properly. The synchronization process is done by alignment of the semaphore that is controlled by `p()` and `v()` and will be described in details in Section 5.3. As a concluding part of the system the program will write a simulation summary to console by execution of the `endtext()`.

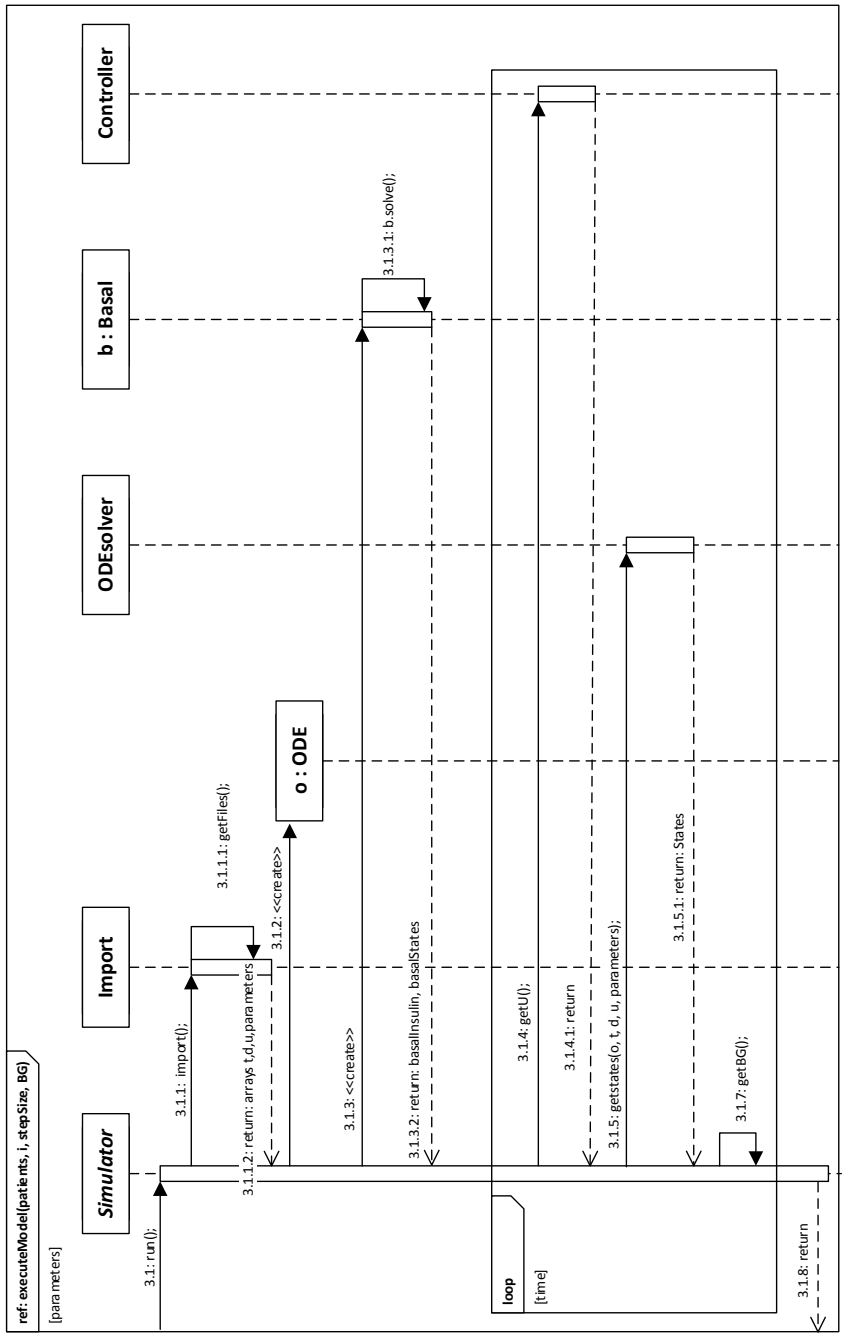


Figure 5.4: *Sequence diagram for the execution of* ***run();***

Table 5.4: *Selected evaluation calls with input and output.*

Call [`Metrics.`]	Input	output
ADRR	Blood Glucose	[*LBGI HBGI ADRR*]
inZone	Blood Glucose, min, max	*TIZ*
belowZone	Blood Glucose, min	*TB*
aboveZone	Blood Glucose, max	*TA*
hb1ac	Blood Glucose	Hb1Ac
auc	Blood Glucose, step size	AUC
MAGE	Blood Glucose, step size	MAGE

5.2.3 Output

The final stage of the software package is an evaluation tool package. This serves as a purpose to conduct all the evaluations as described in Section 3.3. Essentially this part of the program can be seen as a library containing the relevant metrics. Table 5.4 gives an overview of the most important of these evaluation metrics, the call and the input needed. As with other libraries in Java the call is `[library name].method`.

The next section will go into details on the implementation of multi-threading to reduce the runtime of the most computational heavy elements of the software.

5.3 Multi-threading

In recent time multi core processors has become widely standardized both in the academic field as well as in personal computers. This leads to the development of multi-threaded[2] software which seeks to take full advantage of the individual cores. Unlike sequential programming where a program is executed in a series of steps a multi-threaded program is the process of executing steps simultaneously. The operating system will distribute the various tasks of the program across the cores [36] as shown in a sequential manner in Figure 5.5 and in a multi-threaded manner in Figure 5.6.

The input and output element in this software is not as computational heavy as the system itself. The system will handle large amounts of data and require numerical methods which are computational demanding. Implementing multi-threading potentially can reduce the run time if compatible with the hardware. Therefore, the software has been decomposed such that each virtual patient will

[2]The term concurrency may also be used. The term indicates when several computations are executed simultaneously.

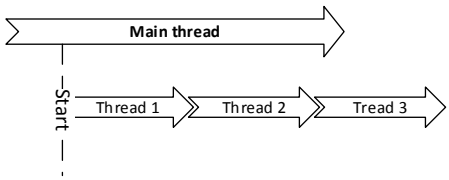


Figure 5.5: *Sequential execution of program*

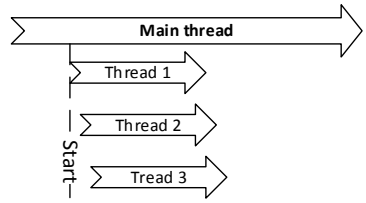


Figure 5.6: *Non-synchronized multi-threaded execution of program*

be executed as an individual thread. The reason for making the decomposition on patient basis is that each patient symbolizes an individual process and they are independent of each other thus suited for multi-threading. However, solving problems in a concurrent manner leads to challenges which needs to be handled as the concurrency is implemented. This leads to the usage of different tools and methods which ensures proper execution of a multi-threaded program. The basic method for securing individual access control is the usage of locks and synchronization methods.

5.3.1 Locks

Locks is a simple synchronization method used to communicate between the individual threads. The goal with a locking mechanism is to limit the access to certain elements of a code by mutual exclusion. The locking mechanism has been implemented in the method `getFiles()` shown in Figure 5.4 and seen in Listing 5.1 by two operations. First a locking operation, followed by an unlocking operation using a semaphore as a key. The first thread to enter this specific piece of code will be allowed access and lock this piece of code to other threads. Other threads will meet the lock and will have to wait until the first thread unlocked. The first thread will identify the files available in a specific folder. Thus, the first thread has exclusive access to the call `listOfFiles = folder.listFiles()` as seen in the bottom of Listing 5.1. Hence, the first thread identifies to all the other threads which files are available in the specific folder. All other threads will instead read the value stored in the variable `listOfFiles`. The Java implementation of the lock can be seen in Listing 5.1.

Listing 5.1: *Lock implemented in the call of getFiles().*

```
1  // Define key – one thread may enter
2  private static Semaphore fileLock = new Semaphore(1);
3  private static File[] listOfFiles;
4
5
6  public static File[] getFiles() {
7
8      if (listOfFiles != null) {
9          // return the File[] array with the files.
10         return listOfFiles;
11     }
12     try {
13         // Lock. one thread will pass.
14         fileLock.P();
15         } catch (InterruptedException e) {
16             // If calling semaphore fails print stacktrace.
17             e.printStackTrace();
18         }
19
20     // main place of n – 1 thread to terminate.
21     if (listOfFiles != null) {
22         fileLock.V();
23         return listOfFiles;
24     }
25     // Only one thread will enter
26     File folder = new File(Simulator.exportDirectory);
27
28     // Create file array consisting of all filenames.
29     listOfFiles = folder.listFiles();
30
31     // Store files in accordance to name
32     Arrays.sort(listOfFiles);
33
34     // Unlock
35     fileLock.V();
36     return listOfFiles;
37 }
```

Algorithm 5 Dijkstra's Semaphore

```
1: READ s ← initial value of s

2: P():
3: while s = 0 do
4:     wait()
5: end while
6: s ← s − 1

7: V():
8: s ← s + 1
```

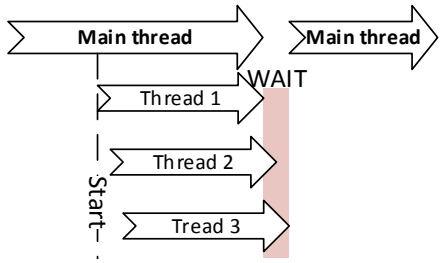


Figure 5.7: *Final synchronization of threads*

5.3.2 Synchronization of threads with Dijkstra's Semaphore

In extension of the locking mechanism is the usage of a semaphore. Previously we used the semaphore as a locking mechanism, but semaphore can also be used for synchronization. Dijkstra's Semaphore is a very easy-to-implement method for synchronizing the individual threads and introduced in 1968 by Edsger Dijkstra [24]. The principle behind the semaphore is a locking mechanism and the classical implementation can be seen in Algorithm 5.

In short, the algorithm relies on a shared variable, `s`, and two functions, `P()` and `V()` both which can manipulate the value of `s`. When a thread executes `V()` the function will increment `s` while calling `P()` will decrement `s` under the condition that $s > 0$. In the case that $s = 0$ the thread will be forced to wait until `V()` will be called. In this software the semaphore has been used to synchronize the threads. More specifically the main program will wait until `P()` and `V()` has been called a similar number of times. Figure 5.7 shows how the principle would work extending from the two previous figures. Referring back to the sequence diagram in Figure 5.3 from Subsection 5.2.2 the main thread is executed in a sequential manner. The method 3.1. `run()` shown in Figure 5.4 shows the

simulation of each individual thread that is executed concurrently. The main thread must be stalled as long as threads are being executed to ensure that the main thread will not terminate before all individual threads has terminated. The semaphore `V()` operation is called once each simulation is started, while the `P()` operator is called just after the simulator. This ensures the main thread will not terminate before all individual threads has terminated. The implementation can be seen in Java in Listing 5.2 and Listing 5.3.

5.3.3 Non-blocking methods

As a final tool in the implementation of multi-threading is the usage of non-blocking algorithms. This ensures that no conflict occurs between the different threads. Additionally non-blocking methods eliminate the waiting time unlike the locking and synchronization mechanisms. Non-blocking algorithms are used through hash tables to store data from the individual threads. Hash tables has been object of research within the last decade and recently newer more efficient methods has been implemented in the standard libraries [41]. The hash table data structure in Java which provides thread-safety is `ConcurrentHashMap` which has been used to link each run with a specific data allocation.

5.3.4 Implementation and Expectations

The implementation of multi-threading has been done with the class `Thread`, which is a build-in class in Java while the implementation of locks and synchronization has been specifically coded for this software. Listing 5.2 and Listing 5.3 shows the implementation in Java. Note that some of the code content has been omitted for simplification purpose. From Listing 5.2 the main thread is located. Once it enters the first `for` loop each thread will be executed immediately by the command `m.start()`. Note from Listing 5.3 that each thread calls the `p.V()` operation after execution has been conducted. Then, as the main thread exit from the loop it will initiate the next `for` loop containing the `p.P()` operation. Here the main thread will stall until all threads has been executed. Thus, synchronising the entire operation.

The Conversion of a sequential algorithm or program to a multi-threaded, as previously explained, can increase the running time theoretically defined by Amdahl's law [22]. Amdahl's law states that given the fraction F_s, which is the computing time needed for the sequential part of the program, and the parallel fraction $1 - F_s$, where $F_s + (1 - F_s) = 100\%$ then the parallel execution time is

Listing 5.2: *Implementation of semaphore that will force wait for main thread.*

```
// ...
// Instantiate semaphore operation
Semaphore p = new Semaphore(0);
// Main loop which will call the Multithread class with ...
    interface thread.
for (int i = 1; i ≤ numberOfPatients; i++) {
    // Create new multi thread m
    MultiThread m = new MultiThread(BW[i − 1], i, stepSize, p, ...
        BGstorage);

    // run();
    m.start();
}
// This for loop will force main thread to wait.
for (int i = 0; i < numberOfPatients; i++) {
    // Decrement semaphore
    p.P();
}
// ...
```

Listing 5.3: *Multithread class extending Thread.*

```
public class MultiThread extends Thread {
    // Private variables has been omitted for simplification
    public MultiThread(double BW, int i, int stepSize, Semaphore p,
        double[][] BGstorage) {
        // Content omitted for simplification
    }
    public void run() {
        try {
            // Run virtual patient simulation
            BGstorage[i − 1] = Simulator.executeModel(BW, i, ...
                stepSize);

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
        // Increments semaphore
            p.V();

        }

    }
}
```

the sum of the part that is sequential and the part that can be made parallel:

$$T_p = T\left(F_s + \frac{1 - F_s}{P}\right) \tag{5.1}$$

Where P is the number of cores available and the theoretical runtime improvement is given by:

$$S_p = \frac{T}{T\left(F_s + \frac{(1-F_s)}{P}\right)} \tag{5.2}$$

However, introducing concurrency to a code introduces communication overhead lowering the actual speed up such that the final speed-up ratio will be defined as:

$$S_p = \frac{T}{T\left(F_s + \frac{(1-F_s)}{P}\right) + T_C} \tag{5.3}$$

Where T_C denotes the communication overhead related to concurrency of the code. In this thesis the running time analysis using Oracle Analyzer® has show that $F_s = 0.01$. From (5.2) we know that as $p \to \infty$ the parallel execution time approaches that of the sequential runtime. This leads to the conclusion that this program has a theoretical running time improvement of no more than $S_p \leq \frac{1}{F_s} = 100$ from the sequential running time assuming that an infinite number of cores were available. While this is a theoretical limit the actual speed-up ratio would likely be much lower due to two reasons. First, unexpected overhead in context with concurrent programming denoted as the communication overhead T_C. Overhead is a term used for the time required to coordinate the individual threads and it potentially will have a large impact on the final computation time. Secondly, the Java class `Thread` cannot distribute computational burdens across different computers. Hence, the program may not take full advantage of the hardware available.

5.4 Configuration-system

A final consideration in the software is the implementation of a configuration system. In short the configuration system is implemented with the `Properties` class which makes it very easy to load and store settings for a program [1]. The configuration file must be located in the same folder as the `.jar` file[3] and the program will automatically look into the configuration file and find the

[3] An exception to this is when executing from Matlab, then the configuration file must be in the same folder as the Matlab script itself.

Listing 5.4: *Configuration system by using the Properties class.*

```
// ...
Properties prop = new Properties();

// Instantiate a File input
FileInputStream configFile = new ...
    FileInputStream("configuration.xml");

// Load from configuration file
prop.loadFromXML(configFile);

// extraction example:
String solver = prop.getProperty("solver");
// ...
```

relevant values. The configuration file is based on XML, originally developed as a data sharing system, and thus a entry key is linked to some value `<Entry key>[value]</entry>` such that the program looks for a key and return the value as a string. Some example of configurations possible in the configuration file `configuration.xml`:

```
<entry key="multithread">Y</entry>
<entry key="solver">dp54</entry>
<entry key="minstep">1.0e-6</entry>
```

Here it would be possible to change the program from running parallel to sequential by changing the value `Y` under the key `multithread` to `N`. More configurations can be modified including the numerical integrator to use (e.g. `dp54` or `Euler`) or the minimum step (default is `1.0e-6`). Other elements that can be changed in the configuration systems is the PID controller and the related parameters. Finally, the working directories for import and export are also defined in the configuration.xml file. The configuration system was made such that the software could have some predefined default values while ensuring these default values could be easy changeable in the future. The `Properties` class showed very useful for this purpose especially due to the entry key feature that exist in XML and its simple way of implementing and using. The usage of the configuration can be seen in Listing 5.4 where the variable `solver` is defined.

5.5 Summary

In this chapter the framework and software capable of testing an AP has been proposed. First, a descriptive analysis was made to identify the requirements needed for the software. These requirements was then implemented in Java and divided into three main functionalities: Input, System and Output. Dividing the software into these three functionalities ensured that each specific part could be used separately. The software was then optimized by implementing multi-threading and a configuration-system to easily change parameters.
The next chapter will test the software in two ways. First, a test of the software's ability for testing of an AP. Second, a test of the performance gain from multi-threading.

Chapter 6

Experimental testing

The objective of this chapter is twofold. First, Section 6.1 gives two case studies for in silico testing of an AP on a virtual cohort. Second, Section 6.2 give a test of the performance gain by implementation of multi-threading.

6.1 In Silico Testing for an Artificial Pancreas

The overall scope with this section is to show a user case of this software. Additionally, it also gives an impression of the capabilities of the software. This section is organised as follows. Subsection 6.1.1 defines the testing protocol. Subsection 6.1.2 explains the software execution process. Subsection 6.1.3 presents the results and finally, a discussion based on the findings will be given in Subsection 6.1.4.

6.1.1 Testing Protocol

For this in silico case study following protocol will be used:

- A cohort of 50 patients suffering from T1D, as described in Subsection 3.1.6, are used for this trial.
- Two treatment options will be compared on the cohort. Using the method from Section 4.3 this trial compares Feed Forward versus No Feed Forward.
- The Trial will be conducted over a period of 35 days (5 weeks).
- The starting time will be 18-03-2014 at 00:00, and the end time will be 22-04-2014 at 23:59.
- A 7 day meal profile as shown in Table 6.1 to Table 6.3 will be used for each patient. The consecutive 4 weeks will simply be repeated.
- A sampling time of 5 minutes will be used, that is $T_s = 5$.

6.1.2 Execution of Software

As the software is executed from Matlab® some description of the actual execution process will be made. The explanation is ordered as follows:

1. Setting up the configuration system.
2. Initialisation of the trial.
3. Defining the virtual cohort.
4. Execution of simulation.
5. Evaluation of simulation results.

A full user guide containing more detailed information and descriptions can be seen in Appendix D.

1. Configuration system
At first a working directory for the software must be defined. This is important since the software will import meal and parameter files for the predefined virtual patients. There are four working directories of importance and they can all be modified in the configuration.xml file. One of these four working directories will be introduced here:

```
<entry key="directory">[your directory]/matlab/Patients/Patient</entry>
```

`directory` is the directory for the location of patient data and also defines the name of the sub-folder containing patient data. Additonal directories includes `load_directory`, `export_directory`, and `final_export_directory`. Each directory has a specific purpose e.g. directory for export to CSV. Refer to the user manual in Appendix D for a comprehensive description.

2. Initialisation
In Matlab it is possible to add the content from the .jar files to the Java Virtual Machine (JVM) nested within Matlab as shown in Listing 6.1.

Listing 6.1: *Initialization of .jar files and functionalities to Matlab JVM*

```
javaaddpath('..\binJava\Trial.jar') % Import input
javaaddpath('..\binJava\patient_Simulator.jar') % Import system ...
    / output
import functionality.*; % Import functionality package
import domain.*; % Import domain package
import simulationModels.*; % Import simulation model package
import inputOutput.*; % Import input / output package
```

The import statements is needed to ensure that all content from the .jar files are imported to the Matlab workspace and consequently saved to memory. This enables Matlab to access all classes and methods from the .jar files. Next, is the definition of the trial which we seek to perform. Listing 6.2 shows the definition of the trial.

Listing 6.2: *Trial definition*

```
stepSize            = 5;  % Step size, Ts
startHours          = 00; % Start time (HH)
startMinutes        = 00; % Start time (MM)
endHours            = 23; % End time (HH)
endMinutes          = 59; % End time (MM)
numberOfPatients    = 50; % Number of patients to participate
days = 35; % The number of days for the trial to run
```

As of now the trial has been defined and the next step is to define the virtual cohort.

3. Virtual cohort
Since all virtual patients will have the same meal profile, a single predefined folder containing 7 CSV files has been made. Each file has a layout as shown below and are stored in the directory defined by the configuration file. For a more detailed description on how to set up meals and physiological parameters please refer to the user manual in Appendix D.

```
Time;Duration;CHO
07:00;10;45
10:30;20;20
12:30;20;60
17:00;10;10
19:30;20;60
```

Table 6.1, Table 6.2 and Table 6.3 shows the content of each of the 7 CSV files.

Table 6.1: *Meal profiles for Monday, Tuesday and Wednesday.*

Monday			**Tuesday**			**Wednesday**		
Time	Duration	Amount	Time	Duration	Amount	Time	Duration	Amount
07:00	10	45	07:00	10	10	07:00	10	45
10:30	20	20	10:30	10	30	10:30	20	20
12:30	20	60	13:30	10	75	12:30	20	60
17:00	10	10	18:30	10	75	17:00	10	10
19:30	20	60	21:30	10	15	19:30	20	60

Table 6.2: *Meal profiles for Thursday and Friday.*

Thursday			**Friday**		
Time	Duration	Amount	Time	Duration	Amount
07:10	15	30	07:10	15	30
11:30	20	60	10:00	5	25
15:00	10	30	12:30	20	60
19:30	20	60	15:00	10	10
			19:30	20	70

Table 6.3: *Meal profiles for Saturday and Sunday.*

Saturday			**Sunday**		
Time	Duration	Amount	Time	Duration	Amount
09:10	15	45	11:00	25	75
11:30	20	10	15:30	20	50
12:30	20	10	19:00	10	55
13:30	20	10	21:30	20	10
15:30	20	10			
18:30	20	10			
20:30	20	50			

The next step is to define the physiological parameters. The physiological parameters will follow the virtual cohort definition from Subsection 3.1.6. This means that the parameters will vary between individuals and oscillate over time within individuals. First, is to notify the software on which parameter to vary as shown in Listing 6.3.

Listing 6.3: *Parameter definition part 1. Defining the parameter names to be changed.*

```
par = [java.lang.String('F01');
     java.lang.String('EGP0') ;
     java.lang.String('K12');
     java.lang.String('SI1');
     java.lang.String('SI2');
     java.lang.String('SI3');
     java.lang.String('ka1');
     java.lang.String('ka2');
     java.lang.String('ka3');
     java.lang.String('Ke');
     java.lang.String('VI');
     java.lang.String('VG');
     java.lang.String('TauD');
     java.lang.String('TauS');
     java.lang.String('AG')];
```

After defining the string array containing the names of the parameters which will be variable, the next step is to feed this into the software as seen in Listing 6.4. Note that one would easily be able to change what parameters to vary by changing the name variable `par` from Listing 6.3 and amending the `GenerateParameters` in Listing 6.4.

Listing 6.4: *Parameter definition part 2. Adding the parameters software.*

```
% Notify program of parameters by instantiating AddToArray
p = AddToArray(par);

% Generate parameters
parameters = GenerateParameters(p,numberOfPatients,days,stepSize);
```

For this run, patient 1 has been withdrawn from the cohort for descriptive purpose. Patient 1 has a BW of 77.03 kg and a CR of 16.18. The physiological parameters which are constant over time for virtual patient 1 can be seen in Table 6.4. Figure 6.1 and Figure 6.2 shows the variable parameters for virtual patient 1. While Figure 6.1 shows the variation of the insulin sensitivities S_{I1},

Table 6.4: *The physiological parameters which are constant for patient 1.*

	V_I	V_G	k_{a1}	k_{a2}	k_{a3}	τ_d	τ_s
Value	0.11993	0.13748	0.00551	0.06674	0.02995	39.99293	55.57602

S_{I2}, S_{I3} and the endogenous glucose production EGP, Figure 6.2 shows the variation in glucose consumption $F01$, the transfer rate parameter k_{12}, the insulin elimination rate k_e and the food absorption constant Ag. The parameters are shown over a 24 hour period.

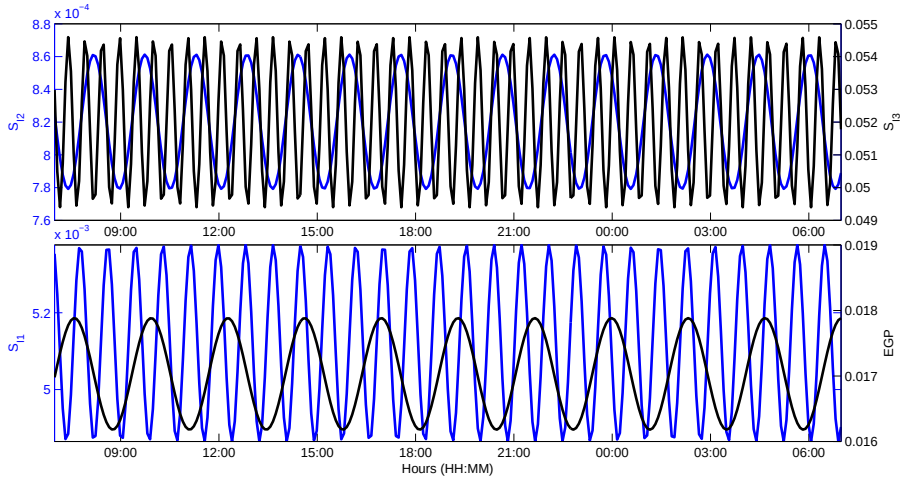


Figure 6.1: *This figure shows four different physiological parameters in a 24 hour period for patient 1. From the top sub plot the blue line indicates S_{I2} and the black line, S_{I3}. In the lowest plot the blue line indicates the variation in S_{I1} and the black line the variations in EGP.*

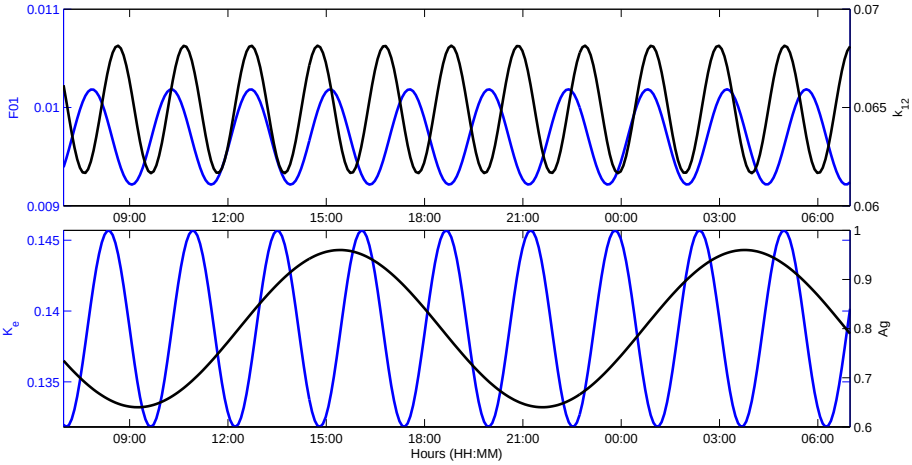


Figure 6.2: *This figure shows four different physiological parameters in a 24 hour period for patient 1. From the top sub plot the blue line indicates the variation in $F01$ and the black line k_{12}. The lowest plot the blue line indicates the variation in k_e and the black line the variations in Ag.*

4. Simulation execution
As of now the program is informed of the virtual cohort by definition of the meal profiles and physiological parameters and the next step is to execute the program as shown in Listing 6.5

Listing 6.5: *Program execution.*

```
% First the input will be created
Execute.CollectPopulation(numberOfPatients, ...
    stepSize,days,startHours,startMinutes,endHours,endMinutes,BW,p);

% Second the simulation will be conducted
Simulator.execute(numberOfPatients,stepSize,BW);
```

The program will now execute and the results will be available as variables in the Matlab workspace. If the configuration file has not been defined properly or faulty input has been provided the program may throw an exception and terminate. Appendix E gives an overview of the most common exceptions and related solutions.

5. Simulation evaluation
As a final element is the call of the various output evaluation parameters. Additonally, some standard plot commands has been made. The calls are shown in Listing 6.6.

Listing 6.6: *Program evaluation.*

```
% Evaluate performance metrics
PMetrics(1:numberOfPatients,SaveMethod,Metrics,stepSize);

% Plot commands
plotindividual
plot_all
Bench_plots
```

The plots will be printed to an output folder and the results will be shown in the next section.

6.1.3 Results

This subsection presents the results generated using the test protocol defined in the previous subsection. Once again, to ease understanding virtual patient

Table 6.5: *Percentage of time in different zones for the single patient case.*

BG Range [mmol/L]	**No Feed Forward**	**Feed Forward**	**Change** [pp]
$10 >$	20%	15%	-5
$8 - 10$	10%	11%	1
$4 - 8$	44%	49%	5
$3 - 4$	14%	16%	2
$0 - 3$	12%	9%	-3

Table 6.6: *Performance metrics for virtual patient 1.*

	No Feed Forward	**Feed Forward**
$\overline{BG}$	6.78 $\pm$(3.81)	6.48 $\pm$(3.24)
low	1.45	1.77
high	22.7	19.65
HBA1c	5.19	5.00
ADRR	29.17	23.85
MAGE	20.11	15.82

1 will be considered as an initial example. Thereafter, the simulation results of the entire cohort will be provided.

6.1.3.1 Considering Patient 1

Patient 1 has a BW on 77.03 kg and a CR on 16.18. The two control strategies are performed on the same patient. The results for patient 1 can be summarized in Table 6.5 and Table 6.6. The two tables essentially state that the control strategy in both cases gives a bad performance, that is, a poor glycemic control. Noticeable is the hypo- and hyperglycemia identified from Table 6.5 as the amount of time spent below 3 mmol/L and above 10 mmol/L, respectively. From Table 6.6 noticably is the risk parameter ADRR which indicates a medium to high risk for the both control strategies since the value is in the range 20 to 40. Additionally parameter MAGE indicates a large glycemic variation as the average change in a 6 hour period is in average 20.11 and 15.82 respectively. Finally, the low BG values indicate hypo glycemia in both cases. Ultimately, the performance metrics illustrates very well the concerns in the treatment for virtual patient 1. The poor glycemic control can be verified by looking at Figure 6.3 and Figure 6.4.

Figure 6.3 and Figure 6.4 shows the first 7-31 hours of the trial using the insulin administration strategy without and with Feed Forward. Figure 6.3 and Figure 6.4 has three subplots. The top part identifies the BG value and the CGM signal. The CGM signal is the input to the insulin administration strategy which output is seen in the lowest plot in both figures, namely the insulin

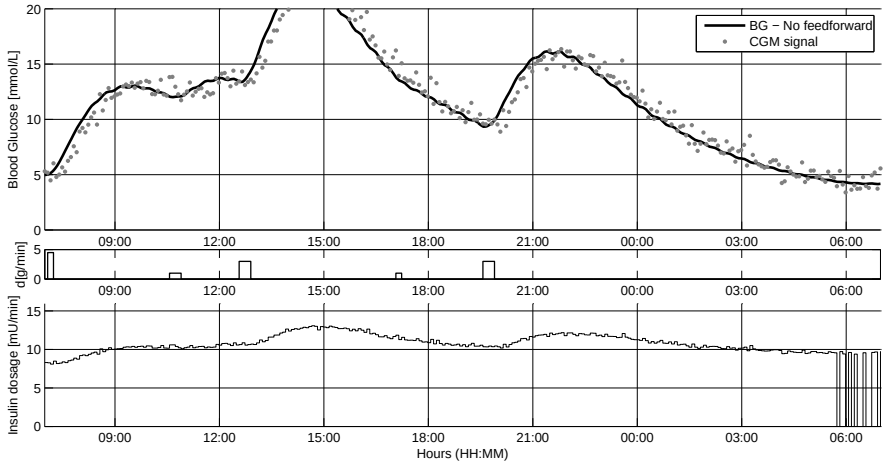


Figure 6.3: *This figure shows the simulation during the first day without Feed Forward. Poor glycemic control is evident. The top part shows the BG and the CGM signal. The middle plot shows the carbohydrate intake and the lowest plot shows the control output.*

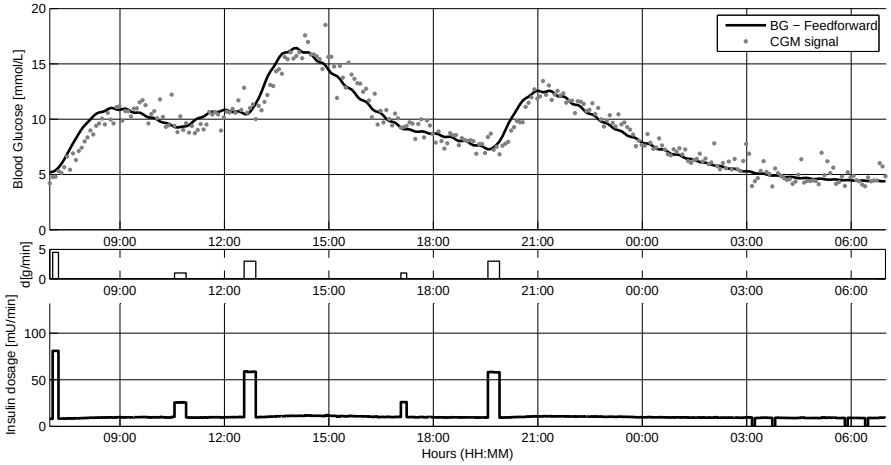


Figure 6.4: *This figure shows the simulation during the first day for patient 1 using Feed Forward. The top part shows the BG and the CGM signal. The middle plot shows the carbohydrate intake and the lowest plot shows the control signal. Better glycemic control can be seen from this figure.*

dosage u_k. The middle plot indicates the meal profile of that particular day.

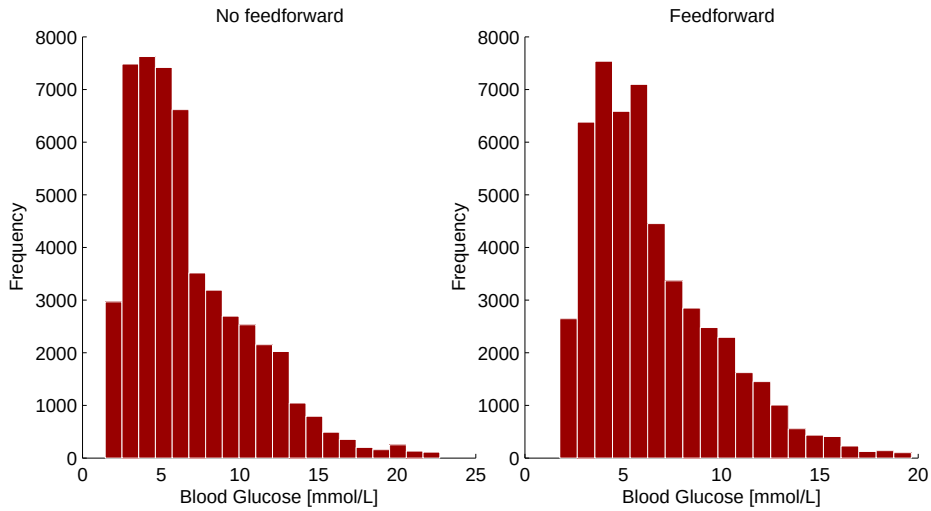


Figure 6.5: *This figure shows a histogram of the BG level for the two control strategies. Both indicate poor glycemic control. Feed Forward does show a bit more tight glycemic control.*

In both cases the patient can be identified as having poor glycemic control. Nevertheless, the glycemic control with Feed Forward does seem slightly better than the case without Feed Forward. This is confirmed by the BG histogram in Figure 6.5. Figure 6.5 identifies a more frequent sampling around the target BG level of 5 mmol/L and especially less BG readings below 3 mmol/L and above 20 mmol/l for the case with Feed Forward. To summarize the performance of virtual patient 1; the glycemic control is bad for both strategies. This is both identified from the actual BG reading but additionally also from the performance metrics. Nevertheless, Feed Fordward does perform better and does to some extend avoid hyper- and hypoglycemia. Next subsection present the result for the entire cohort.

Table 6.7: *Percentage of time in different zones for the entire virtual cohort.*

BG Range [mmol/L]	**No Feed Forward**	**Feed Forward**
$10 >$	17%(±5%)	14%(±6%)
$8 - 10$	11%(±2%)	10%(±2%)
$4 - 8$	41%(±7%)	46%(±10%)
$3 - 4$	14%(±2%)	15%(±3%)
$0 - 3$	17%(±7%)	14%(±8%)

6.1.3.2 The Entire Virtual Cohort

The entire cohort has a $\bar{BW} = 73.26$ kg(±14.81 kg) with a minimum value of 39.98 kg and a maximum value of 104.80 kg. The cohort has a $\bar{CR} = 18.89(\pm 5.52)$ with a minimum value of 12.25 and a maximum value of 40.65. The simulation can be summarized into Table 6.7 and Table 6.8. We will consider a one sided paired Z-test for comparison of the two methods using a significance level, $\alpha = 0.05$. We construct the statistical hypothesis assuming that the Feed Forward method performs better, thus the reason for using a one sided Z-test. Overall, Table 6.7 and Table 6.8 show a level of variation across the entire cohort. Table 6.7 does show a better performance for Feed Forward in the mean cases but the standard deviation pictures that much variation exist an that performance are not good for all cases. Table 6.8 also show some performance metrics to be significantly different (highlighted with bold). These are HbA1c, MAGE and HBGI. These metrics represent a statistical significant performance improvement for the Feed Forward case. However, ADRR and LBGI does not show to be significantly different between strategies. This is an interesting observation. This means that the risk parameters ADRR and LBGI does not acknowledge Feed Forward to perform better. As one look at Figure 6.6 and Figure 6.7 showing the two scenarios for the first 7-31 hours Figure 6.7 appears to have better glycemic control. This is in contradiction with the risk parameter ADRR from Table 6.8. Nonetheless, this may indicate that the risk parameter ADRR are good to identify the actual performance of the treatment. the top plot in Figure 6.6 shows the mean BG across the cohort with the errorbars representing the standard deviation of the entire cohort. The lower part of Figure 6.6 represent the mean insulin infusion with the errorbars representing the standard deviation for the entire cohort. Figure 6.6 is for the case with no Feed Forward. Figure 6.7 is the case with Feed Forward.

An interesting parameter is the HbA1c as seen plotted as a histogram in Figure 6.8. Even though both groups is in risk of hyper- and hypo-glycemia, denoted by several of the previously mentioned performance metrics, the HbA1c values are normal. The Feed Forward strategy gives a slightly better HbA1c - though both are within normal range. This is an interesting observation and may show that HbA1c, in this case, does not consider risk very well. Intuitively this makes

Table 6.8: *Performance metrics of entire cohort.*

	No Feed Forward	Feed Forward	Paired Z-test
HbA1c	4.98(±0.24)	4.84(±0.26)	$p = 0.0026$
MAGE	20.17(±7.28)	17.24(±6.49)	$p = 0.017$
ADRR	60.60(±72.38)	52.08(±66.30)	$p = 0.2709$
LBGI	10.42(±6.52)	9.06(±6.26)	$p = 0.1446$
HBGI	4.26(±2.24)	3.44(±2.16)	$p = 0.0384$

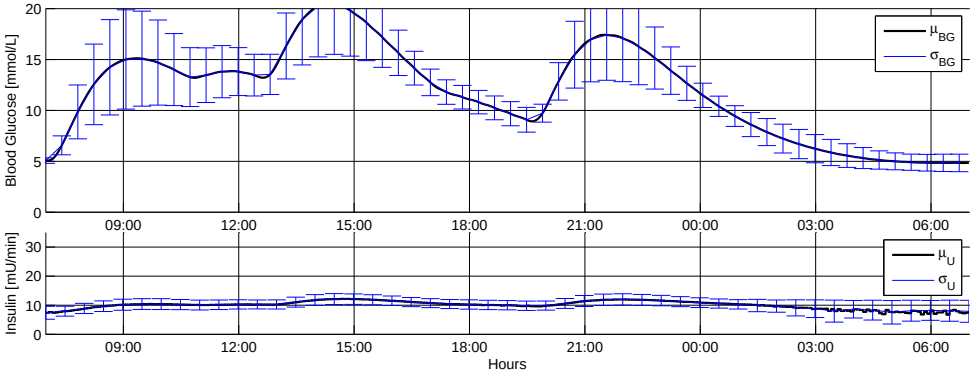


Figure 6.6: *This figure shows the average BG level with the standard deviation in the case without Feed Forward.*

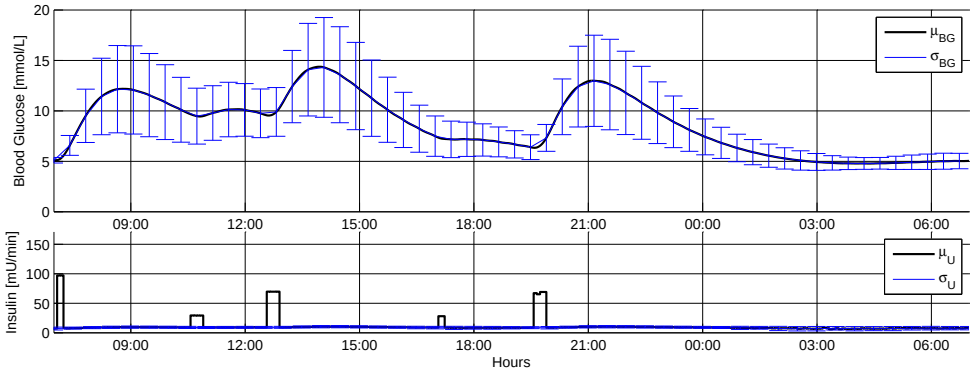


Figure 6.7: *This figure shows the average BG level with the standard deviation for the Feed Forward case of the virtual population.*

sense since HbA1c is estimated from the mean BG level which are within glycmic range for all individuals. Figure 6.9 summarizes all parameters into a single plot and show the mean BG level as a function of the risk parameter ADRR. Additionally, the size of the circles notes the glycemic variability by the standard

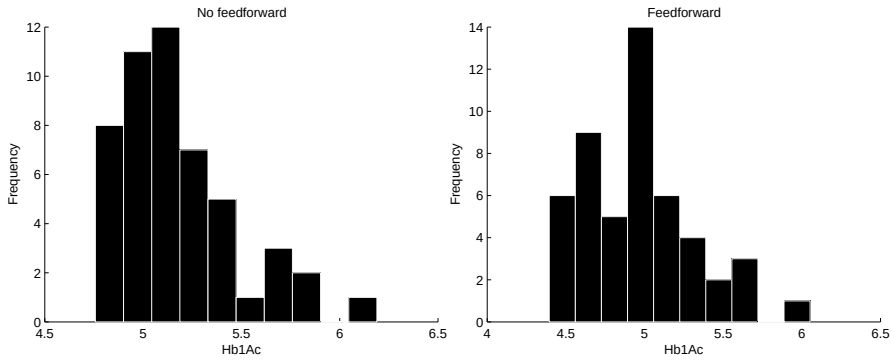


Figure 6.8: *This figure shows a histogram for the HbA1c frequency for each of the two groups.*

deviation of the BG readings. Overall, it can be seen that a selected number of patients has a very good performance as the ADRR is below 20. This is the group which has excellent glycemic control. The rest of the group appears to be in risk both for the Feed Forward and the Non Feed Forward case. From Figure 6.9 it is also noted that there exist a clear connection between the BG variation and the level of risk represented by ADRR. The subsequent Section will give a discussion of the results presented.

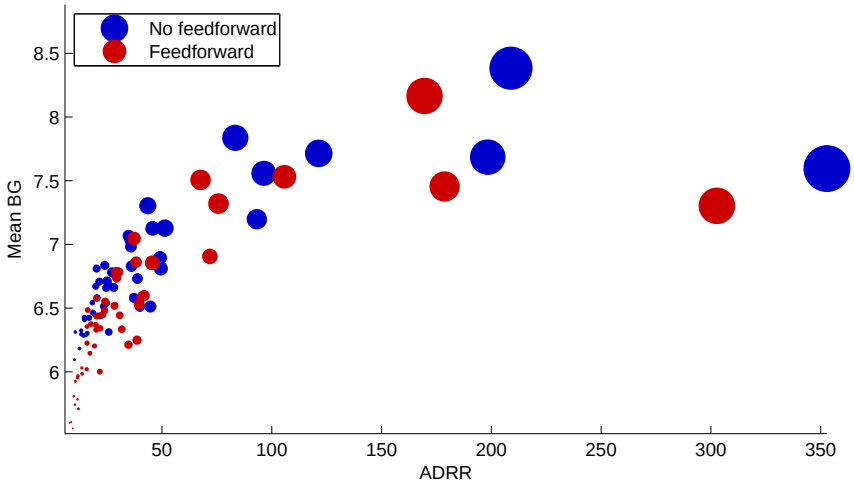


Figure 6.9: *This figure shows the two groups and their mean BG value as a function of the ADRR risk measure. Furthermore, the size of the dots represent the BG standard deviation.*

6.1.4 Discussion of the Results

Previous section showed a Proof-Of-Concept implementation of an AP tested on a virtual cohort. The results showed that it is possible to test various control strategies against each other across individuals while ensuring inter- and intra-variability based on key metabolic parameters. It was identified that only a limited number of the virtual patients actually obtained a good glycemic control. Most likely this is caused by the fact that the PID controller were tuned on the average population. Hence, patients having closer to average physiological parameters will have a tendency to perform better compared to individuals outside the average. Therefore, one may argue that the tuning strategy must be individualized. Additionally, the control strategies implemented did show to perform good in some metrics and bad in other metrics. HbA1c, for example, did not show any risk of elevated mean BG levels. Since HbA1c is defined as a theoretical relation between the MBG, the HbA1c may be higher in an actual patient. Additionally, the usage of HbA1c as a performance metric can only identify if the BG levels are elevated over a longer period of time. For this reason Hb1Ac should be accompanied by other relevant risk metrics. The risk parameter ADRR (including LGBI) did show relevant results as a metric. Referring to Table 6.7 it is noted that the average time spend in the low and high zones are high for both group. Conclusively both groups are generally to be said to be in poor glycemic control. This point is extracted directly from the risk parameter ADRR from Table 6.8. The high average and large standard deviation reveals that a lot of the virtual subjects are in very bad glycemic control. One may additionally argue that due to this large variation in ADRR the paired Z-test may not be well suited due to the fact that the data unlikely is normally distributed, one of the fundamental assumptions of the Z-test. As a consequence one may seek to investigate the performance metrics and relevant statistical measures further.
Essentially, every single performance metric gives a different perspective on the glycemic control and variability of a patient. Hb1Ac are good to investigate if the BG levels are elevated over longer periods of time, MAGE is ideal for tracking glycemic variability and ADRR is good to test the overall glycemic control of the virtual patient. Consequently, it is paramount to have several metrics to evaluate the simulations as each metrics show different perspectives.

While the simulator has shown to be a valuable tool it can be used to more than testing of controllers. To further explore the usage of the simulator we propose another in silico experiment in the subsequent subsection.

6.1.4.1 A conceptual experiment of fast acting insulin

This experiment is based on the same virtual cohort as the one defined in Subsection 3.1.6. The experiment seeks to test an insulin bolus calculator, which is currently prescribed and used by physicians and automated devices, such as the Medtronic Minimed® [35]. The bolus calculator is defined as follows:

$$bolus = \frac{d}{CR}$$

d [g], is the meal intake and CR is the insulin-to-carbohydrate ratio estimated by Algorithm 4. This bolus calculator is fairly simple and we seek to test the bolus calculators performance if fast acting insulin hits the market in the future. We simulate fast acting insulin products by variation in the insulin absorption constant τ_s and the elimination rate of insulin k_e in the Hovorka model. This experiment seeks to test the consequence of using the same bolus when the insulin becomes fast acting. From Figure 6.10 and 6.11 the results of the experiments can be seen, as the maximum and minimum BG versus the average insulin absorption constant $\bar{\tau}_s$. Figure 6.10 gives the experiment for a lower insulin elimination rate, k_e, compared to Figure 6.11. Essentially, the fast acting insulin reduces the maximum BG values for the virtual cohort. An insulin absorption constant at least equal to the glucose absorption constant (i.e. $\tau_s = 40$ minutes) completely removes hyperglycemic events. A lower elimination rate k_e gives a lower minimum BG value (seen by comparing the right plot on Figure 6.10 with Figure 6.11) and may compromise the current bolus calculator - especially for larger meals. Finally, if τ_s is very small (i.e. $\bar{\tau}_s < 30$ min) we see a drop in the minimum BG. These observations may lead to the conclusion that the insulin is absorbed too fast and therefore a pre-meal drop in BG level is observed. Neither way the simulations have shown that one must be particular careful if new insulin products should be introduced on the market. One may consider using more sophisticated bolus calculators to adjust the meal bolus accordingly.

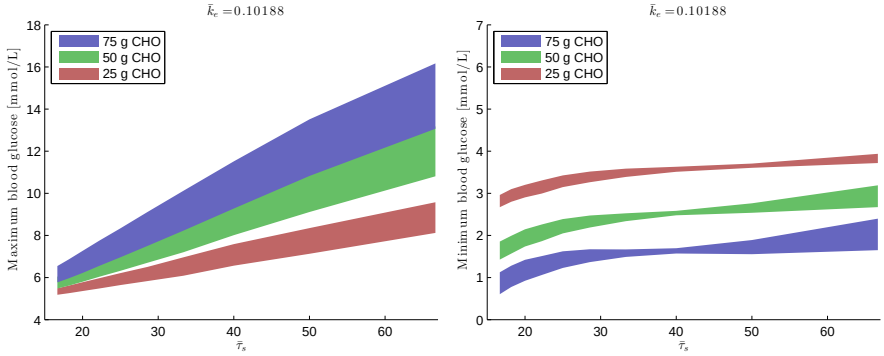


Figure 6.10: *This figure shows the standard deviation range of the maximum and minimum glucose value for the virtual cohort, where* $k_e \sim N(0.1, 0.035^2)$.

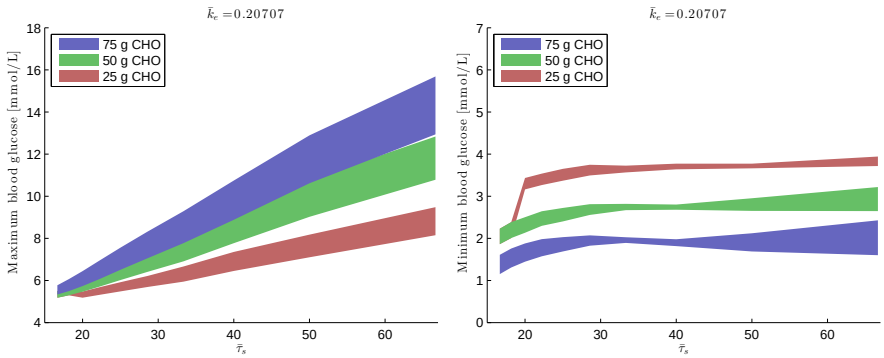


Figure 6.11: *This figure shows the standard deviation range of the maximum and minimum glucose value for the virtual cohort, where* $k_e \sim N(0.2, 0.035^2)$.

6.2 Software Runtime Testing

This test investigates how well the software deals with the implementation of multi-threading. This section first gives a presentation of the hardware used to conduct the experiments. Secondly, the testing methods will be explained followed by a presentation of the results.

6.2.1 Hardware

Two systems has been used in this testing section. The systems chosen is:

- **System one (S1):** Lenovo Thinkpad® T410 with Intel® i5 CPU @ 2.67GHz with 4 cores (2 cores with hyper-threading). 4GB of RAM running windows 7 64-bit and JAVA SE Runtime Environment (build 1.7.0_11b21).
- **System two (S2):** Intel® Xeon® CPU E5-2665 @ 2.40GHz with a total of 8 cores. 64GB of RAM running on Scientific LINUX 6.4 and JAVA SE Runtime Environment (build 1.6.0_17-b04).

Two different hardware essentially ensures that the software will be tested in a consistent manner across hardware and platforms. The next section describes the test protocol in details.

6.2.2 Testing Protocol

The following set of methods and principles were used during the conduction of the experimental performance test:

- The software is compiled in Java 1.6 to ensure compatibility on both S1 and S2.
- JAVA Heap Space and the number of garbage collectors has been maximised on each system. This has been done on S1 by changeing the Java Heap Space in system preferences and on S2 by using the call: `unset JAVA_TOOL_OPTIONS`.

- To reduce simulation variability during runtime tests, the virtual patients have been constructed with the same physiological composition's and meal profiles.
- The trial is conducted in a 14 day period with a sampling time, $T_s = 5$, corresponding to approximately 0.45 mb of data per patient.
- The program runtime has been calculated using Matlab's `tic` and `toc`.
- The runtime includes reading input from csv files and simulations of patients.
- Each runtime is calculated as an average of 3 runs. $\bar{R}_s$ and $\bar{R}_p$ denotes the average of the sequential and parallel runtime respectively.
- The speed up ratio is defined as $S_p = \frac{\bar{R}_s}{\bar{R}_p}$.

6.2.3 Runtime Test Results

Figure 6.12 and 6.13 shows the runtime tests on S1 and S2 respectively. From the top part of Figure 6.12 the black line indicates the average sequential runtime, $\bar{R}_s$, while the blue line indicates the parallel runtime, $\bar{R}_p$, conducted on S1. Figure 6.12 shows that the runtime has been improved by the implementation of multi-threading. Amdahl's law, from (5.3), states a minimum bound on the runtime achievable. Amdahl's law indicates that there exist a difference between $\bar{R}_p$ and the theoretical limit. This phenomenon can also be seen in the lower part of Figure 6.12 indicating the speed up ratio, S_p. The average speed ratio through all runs is $\bar{S}_p = 2.44(\pm 0.47)$. While the theoretical value of the speed up ratio is $S_p^* = 3.88$, given by (5.2). The observed average speed up ratio, $\bar{S}_p$, is approximately 63% of the Amdahl's. Figure 6.12 also shows that the S_p is decreasing as the number of patients increase. An explanation to this phenomenon could be that as the number of threads increases the level of overhead becomes relatively larger hereby forcing the speed up ratio to become lower.

The results generated from S2 are presented in Figure 6.13. Figure 6.13 contains the same elements as the ones presented in Figure 6.12, that is, the average sequential run time, $\bar{R}_s$, the parallel runtime $\bar{R}_p$ and Amdahl's law. From Figure 6.13 a higher performance increase is seen compared to S1. This is intuitively also expected as the processing power is increased with a factor of 2 and the RAM amount is increased with a factor of 16. The average speed up ratio over all runs is $\bar{S}_p = 5.08(\pm 0.68)$. While the speed up ratio given by Amdahl is, $S_p^* = 7.48$ with $p = 8$ this means that the speed up ratio achieved is

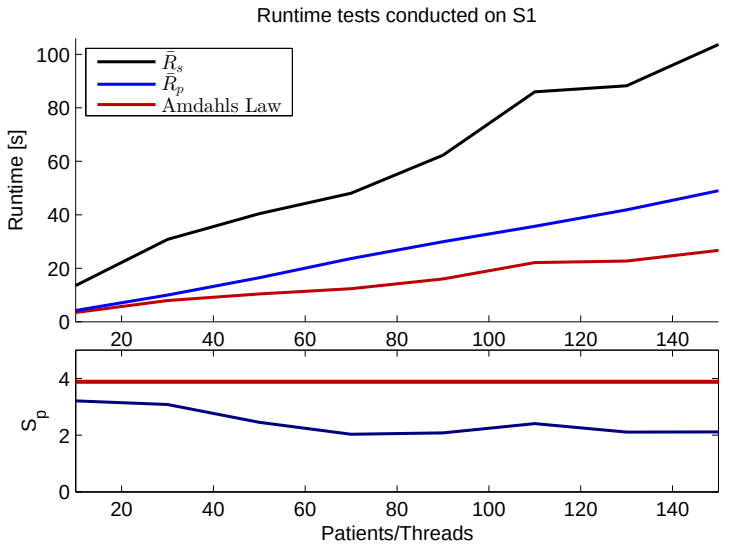


Figure 6.12: *This Figure shows the runtime for S1 as a function of the number of patients. The upper figure represent the total run time while the lower figure represent the speed up ratio S_p. Amdahl's law represent an upper bound on the ratio $S_p^* = 3.88$ with $P = 4$ processors and a lower bound on the runtime.*

approximately 68% of Amdahl's. The speed up ratio does not decrease as the patients increase for S2. This may be due to the extensive level of RAM in S2.

In both S1 and S2 a significant performance improvement were achieved by multi-threading. Even though the theoretical runtime given by Amdahl's law in practice is impossible to reach the runtime improvement is still far from the theoretical given by Amdahl's law [22]. This may indicate that the overhead in the program is consistent between systems. To investigate this further it is interesting to see why the speed up ratio were lower than anticipated. An analysis on the computational workload is given by Table 6.9 performed on the profiling tool Oracle Solaris Studio Performance Analyzer®. A profiler makes it possible to investigate how much computational time is used on the various tasks in a software. The analysis is conducted on S2 and shows the different time consuming task in the sequential and concurrent execution. The import of data indicates that reading from the disc is a relatively larger part of the runtime. This is caused by the fact that reading from the disc cannot be done in a concurrent matter leading to a performance bottleneck. The JVM, which also includes the Java Garbage Collector, indicates that the programming language Java does have a high level of overhead related to its environment. From Table

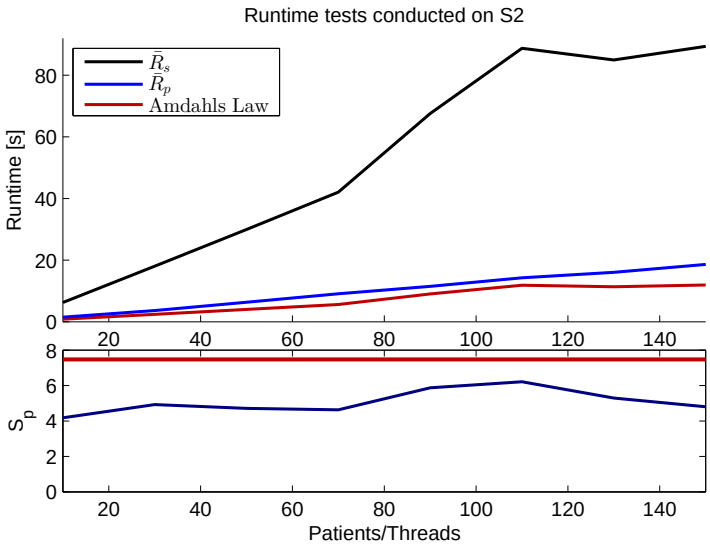


Figure 6.13: *This Figure shows the runtime for S1 as a function of the number of patients. The upper figure represent the total run time while the lower figure represent the speed up ratio S_p. Amdahl's law represent an upper bound on the ratio $S_p^* = 7.48$ with $p = 8$ processors and a lower bound on the runtime.*

6.9 it also observed that the numerical computations are the single largest part of the entire computation load.

6.2.4 Discussion of the Runtime Results

Overall the implementation of multi-threading has shown to give a significant performance increase on the systems tested. The results shows that the implementation of concurrency has a great potential even with an object oriented program as Java. The performance of multi-threading was not as close to the theoretical runtime as anticipated. Table 6.9 identified that $63 - 72\%$ of the total computational time was used on the actual computations while the rest was used on different tasks, including import of data and garbage collecting. This potentially indicates two things. First, that Java has some related overhead to its execution and second, that the software developed may not be fully optimised. While the import of data cannot be executed in a concurrent manner there may be more performance potential in the Java platform, including optimisation of the garbage collector. This however, is out of scope of this the-

Table 6.9: *Percentage of running time used on different tasks performed in a sequential and Multi-threading matter. Profiling has been conducted on S2 and using 30 patients for a 14 day period with a sampling time $T_s = 5$.*

Subject Runtime	**Sequential** $\bar{R}_s = 18.0s$	**Multi-threading** $\bar{R}_p = 3.67s$
Import of data	5%	12%
Numerical Computations	72%	63%
Java Virtual Machine	13%	12%
Other	10%	13%
Total	100%	100%

sis. Java does have a significant portion of overhead related to its environment, which we saw from Table 6.9. This does reduce the overall performance of the software. Together with the import this probably describes the overhead related to multi-threading and why we only achieved around 65% of Amdahl's law.
An interesting further analysis would be to test this software on an even more powerful system than S2. This would identify if the speed up ratio would continue to improve as the number of cores increased even further. This could lay the foundation for further optimization of the program. Especially interesting could be to use another numerical library which makes use of more speed optimised methods. This could also include comparing libraries written in low-level languages such as C to compare the numerical performance to the one achieved in Java.

While Java is a high level, object oriented, platform independent language it does include a large part of overhead. Java was chosen specifically in this thesis due to its high level simplicity, the mathematical library's available and the fact that it could be tested both on Linux and on windows machines unlike a high level object oriented language such as C#. The documentation available for programming in Java is comprehensive and a very solid reason that Java has become preferred programming language for many programmers.

6.3 Summary

In this section two tests of the software were performed. First, a test on a cohort of 50 virtual patients were conducted. The software enabled easy establishment of many different virtual patients with inter- and intra-personal variability. The cohort were tested on two different insulin administration strategies. The software enabled easy assessment of the two strategies on various statistical and risk measurements. Hereby, a Proof-Of-Concept implementation of an AP in

the simulation framework were performed. The entire source code including the simulation runs from this experiment can be located in Appendix F. A runtime test documented the performance improvement gained by the implementation of multi-threading. The test showed a speed up ratio of a factor of 5 on the hardware available. The test also recognised a potential in reducing the runtime of the software even further.

Chapter 7

Closure

This closure gives a perspective of the thesis findings followed by a return to the problem statement. This chapter reflects the authors ideas and thoughts about the project.

7.1 Thesis Perspective

Before taking this software as a "gold standard" of benchmarking of the AP we need to confirm that the software can be descriptive of the actual system. Hence, clinical validation of the software is paramount for the usability of this software. Hovorka et al. describes in [28] how they conducted a clinical comparison with the simulation framework they developed. Hovorka et al. were able to show that simulated closed-loop study were not significantly different from the results obtained from a clinical study (n = 12 adolescence)[28]. We note that this sample is rather small, though the findings very appealing. Hovorka et al. also noted that further validation studies should be undertaken to further asses the affect of variations in the population on the different control algorithms. If the simulation software is a good indication of the actual patient variability it serves as a strong tool in assessing the performance of different AP in a very time- and cost-efficient way. Nonetheless, in silico trials will never fully substitute real life trials as actual human trials are of high importance for the assessment of the

effect, safety and usability of closed loop insulin delivery systems [19, 27, 28, 34, 35].

7.1.1 Future work

The areas of future work can be divided into content development aspects and software optimization aspects. The future work within the content development aspects is actual improvements of the software seen from an AP testing perspective. Within the software optimization aspects is the elements which would increase the usability of the software.

7.1.1.1 Content Development

- **Control Strategies:** As this software seeks to test various control strategies, an adjacent point of future work is the development and implementation of other administration strategies. If the control strategy were originally developed in C this could be integrated in Java using the Java Native Interface or the `MatlabControl` class if developed in Matlab. Since this software serves to test various AP it would be highly relevant to implement a various set of control algorithms. Hereby, the software would be able to test a various set of control strategies on the existing simulation framework. This would make the simulation framework more adequate for future assessment of the AP.

- **Physiological models:** Future projects could look into two elements. First, is the development and additions to the existing model. Second, is the implementation of new models. The physiological model implemented only considered the single-hormone case and a future revision could include a dual-hormone physiological model considering also the dynamics of glucagon. Hereby, the implementation of a dual hormone control strategy could be implemented.

- **Virtual Cohort:** The patient variability is a key challenge for proper glycemic control with the AP. Being able to describe many individuals by a simulation tool serves as a strong platform for testing of closed-loop control systems. The virtual cohort could be expanded either by conducting clinical trials or a literature study. Expanding the virtual cohort such that it included both children and adolescence could be of interest as this would increase prediction horizon of the actual clinical studies.

- **Performance metrics:** Another area of attention could be adding additional performance metrics mapping the BG to a risk measure. From the previous chapter we saw that different performance metrics gave different results and interpretations and conclusions. Having a set of performance metrics is paramount to test for the different risk related elements, that is, hyperglycemia, hypoglycemia and glycemic variation. In other words, the performance metrics identifies consistency of treatment across the cohort in a fast manner.
- **Other AP components:** Other simulation systems available, such as the T1DMS have a various set of components to chose from [12]. Thus, one can test e.g. different CGMs from different manufacturers. Adding different commercial products to the simulation framework could be an interesting addition in the future. Hereby, new equipment and devices could be tested across a cohort and across closed-loop control algorithms. This could also include a failure mode which would consider the failure scenarios of the components.

7.1.1.2 Software Optimization

This subsection goes into details on further areas of improvements related to the software.

- **Graphical User Interface:** Even though the GUI were out out scope of this thesis it is still considered as a relevant addition. A GUI would highly increase the usability of this software. This would make the daily usage more efficient and, thus, researchers with other backgrounds could use this software, making the software more versatile. Additionally, this would free the software from Matlab such that no commercial interference exist.
- **Further runtime improvements:** As the software still have a high runtime, a focus on reducing it further are still of much interest. Two focus areas for further runtime improvements exist. First, is the improvement of the existing algorithms and routines implemented including simplifications, re-factoring and tunings to speed up the existing program without increasing the complexity. An example of this could be a data structure simplification. Second, is the implementation of work distribution between network computers, such as Message Parsing Interface for Java. The current multi-threaded setup does not support distribution of data and workload across machines. However, Java libraries such as JMS or Hazelcast could distribute data and workload across several machines. Nonetheless,

the Message Parsing Interface for Java is still in its early phase and are not part of the native Java libraries. Thus, caution should be on such implementation. Both elements could be interesting extensions to the current software.

7.2 Returning to the Problem Statement

This thesis has shown a feasible implementation of a simulation framework for in silico testing of an AP in Java resembling the usage of an AP in a virtual cohort by:

- Implementing a Proof-Of-Concept AP.
- Creating a system for modelling of a virtual cohort which can be defined to a target population
- Establishing several performance metrics such that easy evaluations across virtual patients can be made.
- Remaining free of intellectual property rights. The use of Matlab for execution is a limitation and must be substituted by a GUI for this to become an entire reality.
- Reducing the runtime significantly by implementation of multi-threading in the software.

Chapter 8

Conclusion

The purpose of this Master's thesis was to develop a software for in silico testing of an AP. The scope was to implement the six principal components of the simulation framework in the programming language Java. Additionally, this implementation should use open source numerical methods to refrain from interference with intellectual property rights in commercial contexts. The implementation of a Proof-Of-Concept AP was successfully conducted. Together with the test procedure described, followed by the performance evaluation the entire simulation framework were developed and tested.

The simulation framework has been presented and implemented. The main components of the system were a mathematical model describing the glucose-insulin dynamics of virtual subjects with T1D, a CGM model, and a CSII delivery model. The virtual cohort were sampled from an actual population with appropriate levels of inter- and intra-variability between subjects. This hereby introduces an experimental playground for future modelling and creation of virtual subjects.

To ensure standardised testing procedure of the AP, several performance metrics were introduced. This includes, but are not limited to, the Average Daily Risk Range (ADRR) to determine the risk of hyper and hypo-glycemia, Mean Amplitude of Glycemic Excursions (MAGE) to asses the variability of a virtual subject, and the HbA1c to asses the mean BG value.

The findings of this thesis showed an example of a fast an easy assessment of a two different Proof-Of-Concept APs. The software successfully conducted simulations, and performance metrics were used to assess the performance across the two APs. The simulations were performed on a virtual cohort (n = 50 adults) with both inter- and intra-individual variability. The tests showed that some performance metrics, particular ADRR and LBGI, were better indicators of risks than others. Therefore, various performance metrics are a key to assessing the AP in the simulation framework for future use.

It were concluded that the virtual cohort could be expanded in the future to include virtual subjects with various physiological compositions. In addition, a future clinical validation of this software is paramount. A clinical assessment would identify if the software is able to describe the actual physiological variability which exist in actual individuals. This would serve as the final acid test for the validity of this software.

To conclude, the concept of testing closed-loop control algorithms in a computer environment based on open source numerical methods has been proven valuable. I am convinced that in silico testing, based on the same concepts from this thesis, will be an essential part of medical device testing in the future.

List of Figures

List of Tables

Appendix A

Steady State for The Hovorka Model

Finding the steady state for the Hovorka Model entails the condition in equation (A.1).

$$\dot{x}(t) = \begin{bmatrix} \frac{D_1(t)}{dt} \\ \frac{D_2(t)}{dt} \\ \frac{S_1(t)}{dt} \\ \frac{S_2(t)}{dt} \\ \frac{Q_1(t)}{dt} \\ \frac{Q_2(t)}{dt} \\ \frac{I(t)}{dt} \\ \frac{x_1(t)}{dt} \\ \frac{x_2(t)}{dt} \\ \frac{x_3(t)}{dt} \end{bmatrix} = \mathbf{0} \tag{A.1}$$

It also entails a steady state condition given some target BG value, r.

$$\frac{Q_1(t)}{V_G} = r \leftrightarrow$$
$$\frac{Q_1(t)}{V_G} - r = 0$$

This leads to a system of 11 equations and 11 unknowns where $d = 0$ and $\theta_k = \theta$:

$$\begin{bmatrix} f(x, u, 0, \theta) \\ \frac{Q_1(t)}{V_G} - r \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix} \tag{A.2}$$

$\mathbf{0}$ indicates the zero vector. The unknowns are the 10 states, x and the insulin dosage u. From Section 3.1 The Hovorka Model the derivation can be made. First, from (A.2) we see that: $Q_1(t) = r \cdot V_G$. This leads to the fact that $D_1(t) = D_2(t) = U_G(t) = 0$.

The derivation of an expression that can be used to find u and x will be performed below. From equation (3.4) we see that

$$0 = u - \frac{S_1(t)}{\tau_S} \Leftrightarrow S_1(t) = u \cdot \tau_S \tag{A.3}$$
$$0 = \frac{S_1(t)}{\tau_S} - \frac{S_2(t)}{\tau_S} \Leftrightarrow S_2(t) = u \cdot \tau_S \tag{A.4}$$
$$\tag{A.5}$$

$Q_1(t)$ and $Q_2(t)$ will be skipped for now. from (3.7) we arange such that;

$$U_I(t) = \frac{S_2(t)}{\tau_S} \Leftrightarrow U_I(t) = u \tag{A.6}$$

Above equation and equation (3.7) leads us to finding that:

$$\frac{U_I(t)}{V_I} - K_e I(t) = 0 \Leftrightarrow \frac{u}{V_I} = K_e I(T) \Leftrightarrow I(t) = \frac{u}{V_I K_e} \tag{A.7}$$

Now, it is possible to find the three remote effects on the insulin level in the plasma by substituting (A.7) to (3.14)

$$K_{b1} I(t) = K_{a1} x_1(t) \Leftrightarrow x_1(t) = \frac{k_{b1} \frac{u}{V_I K_e}}{k_{a1}} \tag{A.8}$$
$$K_{b2} I(t) = K_{a2} x_2(t) \Leftrightarrow x_2(t) = \frac{k_{b2} \frac{u}{V_I K_e}}{k_{a2}} \tag{A.9}$$
$$K_{b3} I(t) = K_{a3} x_3(t) \Leftrightarrow x_3(t) = \frac{k_{b3} \frac{u}{V_I K_e}}{k_{a3}} \tag{A.10}$$

Finally from equation (3.10) it is possible to isolate $Q_2(t)$,

$$Q_2(t) = \frac{x_1(t)Q_1(t)}{k_{12} + x_2(t)} = \frac{k_{a1}^{-1} k_{b1} \frac{u}{V_I k_e} r \cdot V_G}{k_{12} + k_{a2}^{-1} k_{b2} \frac{u}{V_I k_e}} = \frac{r \cdot V_G \cdot k_{b1} \cdot u}{k_{a1} V_I k_e \left(k_{12} + k_{b2} \frac{u}{k_{a2} V_I k_e}\right)} \tag{A.11}$$

and substitute this equation and all of the above into (3.9) which is the function we will use to find the optimal value for u_k, denoted $f(u)$:

$$\begin{aligned} q(u) = &- F_{01}^c - F_R - \frac{r \cdot V_G \cdot k_{b1} \cdot u_k}{V_I \cdot k_e \cdot k_{a1}} \\ &+ k_{12} \cdot \frac{r \cdot V_G \cdot k_{b1} \cdot u_k}{k_{a1} V_I k_e \left(k_{12} + k_{b2} \frac{u}{k_{a2} V_I k_e}\right)} \\ &+ EGP_0 \cdot \left(1 - \frac{k_{b3} \cdot u}{V_I \cdot k_e \cdot k_{a3}}\right) \end{aligned} \tag{A.12}$$

Finding the derivative with respect to u of equation (A.12) function will be needed.

The derivative of equation (A.12) can be found using the product and reciprocal rule for differentiation. The two first elements of equation (A.12) disappears. The third and fifth element is fairly easy and is a linear differentiation. However, the fourth element is non-linear and requires some attention. Two new functions are defined, the upper and lower part of the fraction:

$$v(u) = k_{12} \cdot r \cdot V_G \cdot k_{b1} \cdot u_k \tag{A.13}$$

$$k(u) = \frac{1}{k_{a1} V_I k_e \left(k_{12} + k_{b2} \frac{u_k}{k_{a2} V_I k_e}\right)} \tag{A.14}$$

$$h(u) = v(u) \cdot k(u) \tag{A.15}$$

such that

$$h(u) = k_{12} \cdot \frac{r \cdot V_G \cdot k_{b1} \cdot u}{k_{a1} V_I k_e \left(k_{12} + k_{b2} \frac{u}{k_{a2} V_I k_e}\right)} \tag{A.16}$$

Then each derivative is given by:

$$\frac{dv(u)}{du} = k_{12} \cdot r \cdot V_G \cdot k_{b1} \tag{A.17}$$

Using the reciprocal rule $\frac{d}{du} \cdot \frac{1}{g(u)} = \frac{-\frac{dg(u)}{u}}{g(u)^2}$:

$$\frac{dk(u)}{du} = \frac{-k_{a1} k_{b2} k_{a2}^{-1}}{\left(k_{a1} V_I k_e k_{12} + k_{a1} k_{b2} k_{a2}^{-1} u\right)^2} = \frac{-k_{b2}}{k_{a2} k_{a1} \left(V_I k_e k_{12}^{-1} + k_{b2} k_{a2}^{-1} u\right)^2} \tag{A.18}$$

From the product rule it is given that $\frac{dh(u)}{du} = \frac{dv(u)}{du} \cdot k(u) + \frac{dk(u)}{du} \cdot v(u)$, hence:

$$\begin{aligned}\frac{dk(u)}{du} &= (k_{12} \cdot r \cdot V_G \cdot k_{b1}) \cdot \left(\frac{1}{k_{a1} V_I k_e \left(k_{12} + k_{b2} \frac{u}{k_{a2} V_I k_e} \right)} \right) \\ &+ (k_{12} \cdot r \cdot V_G \cdot k_{b1} \cdot u) \cdot \left(\frac{-k_{b2}}{k_{a2} k_{a1} \left(V_I k_e k_{12}^{-1} + k_{b2} k_{a2}^{-1} u \right)^2} \right) \\ &= \frac{k_{12} \cdot r \cdot V_G \cdot k_{b1}}{k_{a1} V_I k_e \left(k_{12} + k_{b2} \frac{u}{k_{a2} V_I k_e} \right)} + \left(\frac{-k_{b2} \cdot k_{12} \cdot r \cdot V_G \cdot k_{b1} \cdot u}{k_{a2} k_{a1} \left(V_I k_e k_{12}^{-1} + k_{b2} k_{a2}^{-1} u \right)^2} \right)\end{aligned} \tag{A.19}$$

The derivative of the basal insulin injection then becomes.

$$\begin{aligned}\frac{dq(u)}{du} = &- \frac{r \cdot V_G \cdot k_{b1}}{V_I \cdot k_e \cdot k_{a1}} \\ &+ \frac{k_{12} \cdot r \cdot V_G \cdot k_{b1}}{k_{a1} V_I k_e \left(k_{12} + k_{b2} \frac{u_k}{k_{a2} V_I k_e} \right)} \\ &+ \left(\frac{-k_{b2} \cdot k_{12} \cdot r \cdot V_G \cdot k_{b1} \cdot u}{k_{a2} k_{a1} \left(V_I k_e k_{12}^{-1} + k_{b2} k_{a2}^{-1} u \right)^2} \right) \\ &- \frac{EGP_0 \cdot k_{b3}}{V_I \cdot k_e \cdot k_{a3}}\end{aligned} \tag{A.20}$$

Appendix B

UML Sequence diagram notation

Notation for UML sequence diagrams: Note that the term ":" on the left side defines the name of the variable or instance and on right side the data type of the variable.

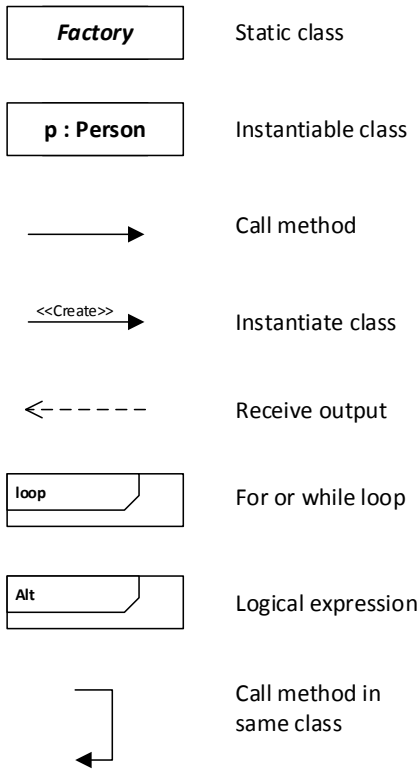


Figure B.1: *UML Sequence diagram notation*

Appendix C

Javadoc

A full Java Documentation page has been made for this software to ensure documentation of software for future usage. Due to the extend of the Java documentation please refer to the disc attached to this report.

Appendix D

Complete User manual

This appendix present a complete user manual to the different functionalities in the software.
The first thing that need to be considered before executing the program is the configuration file system.

Configuration System

The configurations systems is simply a file names `configuration.xml` where the software will load some essential parameters. The configuration file is an important element of this software since it allows you to change all sorts of things, including but not limited to the directory of export, the solver to use, to print to csv etc. The file content looks as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>Configuration file for the AP system</comment>
<entry key="directory">[your directory]/matlab/Patients/Patient</entry>
<entry key="export_directory">[your directory]/matlab/ExportData/</entry>
```

```
<entry key="final_export_directory">[your directory]/matlab/ExportData2/</entry>
<entry key="load_directory">[your directory]/matlab/Patients/</entry>
<entry key="multithread">Y</entry>
<entry key="tocsv">N</entry>
<entry key="model">H</entry>
<entry key="solver">dp54</entry>
<entry key="minstep">1.0e-6</entry>
<entry key="maxstep">1.0</entry>
<entry key="absolutetolerance">1.0e-6</entry>
<entry key="relativetolerance">1.0e-6</entry>
<entry key="eulerstep">1.0e-1</entry>
<entry key="newton_basal_min">0</entry>
<entry key="newton_basal_max">15</entry>
<entry key="newton_basal_startguess">5</entry>
<entry key="newton_maxeval">1000</entry>
<entry key="target">5.0</entry>
<entry key="PID">Y</entry>
<entry key="kgain">-0.25</entry>
<entry key="ngain">100</entry>
<entry key="Ti">349</entry>
<entry key="Td">0</entry>
<entry key="Kd">1</entry>
<entry key="Tt">5</entry>
<entry key="mealBolus">820</entry>
<entry key="tuningdays">3</entry>
<entry key="feedforward">Y</entry>
</properties>
```

The most important elements are `directory` - location and name of input folders, `export_directory` - location of output folders, `final_export_directory` - location of final output folders and `load_directory` - the directory for patients. `directory` specifies the folder to locate the patients from and the last part of the name `Patient` indicates the filenames for the patients. `export_directory` gives the directory for which the transformed files should be loaded to. `final_export_directory` gives the output folder for the simulation and finally, `load_directory` indicates what location patients should be imported from. Other important keys are the `tocsv` which decides whether the simulation output should be printed to a CSV output. If `N` the output will be stored in memory. The keys `PID` to `feedforward` denotes elements of the PID controller which can be adjusted *on-the-go* for the specific cohort.

The input part of the program serves as main purpose to transform data. From a user-friendly input denoted in table 5.1 and 5.2 to the output ready for simu-

lation as shown in table 5.3.

Input

The input should be in the form as table 5.1 and 5.2 and should be located in each separate folder. One csv file corresponds to a single day. So if one wishes to simulate 10 days one could define parameters and meals for 10 days. However, if only 5 days of csv files are defined, the csv files will simply be reused. The meal profiles files (meal000.csv) should be arranged as follows:

```
Time;Duration;CHO
7:0;20;40.0
10:0;20;40.0
12:0;20;40.0
```

And the parameter file (parameter000.csv):

```
Time;EGP0;F01;K12;Ka1;Ka2;Ka3;SI1;SI2;SI3;Ke;VI;VG;TD;TS;Ag
7:0;0.0161;0.0097;0.066;0.006;0.06;0.03;0.00512;8.2E-4;0.052;
0.138;0.12;0.16;40.0;55.0;0.8
10:0;0.0161;0.0097;0.066;0.006;0.06;0.03;0.00512;8.2E-4;0.052;
0.138;0.12;0.16;40.0;55.0;0.8
```

Since there is no GUI the file input is only error checked in a very limited fashion. One should therefore be very carefull when defining the values in the files. A final remark is the file names. This is very important since sorting algorithms will be used to sort the files. Hence, if one seeks to have a certain number of files in a particular order it is very important to add numbers in the name such that ordering can take place. If one seeks to create 14 meal files one should name the files as follows:

```
meal00.csv
meal01.csv
...
meal14.csv
```

The important thing to remember is that all numbers is the files must be of similar length. Hence, file meal01.csv must not be named meal1.csv - otherwise the sorting algorithm consequently will fail.

From here one can now execute the program:

Execution

In general there are two ways to execute the program. Either through the terminal or through Matlab. Executing from the terminal would be done as follows:

```
java -jar Trial.jar [input]
```

Here input indicates the optional input parameters that can be requested. The input parameters is chronologically as follows: Number of patients to simulate, the step size, the number of days, runner, start time (HH MM) and end time (HH MM). The runner is a string which can be either be "C" for collect or "R" for random indicating whether to collect a population or create one randomly. The execution call from the terminal could look something like this:

```
java -jar Trial.jar 10 5 5 C 06 30 07 00
```

Simulating 10 patients in 5 days with a step size of 5 that will collect all information from predefined files starting at 06:30 today ending 07:00 5 days ahead.

Besides from terminal execution one can simply execute from Matlab. This is done in following manner:

```
> javaaddpath('..\binJava\Trial.jar')
> import domain.*;
> import functionality.*;
> import inputOutput.*;
```

From here two calls exist, either collecting a pre-defined population or creating one randomly:

```
% Initialize:
stepSize            = 5;
startHours          = 00;
startMinutes        = 00;
```

```
endHours            = 23;
endMinutes          = 59;
numberOfPatients    = 50;
days = 35;
BW = 74.9*ones(numberOfPatients,1)+14.4*normrnd(0,1,[1 numberOfPatients])';


% If you want to change the parameter input also remember to do this in
% GenerateParameters
par = [java.lang.String('F01');
    java.lang.String('EGP0') ;
    java.lang.String('K12');
    java.lang.String('SI1');
    java.lang.String('SI2');
    java.lang.String('SI3');
    java.lang.String('ka1');
    java.lang.String('ka2');
    java.lang.String('ka3');
    java.lang.String('Ke');
    java.lang.String('VI');
    java.lang.String('VG');
    java.lang.String('TauD');
    java.lang.String('TauS');
    java.lang.String('AG')];

% Notify java program on the parameters to change
p = AddToArray(par);
parameters = GenerateParameters(p,numberOfPatients,days,stepSize);

% Execute
Execute.CollectPopulation(numberOfPatients, stepSize,days,startHours,startMinutes
Simulator.execute(numberOfPatients,stepSize,BW);

% Save Metrics
p1  = PMetrics(1:numberOfPatients,SaveMethod,Metrics,stepSize);
```

The empty square in the end is important. Assume I would like to collect a predefined population but I would like to vary some parameters across the population. This would be done in following manner, assuming that the insulin sensivity would vary over time as a function of $sin(t)$:

```
parameters = ['SI1'];
par = AddToArray(parameters);
```

```
par.addToArray(sin(t));

Execute.CollectPopulation(numberOfPatients, stepSize,days,...
startHours,startMinutes,endHours,endMinutes,BW,par);
```

The program will execute and print following message:

```
==================================TRIAL=====================================
==================================INPUT=====================================

 MODEL:    HOVORKA
 Step size:    5 minutes
 Number of patients:  10
 The trials starts:  [start time]
 The trials end:  [end time]
 Total minutes:  [Total number of minutes]
 Total steps:    [Total number of steps]

==============================PROGRAM STARTS================================
PROGRAM COMPLETED SUCCESFULLY TIME[s]: 2.481
```

From here the simulator is executed and following message are received.

```
==============================PROGRAM REPORT================================
     Body weight:
  Mean:  75.0kg  +-(0.0kg).
  High:  75.0kg.
low:  75.0kg.

     Addr:
  Mean:  260.0249787022015  +-(0.0).
  High:  260.0249787022015
low:  260.0249787022015

     Time in zone:
  Mean:  1.1852260198456448%  +-(0.0%)
  High:  1.1852260198456448%
  Low:  1.1852260198456448%

     Time below 4 mmol/L:
  Mean:  0.0%  +-(0.0%)
```

Table D.1: *Selected evaluation calls with input and output.*

Call [`Metrics.`]	Input	output
ADRR	Blood Glucose	[*LBGI HBGI ADRR*]
inZone	Blood Glucose, min, max	*TIZ*
belowZone	Blood Glucose, min	*TB*
aboveZone	Blood Glucose, max	*TA*
hb1ac	Blood Glucose	Hb1Ac
auc	Blood Glucose, step size	AUC
MAGE	Blood Glucose, step size	MAGE

```
  High:  0.0%
  Low:  0.0%

     Time above 10 mmol/L:
  Mean:  98.81477398015436%  +-(0.0%)
  High:  98.81477398015436%
  Low:  98.81477398015436%

==============================END OF PROGRAM==============================
```

Output

In the configuration file one can specify to print results to csv. One will get an output file in the format:

```
timestamp;mean;sigma;median;low;high;adrr;mage;cr;hb1ac;t;bg;d;u;cgm
```

The output files will be save into the folder specified in the configuration system, as described in D. Each output file represent one patient and the output will be in the following format (export0001.csv). If one does not wish to print to csv the metrics can be called directory from memory. One can see this as a library addition and the calls can be seen from the table below:

Appendix E

Software Exceptions and solutions

During testing a few error scenarios has been experienced. This appendix serves to give solutions to the most common issues.

FileNotFound Exception

This exception is thrown when the program cannot create or locate the folders for import and export of the CSV files. Solutions is provided below:

Be sure that the directory path in the configuration.xml file has been entered correctly. Close Matlab and move files into another directory and retry running the file, while updating the configuration file to the correct directory.

IOException: Too many open files

This is a problem on LINUX related machines. The error occurs due to a file limit in LINUX. Simply type following into the terminal and the problem should be solved:

```
ulimit - n 32768
```

OutOfMemoryError: Java heap space

Simply allow the JVM enviroment more memory. If the program is executed from the terminal do following:

```
java -Xms1200m -Xmx1300m -jar patient_Simulator.jar
```

This will set the memory to a range between 1.2 and 1.3Gb. Otherwise, if the program is executed from Matlab go to preference → Java Heap Memory and set the scale to the maximum value. Note that a restart of Matlab is needed.

JodaTimeException: Joda Time fails to load

If JodaTime fails to load, simply close Matlab and try again.

java.lang.NullPointerException

In the case, typically when a file is open or no files has been defined the software will throw a null PointerException as shown below:

```
java.lang.NullPointerException

at inputOutput.CreateCsv.createCSV(CreateCsv.java:204)

at domain.Factory.executeTrial(Factory.java:219)

at functionality.Execute.CreateRandomPopulation(Execute.java:59)
```

The simple solution is to make sure that all CSV files that the program need access to is saved, closed and stored in the correct location.

Appendix F

Source Code and Experiments

Due to the extend of the source code of the software please refer to the disc attached to this report.

Appendix G

Apache Commons Mathematics Source Code

Due to the extend of the ACM library please refer to the disc attached to this report.

Bibliography

[1] Taming tiger: Loading properties from xml. https://www.ibm.com/developerworks/library/j-tiger02254/. Accessed: January 2014.

[2] Accu-chek spirit insulin pump. https://www.accu-chekinsulinpumps.com/ipus/products/insulinpumps/index.html. Accessed: February 2014.

[3] Aida type 1 diabetes simulator. http://www.2aida.org/aida/intro.htm. Accessed: February 2014.

[4] Apache commons math. http://commons.apache.org/proper/commons-math/, . Accessed: January 2014.

[5] Apache license. http://www.apache.org/licenses/LICENSE-2.0, . Accessed: January 2014.

[6] Dexcom g4 platinum. http://www.dexcom.com/dexcom-g4-platinum. Accessed: February 2014.

[7] Jdrf artificial pancreas project consortium. http://jdrfconsortium.jaeb.org/ViewPage.aspx?PageName=Home, .

[8] Artificial pancreas project research. http://jdrf.org/research/treat/artificial-pancreas-project/, . Accessed: February 2014.

[9] Joda-time - java date and time api. http://www.joda.org/joda-time/. Accessed: January 2014.

[10] Maxsim2 – interactive pkpd simulation. http://www.maxsim2.com/. Accessed: February 2014.

[11] Minimed® 530g with enlite®. http://www.medtronicdiabetes.com/treatment-and-products/minimed-530g-diabetes-system-with-enlite, note = Accessed: February 2014.

[12] The epsiolon group - type 1 diabetes simulator. http://tegvirginia.com/solutions/t1dms/. Accessed: February 2014.

[13] Yang Kuang Athena Makroglou, Jiaxu Li b. Mathematical models and software tools for the glucose-insulin regulatory system and diabetes: an overview. *Applied Numerical Mathematics*, 56, 2006.

[14] Peter A. Baghurst. Calculating the mean amplitude of glycemic excursion from continuous glucose monitoring data: An automated algorithm. *DIABETES TECHNOLOGY and THERAPEUTICS*, 13(3):296–302, 2011. ISSN 15209156, 15578593. doi: 10.1089/dia.2010.0090.

[15] Donald Bell. Uml basics: The sequence diagram. *IBM Developerworks*, 2004. URL http://www.ibm.com/developerworks/rational/library/3101.html.

[16] Dimitri Boiroux. Model predictive control algorithms for pen and pump insulin administration. September 2012.

[17] WHO Media Centre. Diabetes fact sheet n312, December 2013.

[18] C. Cobelli, E. Renard, and B. Kovatchev. Artificial pancreas: past, present, future. *DIABETES -NEW YORK-*, 60(11):2672–2682, 2011. ISSN 0012-1797.

[19] Chiara Dalla Man, Boris P. Kovatchev, Marc Breton, and Claudio Cobelli. In silico preclinical trials: A proof of concept in closed-loop control of type 1 diabetes. *Journal of Diabetes Science and Technology*, 3(1):44–55, 2009. ISSN 19322968.

[20] Edward R. Damiano, Firas H. El-Khatib, Hui Zheng, David M. Nathan, and Steven J. Russell. A comparative effectiveness analysis of three continuous glucose monitors. *Diabetes Care*, 36(2):20251–259, 2013. ISSN 01495992, 19355548. doi: 10.2337/dc12-0070.

[21] J.R. Dormand and P.J. Prince. A family of embedded runge-kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26, 1980. ISSN 0771050x.

[22] Victor Eijkhout. *Introduction to High-Performance Scientific Computing.* Texas Advanced Computing Center Research Office Complex 1.101 J.J. Pickle Research Campus, Building 196 10100 Burnet Road Austin, Texas 78758-4497, first edition edition, 2011.

[23] John Jian B.S. Firas H. El-Khatib, Ph.D. and Ph.D. Edward R Damiano. Adaptive closed-loop control provides blood-glucose regulation using dual subcutaneous insulin and glucagon infusion in diabetic swine. *Journal of Diabetes Science and Technology*, 1(2), March 2007.

[24] S. Hodgson, N. Dunstan, and I. Fris. Extended semaphore operations. *Concurrency: Practice and Experience*, 12(15):1495–1509, 2000. ISSN 1096-9128. doi: 10.1002/1096-9128(20001225)12:15<1495::AID-CPE541>3.0.CO;2-F. URL `http://dx.doi.org/10.1002/1096-9128(20001225)12:15<1495::AID-CPE541>3.0.CO;2-F`.

[25] Ludovic J Chassin Ulrich Haueter Massimo Massi-Benedetti Marco Orsini Federici Thomas R Pieber Helga C Schaller Lukas Schaupp Thomas Vering Hovorka, Valentina Canonico and Malgorzata Wilinska. Nonlinear model predictive control of glucose concentration in subjects with type 1 diabetes. *PHYSIOLOGICAL MEASUREMENT*, 25, 2005.

[26] Roman Hovorka, Fariba Shojaee-Moradie, Paul V. Carroll, Ludovic J. Chassin, Ian J. Gowrie, Nicola C. Jackson, Romulus S. Tudor, A. Margot Umpleby, and Richard H. Jones. Partitioning glucose distribution/-transport, disposal, and endogenous production during ivgtt. *American Journal of Physiology. Endocrinology and Metabolism*, 282(5):E992–1007, 2002. ISSN 01931849, 15221555.

[27] Roman Hovorka, Ludovic J. Chassin, and Malgorzata E. Wilinska. In silico testing-impact on the progress of the closed loop insulin infusion for critically ill patients project. *Journal of Diabetes Science and Technology*, 2(3): 417–423, 2008. ISSN 19322968.

[28] Roman Hovorka, Janet M. Allen, David B. Dunger, Carlo L. Acerini, Ludovic J. Chassin, and Malgorzata E. Wilinska. Simulation environment to evaluate closed-loop insulin delivery systems in type 1 diabetes. *Journal of Diabetes Science and Technology*, 4(1):132–144, 2010. ISSN 19322968.

[29] John Bagterp Jørgensen. The artficial pancreas - diabetes and control, December 2013.

[30] Eric S. Kilpatrick, Alan S. Rigby, and Stephen L. Atkin. Variability in the relationship between mean plasma glucose and hba(ic): Implications for the assessment of glycemic control. *CLINICAL CHEMISTRY*, 53(5):897–901, 2007. ISSN 00099147, 15308561. doi: 10.1373/clinchem.2006.079756.

[31] B. P. Kovatchev, E. Otto, D. Cox, L. Gonder-Frederick, and W. Clarke. Evaluation of a new measure of blood glucose variability in diabetes. *DIABETES CARE -ALEXANDRIA VA-*, 29(11):2433–2438, 2006. ISSN 0149-5992.

[32] Boris Kovatchev and Marc Breton. Analysis, modeling, and simulation of the accuracy of continuous glucose sensors. *Journal of Diabetes Science and Technology*, 2(5):853–862, 2008. ISSN 19322968.

[33] Linde Wittmeyet-Koch Lars Eldén and Hans Bruun Nielsen. *Introduction to Numerical Computation*. www.studentlitteratur.se, Lund, Sweden, first edition edition, 2004.

[34] L. Magni, D. M. Raimondo, C. Dalla Man, G. De Nicolao, B. Kovatchev, and C. Cobelli. Model predictive control of glucose concentration in type i diabetic patients: An in silico trial. *BIOMEDICAL SIGNAL PROCESSING AND CONTROL*, 4(4):338–346, 2009. ISSN 17468094, 17468108. doi: 10.1016/j.bspc.2009.04.003.

[35] C. D. Man, F. Micheletto, D. Lv, M. Breton, B. Kovatchev, and C. Cobelli. The uva/padova type 1 diabetes simulator: New features. *Journal of Diabetes Science and Technology*, 8(1):26–34, 2014. ISSN 19322968. doi: 10.1177/1932296813514502.

[36] Jeff Meisel. Multithreaded programming. *EE: Evaluation Engineering*, 46(12):12 – 17, 2007. ISSN 01490370. URL `http://search.ebscohost.com.globalproxy.cvt.dk/login.aspx?direct=true&db=afh&AN=27898088&site=ehost-live`.

[37] C. L. Rohlfing, H-M Wiedmeyer, R. R. Little, J. D. England, A. Tennill, and D. E. Goldstein. Original articles - epidemiology/health services/psychosocial research - defining the relationship between plasma glucose and hba1c: Analysis of glucose profiles and hba1c in the diabetes control and complications trial. *Diabetes Care*, 25(2):275, 2002. ISSN 01495992, 19355548.

[38] Steven J. Russell, Firas H. El-Khatib, David M. Nathan, Kendra L. Magyar, John Jiang, and Edward R. Damiano. Blood glucose control in type 1 diabetes with a bihormonal bionic endocrine pancreas. *DIABETES CARE*, 35(11):2148–2155, 2012. ISSN 01495992, 19355548. doi: 10.2337/dc12-0071/-/DC1.

[39] J.W. Schmidt and R.E. Taylor. *Simulation and analysis of industrial systems*. Richard D. Irwin, 1970.

[40] Signe Schmidt, Dimitri Boiroux, Anne Katrine Duun-Henriksen, Laurits Frøssing, Ole Skyggebjerg, John Bagterp Jørgensen, Niels Kjølstad Poulsen, Henrik Madsen, Sten Madsbad, and Kirsten Nørgaard. Model-based closed-loop glucose control in type 1 diabetes: the diacon experience. *Journal of Diabetes Science and Technology*, 7(5):1255–1264, 2013. ISSN 19322968.

[41] Nir Shavit, Maurice Herlihy, and Moran Tzafrir. Hopscotch hashing. *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5218:350–364, 2008. ISSN 03029743, 16113349. doi: 10.1007/978-3-540-87779-0_24.

[42] John Walsh. Introduction to pumping - starting and success. 2007.

[43] Björn Wittenmark, Karl Johan Åström, and Karl-erik Årzén. Computer control: An overview. 2009.